

Beginning BASIC

with the
ZX Spectrum

Also for the
ZX Spectrum +



Beginning BASIC with the ZX Spectrum

Macmillan Microcomputer Books

General Editor: Ian Birnbaum (General Adviser (Microelectronics in Education),
Education Department, Humberside County Council)

Advanced Graphics with the Acorn Electron

Ian O. Angell and Brian J. Jones

Advanced Graphics with the BBC Model B Microcomputer

Ian O. Angell and Brian J. Jones

Assembly Language Programming for the Acorn Electron

Ian Birnbaum

Assembly Language Programming for the BBC Microcomputer (second edition)

Ian Birnbaum

Using Your Home Computer (Practical Projects for the Micro Owner)

Garth W. P. Davies

Beginning BASIC with the ZX Spectrum

Judith Miller

Using Sound and Speech on the BBC Microcomputer

Martin Phillips

Beginning BASIC with the ZX Spectrum

Judith Miller

*Convent of the Sacred Heart
Woldingham*

M
MACMILLAN

© Judith Miller 1985

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission

Dedicated to my parents

Cartoons (in text and on cover)
by Keith A. Miller

First published 1985 by
Higher and Further Education Division
MACMILLAN PUBLISHERS LTD
Houndmills, Basingstoke, Hampshire RG21 2XS and London
Companies and representatives throughout the world

Typeset in Great Britain by
RDL ARTSET LTD, Sutton, Surrey
Printed in Great Britain by
Camelot Press Ltd, Southampton

British Library Cataloguing in Publication Data
Miller, Judith

Beginning BASIC with the ZX Spectrum—
(Macmillan microcomputer books)
1. Sinclair ZX Spectrum (Computer)—Programming
2. Basic (Computer program language)
I. Title
001.64'24 QA76.8.S625

ISBN 0-333-37995-0



728544

Contents

<i>Preface</i>	<i>ix</i>
<i>Using the SPECTRUM + keyboard</i>	<i>xi</i>
<i>Introduction</i>	<i>1</i>
Unit 1: Using the keyboard Keyword mode, Letter mode, Capitals mode, Delete, Shifted keys, Upper and lower case letters	3
Unit 2: Correcting mistakes Moving the cursor to the left and the right	7
Unit 3: Making a program RUN, NEW, LIST	9
Unit 4: More about numbering lines	11
Unit 5: Editing a program Moving the cursor up and down	13
Unit 6: Extended mode. Using the cassette recorder LOAD, SAVE, VERIFY	15
Unit 7: Testing your knowledge — questions on the Introduction and Units 1–5	18
Unit 8: Numeric or simple variables LET	21
Unit 9: String variables Concatenation	25
Unit 10: Extracting parts of strings	29
Unit 11: Using the computer as a calculator	32
Unit 12: Controlling the print position Print zones, commas and semicolons, TAB	37
Unit 13: Beginnings and ends of programs REM, STOP	41
Unit 14: INPUT statement Input prompts	42

Unit 15: Program descriptions – algorithms Flowcharts	46
Unit 16: Loops – using GOTO Using the colon for multistatements	51
Unit 17: Comparisons IF-THEN, Equals and not equals	56
Unit 18: More comparisons Greater and less than, greater or equal to, less than or equal to, AND, OR	65
Unit 19: Adding a counter	77
Unit 20: FOR-NEXT loops	82
Unit 21: READ and DATA statements	92
Unit 22: Nested loops	99
Unit 23: Graphics using PRINT statements, colour and moving graphics Graphics mode, PRINT AT, Colour – BORDER, PAPER, INK, Moving graphics – PAUSE, INKEYS, FLASH, BRIGHT, INVERSE	105
Unit 24: Arrays – DIM statement One-dimensional arrays, Two-dimensional arrays	116
Unit 25: Subroutines GOSUB and RETURN	132
Unit 26: Making music BEEP	140
Unit 27: Some useful functions – RND, RAND and INT	150
Unit 28: High-resolution graphics – PLOT, DRAW, CIRCLE and INVERSE	155
Unit 29: User-defined characters BIN, POKE, USR	170
Unit 30: Some interesting string functions LEN, CHR\$, CODE, VAL	173
Unit 31: Using the printer LPRINT, LLIST, COPY	177
Solutions	179

Preface

The aim of this book is to teach programming in the BASIC language, as used by the ZX Spectrum, to the complete beginner. It is not an 'O' level or C.S.E. textbook, although it does cover practically all the programming skills required at those levels. Mathematics has been kept to a minimum and therefore arithmetic functions such as ABS, EXP, LN, PI, SGN and SQR together with trigonometrical functions SIN, COS, TAN, ASN, ARC and ATN have not been included.

Each unit is written so that it relies only on information already covered in previous units. This means that if the reader works through each unit in turn, he or she will not need to have any knowledge other than that already encountered in earlier units. If, however, the reader covers the units out of order, then difficulties may be experienced unless he or she has previous experience.

The book is best used with a microcomputer at hand as it is recommended that the sample programs are typed in. The program listings are direct copies of the screen, thus providing an easy check that the program has been correctly entered. Each unit gives plenty of opportunity to practise the skills just acquired and solutions to the activities are included at the back of the book. It should be remembered, however, that in programming there may often be several solutions to a problem and therefore if the reader's program works successfully it is just as acceptable as the solution given.

Woldingham, 1984

JUDITH MILLER

Using the ZX Spectrum + Keyboard

Although this book is based on the standard version of the ZX Spectrum, all of the programs and references to BASIC apply equally to the ZX Spectrum +.

If you are using a ZX Spectrum +, there will be minor variations in the steps to follow to obtain certain of the symbols on the keys. Therefore, please disregard the instructions in the text of this book on how to obtain certain 'Extended Mode' and other functions; instead follow the instructions in the *Spectrum + User Guide* or the summary below. You will find that, in all cases, the modifications will make it much easier to enter and manipulate information from the keyboard.

The ZX Spectrum + is essentially a ZX Spectrum, provided with a real keyboard. All software that works on the ZX Spectrum will be compatible with the Spectrum +, and interfacing with other items should be unchanged.

The obvious difference that the user will encounter is that of the feel and use of the full-travel keyboard, which will make keying-in of material much easier.

Earlier criticisms of the keyboard colouring has been taken into account, and the positioning of many of the legends has been subtly altered to increase readability.

There are several differences between the old and new keyboards — 18 extra keys are added to the original 40 keys and, for the first time, there is a RESET button and a conventional space bar.

The RESET button, situated on the right-hand side of the computer, should be used with care — its effect is to switch the computer off and then on, thus losing any program or data in the computer's memory. You will probably want to use it only when the computer has 'crashed' — that is, when the computer fails to respond to any of your instructions.

The main difference between using the Spectrum and the Spectrum + lies in the location and pressing of the following keys: 'EXTEND MODE', 'GRAPH', 'DELETE', 'EDIT', 'SYMBOL SHIFT', 'CAPS SHIFT', 'CAPS LOCK', and 'SPACE'. The operation and principles of use remain unchanged.

See pages 8, 20 and 21 of the *ZX Spectrum + User Guide* for full information on the use of the keys.

Other changes requiring specific mention are:

1. The 'arrow keys' which are used for moving the cursor when editing or when playing many games have been moved from the number keys, and are to be

- found on either side of the space bar — with 'up' and 'down' to the right, and 'left' and 'right' to the left.
2. A welcome addition is the provision of a separate key to provide the quotation marks — which are used at the beginning and end of most PRINT statements.
 3. 'DELETE', 'EDIT', 'CAPS LOCK' and 'GRAPH' together with the ',', and '.' keys each now get a key of their own, and 'CAPS SHIFT' and 'SYMBOL SHIFT' have a key on each side of the keyboard.

How to obtain the different functions on each key

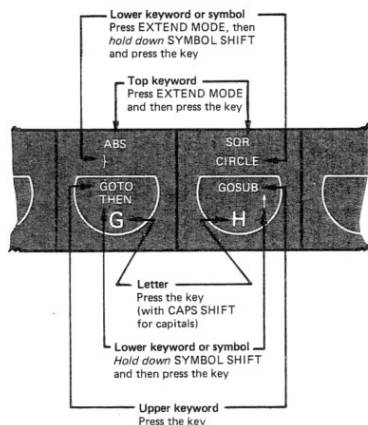
Letter keys

These keys generally have five separate legends on them. On the moving part of the key you will find three items — the letter itself, and either one word and a symbol, or two words.

Just above the moving part of the key, you will find two words or abbreviations.

To obtain the different words and symbols you need to press one or two other keys at the same time as you press the key on which the item is found.

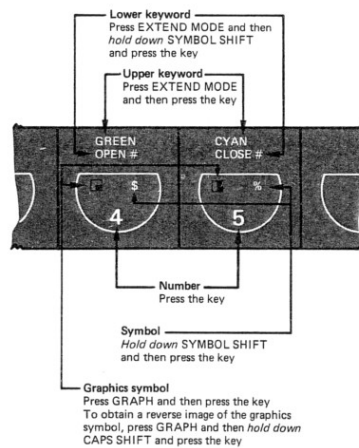
A couple of sample keys are shown below.



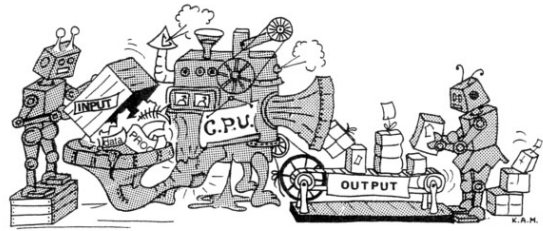
Note that the **letter** can only be obtained when the screen cursor is an 'L', and that **keywords** can only be obtained when the cursor is a 'K'.

Number Keys

Number keys operate in much the same way, and a couple of examples are shown here.

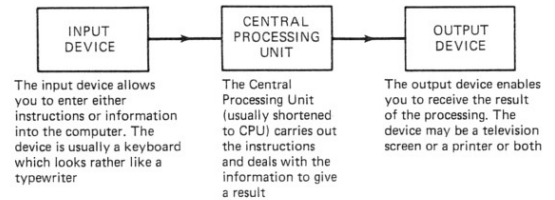


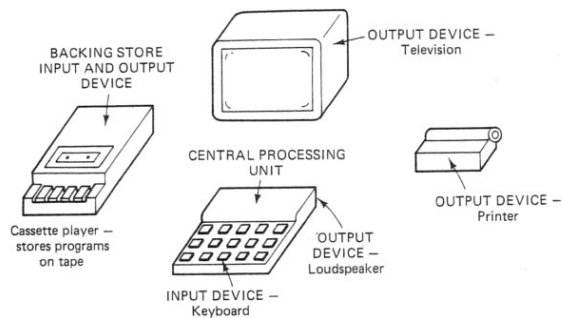
Introduction



The Microcomputer System

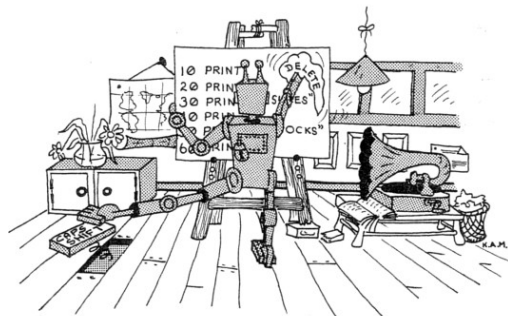
The system is composed of three main parts as illustrated below:





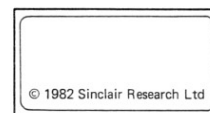
A computer is a machine that enables us to solve certain kinds of problem. It uses symbols or *characters* such as are used in everyday life; for example, the letters of the alphabet, the numbers 0 to 9, punctuation marks and some special characters such as +, -, < and *. The ZX Spectrum uses a language called BASIC which stands for *Beginners' All-purpose Symbolic Instruction Code*.

Unit 1: Using the keyboard



Turn on the computer.

After a few seconds the screen will look like this:



Press the key marked ENTER.

A flashing **K**, called the *cursor*, appears on the screen. It is a signal from the computer that it wants something to do; it is asking for a command. A command must begin with a *keyword*. Keywords are written in white on all the letter keys.

For example

letter A has the keyword NEW
letter P has the keyword PRINT
letter R has the keyword RUN

When the cursor is a **K** the computer is telling you that it is in *keyword mode* and if any letter is pressed a keyword will appear on the screen, not a letter. If you want the computer to print something you must begin the command with the keyword PRINT found on key P.

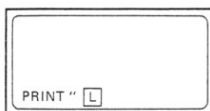
Press PRINT. (Notice that the whole word appears; on many computers the keywords have to be typed in.)

(If you press the wrong word remove it by pressing CAPS SHIFT (bottom left-hand corner) and DELETE (top right-hand corner) *at the same time*.)

The cursor has now changed to a flashing **L** which is a signal from the computer that it wants to know what it is to print. The computer wants some letters or other characters to print and is now in *letter mode*. The Spectrum will print letters (text) only if they begin with inverted commas or quote marks ("). These are also on key P and are obtained by pressing SYMBOL SHIFT and key P *at the same time*.

Press SYMBOL SHIFT and P at the same time.

The screen will now look like this:



If you look at the keys you will notice that most have words or symbols written *on them* in red. For example

key M has a full stop or decimal point
key N has a comma
key X has a pound sign

Like the inverted commas these can only be obtained by pressing SYMBOL SHIFT at the same time as the key required. For convenience, in future, these red symbols will be referred to as *shifted keys*.

Type in a short line of letters.

Inverted commas are also needed at the end of the line.

Press SYMBOL SHIFT and P.

Now the computer is ready to obey the command.

Press ENTER.

Your line of letters now appears at the top of the screen; the computer has obeyed your command. The computer also gives you a message: 0 OK, 0 = 1. This means that everything is OK; there are no mistakes and one line has been executed.

Press ENTER.

The letters disappear and the flashing **K** returns to the screen. The computer is back in *keyword mode*.

The Spectrum can print small letters, as you have already seen, called *lower case letters* or capital letters called *upper case letters*. When the keys are pressed lower case letters are automatically printed. To obtain upper case letters CAPS SHIFT must be held down while the required letter is pressed.

TYPE: PRINT " (remember inverted commas are SYMBOL SHIFT and P).

TYPE: The alphabet using alternate upper and lower case letters. (If you make a mistake remove the wrong letter by pressing CAPS SHIFT and DELETE.)

At the end of the alphabet finish with inverted commas.

Press ENTER.

The alphabet will appear at the top of the screen and the same message as before will be written at the bottom of the screen.

Press ENTER.

When writing a sentence the words need to be spaced out correctly; this is achieved by using the key marked BREAK/SPACE. Practise this by typing in the next sentence.

TYPE: PRINT " I am learning to program on a ZX SPECTRUM using the language BASIC "

Press ENTER.

When writing sentences it is possible to use the correct punctuation marks. For example

- (.) comma — shifted key N
- (;) semi-colon — shifted key O
- (:) colon — shifted key Z
- (.) full stop — shifted key M
- (?) question mark — shifted key C
- (!) exclamation mark — shifted key 1
- (') apostrophe — shifted key 7

Practise using the keyboard by trying any of the following exercises. Remem-

ber to start each line with the keyword PRINT followed by inverted commas. Finish each exercise with inverted commas and then press ENTER. If you forget either the keyword or the inverted commas and try to enter a line of text, the computer will give you a question mark which indicates an error.

1. TYPE: a , B ; C : d , e ! F ? g H , i ; J .
2. Type in the numbers 1 to 20 with a comma separating each number (note that number 0 is on key marked DELETE and looks like this: Ø. A capital letter O cannot be used in its place).
3. Type this sentence:

THE quick BROWN fox JUMPED over the LAZY dog.

4. Write a short letter to a friend.

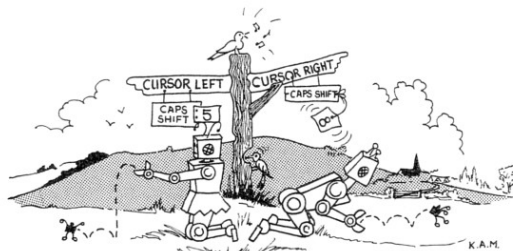
If you wish to type in upper case letters for several words the computer can be put into *capitals mode*. This is done by pressing CAPS SHIFT and CAPS LOCK (key 2) at the same time. The cursor changes to a flashing C reminding you that the computer is in *capitals mode*. Try this.

TYPE: PRINT "COMPUTER PROGRAMMING IS FUN"

Press CAPS SHIFT and CAPS LOCK again; this puts this computer back into *letter mode*. Press ENTER.

At the end of each exercise always clear the screen or remove the old program from the computer. At the moment all you need to do is to press ENTER. Unit 3 explains how to remove a program.

Unit 2: Correcting mistakes



In unit 1 you learned that a letter could be removed by pressing CAPS SHIFT and DELETE. Sometimes mistakes are not noticed immediately and you may have typed in a whole sentence before realising that a mistake has been made. In order to correct this mistake it is not necessary to delete the whole sentence.

Type in this sentence as it is written including all the mistakes

PRINT "Their are five speling nisteakes inn this sentence."

Do not press enter.

To correct the sentence the L cursor has to be moved. CAPS SHIFT and key 5 moves the cursor to the left and CAPS SHIFT and key 8 moves it to the right. The word 'sentance' should be spelt 'sentence'.

Using the correct keys move the cursor so that it lies to the *right of the wrong letter*: sentaLnce. (When you move the cursor hold CAPS SHIFT down firmly otherwise you will have 5s and 8s appearing in your sentence.)

Now press CAPS SHIFT and DELETE and the 'a' will disappear. Press key E and the word is now correct.

The word 'speling' has an 'l' missing.

Move the cursor into the position of the missing letter and then press key L.

Now correct the rest of the sentence which should read: "There are five spelling mistakes in this sentence."

Press ENTER.

The correct sentence now appears at the top of the screen.

Other examples to try.

- (a) The whether tooday is dul and cloudie; I fink it will reign.
Correct version: The weather today is dull and cloudy, I think it will rain.
- (b) The hair jumppped over the fense.
Correct version: The hare jumped over the fence.
- (c) The preist preyed four worlde piece.
Correct version: The priest prayed for world peace.

Unit 3: Making a program

TYPE: 10 PRINT "This is a program."

Press ENTER.

The whole sentence plus the number and the keyword PRINT appears at the top of the screen. The computer is now making a *program* by storing the information for later use.

TYPE: 20 PRINT "A program consists of a list of instructions or statements."

Press ENTER.

TYPE: 30 PRINT "Program lines are numbered in tens."

Press ENTER.

Notice that at the end of every line you had to press ENTER. This transferred the line to the top of the screen and made it part of the program. You cannot type in another statement until this has been done.

Press the keyword RUN on key R followed by ENTER.

The computer now executes or performs the program and gives a similar message to the one it gave before. Each statement will be printed in the same order as that in the original program. In other words the numbers at the beginning of each line tell the computer in which order to follow the instructions.

Press ENTER.

The program has returned to the screen. It can be executed as many times as you wish but the computer will only do this if you give it the command or keyword RUN followed by ENTER. This should be done every time a program is typed in. It is the only way of telling if the program is correct. The computer will not execute a program that has mistakes or *bugs* in it. Also it is the only way of obtaining the result of your program.

Press the keyword NEW on key A followed by ENTER.

The program has now gone. It has been removed from the computer's memory.

Try the following exercises. Remember to RUN each program and then remove it from the computer before you start the next.

- (1) Type in a program that prints out your name and address as it would appear on an envelope.

- (2) Type in a program that prints your name, age and date of birth.
- (3) Type in a program that prints out information about your family.

The program list can be returned to the screen by pressing ENTER. Later on as your programs get longer they will not necessarily fit on to the screen; you should then press LIST (key K) followed by ENTER. This will give you as much program as will fit on to the screen and the message 'scroll'. You then press any key other than N for 'no' or BREAK/SPACE and the next part of the program will appear. This can be repeated until you reach the end of the program. If you only want a certain section of the program then press LIST and the line number you require, followed by ENTER.

Unit 4: More about numbering lines

Program lines are usually numbered in intervals of ten; for example, 10, 20, 30, 40 etc. The reason for this is that when a program is first made lines are often missed out by mistake. Having intervals of ten between lines enables a further nine lines to be inserted later if necessary. For example

```

10 First line of program
11
12
13
14
15 } Space for nine more lines if needed
16
17
18
19
20 Second line of program

```

Type in the following instructions which were given to a girl who did not know how to change her socks.

```

10 PRINT "How to change socks"
20 PRINT "Take off shoes"
30 PRINT "Put on clean socks"
40 PRINT "Put on shoes"

```

Like a computer the girl is very stupid and will only do exactly what she is told to do, so one day she finds that she cannot get her shoes on as nobody told her to take off her dirty socks! Therefore the program must be improved.

TYPE:

```
25 PRINT "Take off dirty socks"
```

Press ENTER.

You will notice that line 25 is automatically inserted into the correct place in your program. If the program had been numbered 1, 2, 3 and 4 this would not have been possible as the computer only accepts whole numbers at the beginning of lines; you cannot use 1.5 or 1.75.

TYPE:

```
10 PRINT "How to wash hands"
20 PRINT "Put plug in basin"
30 PRINT "Turn off taps"
40 PRINT "Rub soap between hands"
50 PRINT "Take plug out of basin"
60 PRINT "Dry hands"
```

At least four instructions have been deliberately left out.

Type in the missing instructions using the correct line numbers.

Unit 5: Editing a program

Editing enables the programmer to alter a mistake in a line or improve a line that is already part of a program.

Type in this program including all the deliberate mistakes.

```
10 PRINT "Helo fred."
20 PRINT "Doyou like computing"
"
30 PRINT "No,not MUCH."
40 PRINT "never mind,you will
to like it."
```

There are one or two mistakes in each line of this program.

Line 10 – 'helo' is spelt wrongly and fred does not begin with a capital letter.

Line 20 – there is no space between 'Do' and 'you' and the question mark has been missed out.

Line 30 – the word 'much' is written in capital letters.

Line 40 – the sentence does not begin with a capital letter and a word is missed out.

These grammar mistakes will not be noticed by the computer; it has never been given English lessons so it will accept any punctuation or spelling. You could write a sentence of complete rubbish and the computer would print it if commanded to do so.

RUN the program; you will see that the mistakes are just ignored. The program is printed exactly as it was typed in. Return the program to the screen by pressing ENTER.

However, as we progress to more complicated programs it will be necessary to correct mistakes especially if they are in BASIC which the computer does understand, so we will learn how to do this.

To correct or edit the program we need to bring each line that has a mistake in it back to the bottom of the screen. Look at line 40, you will see a > sign known as the program cursor lying between the number and the keyword PRINT.

Press the key marked EDIT (key 1) and CAPS SHIFT at the same time.

Line 40 should have returned to the bottom of the screen where it can be

corrected in the normal way. (There is still a copy of this line left in the program.)

Correct line 40 (see unit 2). When it is correct press ENTER.

The correct line now takes the place of the incorrect one. You do not finish up with two lines 40.

Next we have to correct line 30. In order to do this we must move the program cursor (>) up to this line.

Look at key 7, it has a small white arrow pointing upwards above it. Press key 7 and CAPS SHIFT at the same time. This moves the program cursor up one place; it should now be between 30 and PRINT.

Press EDIT and CAPS SHIFT at the same time — line 30 is now at the bottom of the screen. Correct and then return to the program.

Move the program cursor up to line 20 (key 7 + CAPS SHIFT).

Bring line 20 to the bottom of the screen (key EDIT + CAPS SHIFT).

Return to program.

Edit line 10.

If you move the program cursor too quickly by pressing key 7 more than once you may pass the line you wish to correct.

The program cursor can be moved downwards by pressing key 6 and CAPS SHIFT.

Sometimes it is necessary to remove or delete a whole line from a program. This is easily done by typing in the number of the line followed by ENTER. Try this on the previous program.

TYPE: 10

Press ENTER.

Activity:

Type in a program that prints out four sentences about your school. Each sentence must contain one or two deliberate mistakes. RUN the program. Correct or edit the program. RUN the corrected program.

(Remember to remove the old program from the computer — keyword NEW followed by ENTER.)

Unit 6: Extended mode. Using the cassette recorder

Extended mode

Some of the commands used in this lesson require the computer to be in *extended mode*. This is because we need to use some of the green and red words written above and below the keys. For example, we will need VERIFY, written in red below key R, and LLIST, written in green above key V. The computer is put into extended mode by pressing CAPS SHIFT and SYMBOL SHIFT at the same time which changes the flashing cursor to an E. Once in extended mode any word in green above a key can be obtained; for example, pressing key A will produce READ. To obtain the red words below the keys SYMBOL SHIFT has to be pressed again at the same time as the required key; for example, SYMBOL SHIFT and key X will produce INK.

Using the cassette recorder

Programs recorded on cassette tapes are known as *software*. Only software prepared for the Spectrum can be used on this machine. If you buy a tape, such as a game, make certain that your computer has enough memory for that program. Your Spectrum will either have a 16K or a 48K memory; you cannot run a 48K program on a 16K machine.

Loading a program

1. Using the lead provided connect the EAR socket of the computer to the EAR socket of the cassette recorder. Use the same coloured leads. *Do not connect the MIC leads.*
2. Turn the volume control on the cassette recorder up to almost full. Do not touch this control while the program is loading. If you have a tone control, move this to almost full.
3. Insert the tape into the cassette recorder with the side that you require uppermost.
4. Type into the computer the keyword LOAD (key J) followed by the name of the program in quotation marks. For example, if you are loading the tape

Horizon supplied with your Spectrum type: LOAD "sidea". (Notice the way "sidea" is written, you must type in the name in exactly the same manner.

"SIDEA" is *not* the same as "sidea".)

5. Press ENTER.
6. Press the PLAY button on the cassette recorder.
7. *Now wait.* If you are loading your own program then the name of the program will appear on the screen. With bought tapes other messages and pictures may appear. Moving horizontal lines are visible around the border of the screen; this is quite normal. If the program is not loading you will be given an error message such as R Tape loading error. In this case stop and rewind the tape. Loading errors are usually caused by:
 - (a) The volume control being too high or too low. If it is too low the computer cannot pick up the signals and if it is too high the signals will be distorted. Vary the level slightly, as appropriate.
 - (b) The wrong leads being connected.
 - (c) The wrong name being typed in.

Check all of these things before you try again.

8. When the message OK appears your program has loaded so stop the tape. With bought tapes you are sometimes told when to stop the tape.

Saving a program

Any program that you have typed into your computer can be saved or recorded on to a tape.

1. Using the lead provided connect the MIC socket on the computer to the MIC socket on the cassette recorder. Use the same coloured leads. *Do not connect the ear sockets.*
2. Turn the volume control on the cassette recorder up to almost full if you do not have automatic-level recording.
3. Insert an empty tape or any tape that has space on it into the cassette recorder.
4. Type into the computer the keyword SAVE (key S) followed by the name of the program in quotation marks. You can call your program anything you like provided that the name consists only of numbers and letters and is not longer than ten characters. For example:
SAVE "Robots".
5. Press ENTER.
6. A message appears on the screen: Start tape, then press any key. Press the PLAY and RECORD buttons on the cassette recorder and then any key on the computer.
7. *Now wait.* Moving horizontal lines appear around the border of the screen.
8. When the end of the program is reached you will be given the message OK; stop the tape. Unfortunately OK does not necessarily mean that your program has been recorded; you need to check this.

Checking that your program has been recorded

1. Rewind the tape back to the place where you started.
2. Connect the EAR socket on the cassette recorder to the EAR socket on the computer. Use the same coloured leads. Disconnect the MIC leads.
3. Type in the keyword VERIFY (below key R — extended mode and then symbol shift + R.) followed by the name of the program in quotation marks. For example VERIFY "Robots".
4. Press ENTER.
5. Press PLAY button on cassette recorder.
6. *Wait.* The name of the program should appear together with the moving horizontal lines if the program has been recorded successfully. When the computer has finished checking it will give the message OK. This means that your program is on the tape. If the name does not appear or the message R Tape loading error does appear then something has gone wrong. Try saving the program again.

Reasons why the program may not have recorded are:

- (a) Volume control is too high or too low.
- (b) Wrong leads are being connected.
- (c) You may have tried to record on the coloured leader at the beginning of the tape.

If your cassette recorder has a counter make a note of the number before you save a program. This can be very useful if your program is in the middle of a tape as it takes the computer a long time to search through a tape looking for a program. However, if you use the fast forward button and go to approximately the right place on the tape, finding the program will be much quicker.

Unit 7: Testing your knowledge— questions on the Introduction and Units 1–5

Try to answer from memory as many questions as you can. Look up those you do not know in the appropriate unit — the questions are divided into sections. Check your answers at the back of the book.

Introduction: The microcomputer system

1. Name an input device.
2. Name two output devices.
3. Name a device that can be used for both input and output.
4. What happens in the Central Processing Unit?
5. (a) What is a character?
(b) Give examples of characters.
6. (a) What is the language used by this computer?
(b) What is the full meaning of this word?

Unit 1: Using the keyboard

7. What mode is the computer in when a flashing **K** is on the screen?
8. What mode is the computer in when the computer is first switched on?
9. What is the flashing **K** called?
10. In which mode does the computer have to be in before you can use a command such as PRINT?
11. Do commands appear on the screen as one word or do you type in each letter separately?
12. On which letter key would you find PRINT?
13. How do you remove or rub out a letter?
14. What mode is the computer in when a flashing **L** is on the screen?
15. When does the flashing **K** change to a flashing **L**?
16. (a) Quote marks are part of the computer's language; when are they used?
(b) How are the quote marks obtained?

17. Quote marks are written in red on key P; give another example of a symbol written in red on a key.
18. What is meant by a shifted key?
19. What is the difference between lower and upper case letters?
20. How is an upper case letter obtained?
21. How is a space between words achieved?
22. Can the letter O be used in the place of the number nought?
23. (a) What is meant by *capitals mode*?
(b) How do you put the computer into this mode?
(c) How do you get the computer out of this mode?

Unit 2: Correcting mistakes

24. The flashing **L** can be moved to the right or to the left by pressing key 5 or key 8. Which other key has to be pressed at the same time?
25. Look at the word below, which has the flashing **L** placed in the centre of it. If CAPS SHIFT and DELETE were pressed which letter would be removed, the 'u' or the 't'?

compuLter

Unit 3: Making a program

26. What must be typed in at the beginning of a line if you wish the statement or instruction to become part of a program?
27. If you have finished typing in a line of a program how do you transfer it to the top of the screen?
28. What does a program consist of?
29. How do you make the computer perform or execute your program?
30. When a program is performed do the line numbers and commands appear on the screen?
31. After a program has been performed how do you return the program listing to the screen?
32. Can a program be performed more than once?
33. How do you remove the program from the computer's memory?

Unit 4: More about numbering lines

34. Why are program lines usually numbered in tens?
35. What is wrong with the program that follows:

```

10 PRINT "Hello."
20 PRINT "How are you?"
40 PRINT "I'm sorry to hear that."
30 PRINT "Not too well."

```

Unit 5: Editing a program

36. What does editing a program mean?
37. (a) What is the > sign which appears by a line in a program called?
(b) How is this sign moved up or down?
38. How do you return a line to the bottom of the screen when it is already part of a program?
39. Having corrected a line in a program and returned it to the program do you have two versions of the line – a correct one and an incorrect one?

General

40. Rewrite the program below correcting all the mistakes.

```

1 PRINT "Do you like robots?"
20 PRINT Yes.
15 PRINT "I don't."
30 LET "I think they are great big metal idiots."

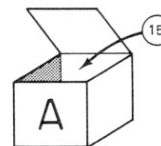
```

If there were many questions you could not answer you probably need to read through these units again. If you answered correctly from memory 30 or more questions then you now know how to handle the Spectrum and are ready to proceed to more advanced programming.

Unit 8: Numeric or simple variables



A numeric or simple variable is a storage space in the computer's memory which is given a name. The storage space stores only numbers, hence its name. The easiest way to imagine this is to think of a box that has a name and contains a number. The example below shows a box called 'A' storing the number 15.



BOX OR VARIABLE CALLED 'A'

The computer uses the keyword LET (key L) and the equals sign (=), which is shifted key L, to set up the variable. For example

```
LET A = 15
```


The computer now knows that every time 'A' is used in the program it is really referring to the contents of the box. The next program illustrates this point.

```
TYPE: 10 LET A = 15
      20 PRINT A
RUN    DO NOT REMOVE
```

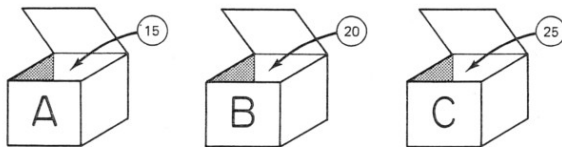
The program shows that in the PRINT statement in line 20 there are no quote marks because we are not asking the computer to print letter A but the contents of 'A' and therefore 15 appears on the screen. If we wanted the letter printed then quotes would have to be used in the normal way. This can be shown by adding one line to the program.

Add to the program:

```
15 PRINT "Box or variable A contains the number"
```

RUN the program and then remove

There may be several variables in a program provided that they each have different names.



Here there are three boxes or numeric variables. Box 'A' remains the same and is unaffected by adding two more boxes. Box 'B' contains 20 and box 'C' 25. A program using these variables would need three LET statements:

```
TYPE:
10 LET A=15
20 LET B=20
30 LET C=25
40 PRINT A
50 PRINT B
60 PRINT C
```

```
RUN    DO NOT REMOVE
```

The result is simply: 15
20
25

The name of the variable does not have to be a capital (upper case) letter, it could equally well be a small (lower case) letter. Try this:

Edit lines 40, 50 and 60

```
40 PRINT a
50 PRINT b
60 PRINT c
```

```
RUN    DO NOT REMOVE
```

The result is the same. The computer does not distinguish between capital and small letters when using variables. Other names can also be used for variables but there are two rules to remember:

- (1) The name must begin with a letter.
- (2) The name can consist only of letters and numbers.

For example the following names are allowed:

apples	} all begin with letters	
CAR		
ZX81		} all consist of letters and numbers only
Spectrum		
total 2		
Two cats	} spaces between words are also allowed	

The following names are *not* allowed

2 apples (begins with a number)
Spectrum ! (exclamation mark is not a letter or a number)
CAR + TON (plus sign is not a letter or a number)

The last important thing to know about a numeric variable is that the box or storage space can remember only one number at a time. It will always store the last number that it is given.

Add to the previous program:

```
70 LET A=150
80 LET B=200
90 PRINT A
100 PRINT B
110 PRINT C
```

```
RUN
```

The result is: 15 (first value of box A)
20 (first value of box B)
25 (first value of box C)
150 (second value of box A)
200 (second value of box B)
25 (first value of C — this box has remained unchanged.)

It is not possible to appreciate the real value of numeric variables at this stage and you may be thinking: why not print out the numbers straight away (for example, PRINT 15, PRINT 20 etc.)? However, they are a very important part of programming and gradually you will see that they have many uses. You will meet them next, in greater detail, in unit 11.

Activities

Write out the results of the following programs.

1.

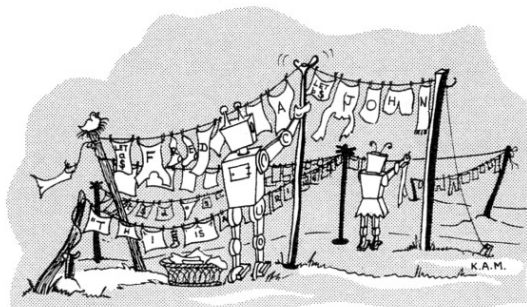
```
10 LET x=32
20 LET y=44
30 PRINT x
40 PRINT X
50 PRINT "y"
```
2.

```
10 LET W=68
20 LET dogs=2
30 LET W2=53
40 LET w=72
50 PRINT W
60 PRINT "dogs"
70 PRINT W2
```

Which of the following names for numeric variables are allowed:

- (a) ABC (b) 2ABC (c) abc (d) Two ABC (e) Abc !
 (f) Abc (g) A+B+C (h) A(B)C (i) A2B2C (j) a b c

Unit 9: String variables



Revision: Characters are the letters of the alphabet, the numbers 0 to 9, punctuation marks and special symbols such as +, -, *.

A group of characters is a *string*. Some examples of strings are:

- dog (a word)
- Queen Victoria (a name)
- ZX81 (a mixture of letters and numbers)
- 12, W(2) (a mixture of characters)

A string can be any mixture of characters; even a space is counted as a character.

A *string variable* is a storage place in the computer that remembers groups of characters or strings. It is very similar to a numeric variable except that the storage place name must be a *single letter of the alphabet* with the sign for a *dollar (\$)* which is *shifted key 4*. The characters to be stored in the string variable must be contained in inverted commas. For example

```
LET a$ = "Fred"
LET b$ = "dog"
LET z$ = "1,2,3,4,5,"
```

TYPE:

```
10 LET a$="This is a string"
20 LET s$="A string is a group
of characters"
30 PRINT a$
40 PRINT s$
```

Notice that as for numeric variables no inverted commas are needed with the PRINT statement as it is not a \$ and s\$ that are to be printed but the contents of a\$ and s\$.

RUN the program. Notice the result.

String variables can be combined with other words to make up parts of sentences. The variables are not contained within inverted commas, as previously mentioned, but must be separated from the other words in the sentence by a semi-colon (;). The semi-colon is part of the BASIC language and cannot be omitted. For example

```
10 LET a$ = "Fred"
20 PRINT "Hello"; a$; "How are you?"
```

The result of this program would be:

HelloFredHow are you?

As you can see the words are all squashed together. This is because the semi-colon also brings the next character to be printed right next to the previous one. Therefore, it is necessary to include spaces as part of the sentence. (Note that spaces will be frequently used in future programs and will be represented in the text by a small dash (-). In listings obtained via the computer printer, *only the space itself will appear.*) For example,

```
10 LET a$ = "Fred"
20 PRINT "Hello-"; a$; "-How are you?"
```

The result will now be:

Hello Fred How are you?

Notice that the spaces came inside the inverted commas as they are counted as characters and are printed, although invisible, in the same way as any other character.

TYPE:

```
10 LET n$="your name"
20 LET a$="your age in years"
30 LET t$="your home town or v
illage"
40 LET c$="your county"
50 PRINT "My name is ";n$
```

Type in your own
personal details
here: not the
words "your
name" etc.

```
60 PRINT "My age is ";a$
70 PRINT "I live in ";t$;" whi
ch is in the county of ";c$
```

RUN the program and notice the result.

If your program will not RUN examine it closely for any omissions; you may have missed out a variable.

There are only 26 storage spaces in the computer for strings, one for each letter of the alphabet.

Which of the following names can be used as storage places for string variables?

- (a) a\$, (b) M\$, (c) T\$, (d) B\$, (e) C\$, (f) \$P\$,
(g) 2\$, (h) K\$, (i) w\$, (j) aa\$, (k) ax\$, (l) a

Think of all the variables that you have met so far and decide which of the following are correct BASIC statements.

- (a) LET A = 87 (e) LET L17 = 38
(b) LET R\$ = "cat" (f) LET eggs = 50
(c) LET m\$ = 1234 (g) LET milk = "twenty-two"
(d) LET K8 = "Spectrum" (h) LET 8A = 6

Programming activities (remember to type in and RUN each program)

- a\$ = "John"
s\$ = "school"
Write a program that prints out these two variables in a sentence.
- m\$ = "Mary"
a\$ = "Anne"
b\$ = "12 years"
c\$ = "13 years"
Write a program, using the variables, that prints out the ages of Mary and Anne.
- Use string variables for the names of two friends and the places where they live. Using these variables in the correct places, print two sentences that give this information.
- Print out a letter that invites a friend to a party. Use the following variables:
n\$ = name of friend.
d\$ = date of party
w\$ = name of day of the party
t\$ = time of the party.

String variables can be added together; this is known as *concatenation*

TYPE:

```
10 LET a$="Miss "
20 LET b$="Miller"
30 PRINT a$+b$
40 LET c$="Con"
50 LET d$="vent"
60 PRINT a$+b$;" teaches at th
e ";c$+d$
```

RUN the program and notice the result.

Activities

5. Type in a program, using string variables, that adds your Christian name to your surname and then prints out the result.
6. Using three string variables divide up the name of a town. Type in a program that adds up the name and prints it out.
7. THE GREEDY BOY ATE MANY CREAM BUNS AND JAM TARTS.
Use a variable for each word in the sentence. For example

a\$ = "THE" b\$ = "GREEDY" etc.

Write a program that adds together some or all of these variables to make different sentences.

Unit 10: Extracting parts of strings

TYPE:

```
10 PRINT "abcdefgh"
20 PRINT "abcdefgh"(1 TO 3)
```

↑
shifted F — do not type in
the letters T and O separately

```
30 PRINT "abcdefgh"(5 TO 8)
40 PRINT "abcdefgh"(2 TO 6)
50 PRINT "abcdefgh"(4)
```

RUN the program.

The result is: abcdefgh
abc
efgh
bcdef
d

You can see that any part of the string can be extracted (taken out) by using numbers in brackets by the side of the string. For example (1 to 3) means print all the characters from the first which is 'a' to the third which is 'c', so the result is 'abc'. If there is only one number in the bracket, as in line 50, this means just print the character at that number, in this case the fourth which is 'd'.

TYPE: 10 PRINT "aybxcwp"

Add lines that will produce these results using the method described above:

```
ayb
cwp
x
bxcw
```

RUN your program and check your results.

Instead of continually typing in the string of characters it is much quicker to use a string variable.

TYPE:

```
10 LET a$="Father Christmas"
20 PRINT a$
30 PRINT a$(1 TO 6)
40 PRINT a$(8 TO 16)
```

RUN DO NOT REMOVE

The result is: Father Christmas
 Father
 Christmas

You may have wondered why 'Christmas' was not (7 to 15) as 'C' is the seventh letter and 's' the fifteenth; but the space between the two words is also a character so it is also counted.

Add to the program so that the following extracts are printed:

(a) her (d) Fat
 (b) rist (e) C
 (c) mas

Use this as a guide:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
 F a t h e r C h r i s t m a s

RUN the program and check your results.

Extracting and concatenation

This is simply taking away some parts of a word and adding it to parts of another word. See lines 40 and 50 of the program below

TYPE:

```
10 LET b$="baked"
20 LET p$="potato"
30 PRINT b$+p$
40 PRINT b$(1 TO 2)+p$(3)
50 PRINT p$(3 TO 4)+b$(3 TO 4)
```

RUN DO NOT REMOVE

The result is: bakedpotato
 bat
 take

Each word is treated separately in this case as they are stored in two different string variables, therefore the first letters in both words have the number one.

Activities

1. Alter line 30 so that 'bakedpotato' is printed with a space between the two words.

2. Add lines so that the following are printed:

toe, pod, ate, pad, depot

RUN the program. Check that your results are correct.
 Remove the program.

3. Write a program that stores the words 'ladle' and 'spoon' and then prints out 'paddle' and 'lap'.
 Add lines to your program to print out five more words taken from 'ladle' and 'spoon'.

Unit 11: Using the computer as a calculator

For arithmetic you use four main operators: add, subtract, multiply and divide. BASIC language has the same operators although two are printed differently.

Everyday symbol	Meaning	Basic symbol	Key
+	add	+	shifted K
-	subtract	-	shifted J
x	multiply	*	shifted B
÷	divide	/	shifted V

Activities

- Write the following arithmetic expressions using BASIC symbols for the operators.
 - $3 + 7$
 - $5 \times 2 + 8$
 - $5 \times 6 \times 7$
 - 3×7
 - $24 - 4 \times 3$
 - $81 - 27 \div 2$
 - $8 \div 4$
 - $30 \div 3 + 2$
 - $23 \div 2 \times 9$
- The following arithmetic expressions have been written in BASIC. Calculate the answers.
 - $2 + 10 + 5$
 - $50/2$
 - 2×100
 - $12 - 7$
 - $2 \times 5 \times 10$
 - $6 \times 5/10$
- Which is the correct answer for each of the following?
 - $2 + 5 \times 4 = 32$ or 26 ?
 - $3 \times 6 - 5 = 13$ or 3 ?
 - $4 + 8/2 = 6$ or 8 ?
 - $6/2 - 2 = 1$ or 2 ?

Type in this program to check your answers.

```
10 PRINT 2 + 6 * 4
20 PRINT 3 * 6 - 5
30 PRINT 4 + 8/2
40 PRINT 6/2 - 2
```

Where two operators are of equal importance (multiplication and division are equal, and addition and subtraction are equal), the computer works out the operations in the order from left to right. For example

$$10 * 10/5 = 100/5 = 20$$

$$10/10 * 5 = 1 * 5 = 5$$

$$10 - 5 + 6 = 5 + 6 = 11$$

The computer will also work out parts of expressions contained in brackets first; before multiplication and division. For example

$$3 + 2 * 4 = 11 \text{ but } (3 + 2) * 4 = 20$$

Activities

- Calculate the following
 - $6 + 4 * 2$
 - $(10 - 2) * 3$
 - $(6 + 4) * 2$
 - $6 + 18/2$
 - $10 - 2 * 3$
 - $(6 + 18)/2$
 Type in a program that will check your answers.
- If A has the value of 2, B has the value of 5 and C the value of 10 calculate the following:
 - $A + B + C$
 - C/A
 - $B - A$
 - $A + C - B$
 - $B * C$
 - $A * C$

TYPE:

```
10 LET A=2
20 LET B=5
30 LET C=10
40 PRINT A+B+C
50 PRINT B-A
60 PRINT B*C
70 PRINT C/A
80 PRINT A+C-B
90 PRINT A*C
```

RUN the program and check your answers to the previous exercise.

As you can see from the program that you have just typed in, arithmetic is usually done by the use of numeric variables and the keyword LET. Lines 10, 20 and 30 set up the variables, therefore the storage places in the computer looked like this:

A	2	B	5	C	10
---	---	---	---	---	----

Now let us consider this program:

```
10 LET B = 15
20 LET C = 10
30 LET A = B - C
```

After lines 10 and 20 of the program the storage spaces in the computer look like this:

A	B	C
	15	10

After line 30 has been executed they look like this:

A	B	C
5	15	10

If we add to the program:

40 LET A = B * C

the storage places will now look like this:

A	B	C
150	15	10

Notice that the content of A has changed. Storage space 'A' retains only the latest piece of information. If we add another line to the program the contents of the storage spaces change accordingly.

50 LET B = A + C

A	B	C
150	160	10

6. Copy the programs listed below and the storage boxes.

Fill in the values at the storage spaces A, B and C after each line of these programs has been executed.

	PROGRAM	STORAGE SPACES		
		A	B	C
(a)	10 LET A = 12	<input type="text"/>	<input type="text"/>	<input type="text"/>
	20 LET B = 5	<input type="text"/>	<input type="text"/>	<input type="text"/>
	30 LET C = A * B	<input type="text"/>	<input type="text"/>	<input type="text"/>
	40 LET A = C - B	<input type="text"/>	<input type="text"/>	<input type="text"/>
(b)	10 LET A = 20	<input type="text"/>	<input type="text"/>	<input type="text"/>
	20 LET B = A * 3	<input type="text"/>	<input type="text"/>	<input type="text"/>
	30 LET C = A/4	<input type="text"/>	<input type="text"/>	<input type="text"/>
	40 LET B = A * C	<input type="text"/>	<input type="text"/>	<input type="text"/>

Programming activities

7. Type in a program that has two numeric variables and then prints out:

- their sum (addition)
- their difference (subtraction)
- their product (multiplication)
- their quotient (division).

8. The length of a rectangle is 35 centimetres and the breadth is 16 centimetres.

Type in a program that uses variables for the length and breadth and then calculates:

- the area of the rectangle (breadth * length)
- the perimeter of the rectangle (the sum of all four sides).

Like string variables, numeric variables can be combined with words. Semi-colons are used to separate the words from the variables.

TYPE:

```
10 LET c=100
20 LET p=40
30 LET t=80
40 PRINT "My ZX Spectrum cost
";c;" pounds"
50 PRINT "The printer cost ";p
;" pounds";
60 PRINT " and the television
cost ";t;" pounds"
70 PRINT "The total cost of my
equipment was ";c+p+t;" pounds"
```

RUN the program and notice the results.

Add a line to the program that prints out the total cost of all the equipment in a room that has six microcomputer systems. (Note: the computer should do the calculating, not you!)

Programming activities

9. The program below states that Mr Blake buys 9 bags of cement at 93 pence per bag and then calculates the cost. Rewrite the program correcting all the mistakes.

```
10 LET x = "Mr Blake"
20 LET b$ = 93
30 LET C = "9"
40 PRINT A bag of cement costs." ; b$ ; "pence.
50 PRINT x$ "ordered." c "bags."
60 LET "The cost of"; c ; "bags is"; b$/c
```

10. The program below gives the names of three children and their examination results. It prints out this information and then calculates the average mark. Rewrite the program correcting all the mistakes.

```

10 LET a$ = Susan Smith
20 PRINT b$ = "John Brown"
30 LET c = "Mary Turner"
40 LET a = "43"
50 RUN b$ = 68
60 LET c = "seventy-three"
70 PRINT "a$"; "-gained-"; a$; "-marks."
80 PRINT b$; "-gained-"; b; "-marks."
90 PRINT c "-gained-"; c$; "-marks."
100 LET "The average mark was-"; a$ + b$ + c/3

```

11. Complete the program below which calculates the length of a car journey. The speed of the car is 65 miles per hour and it travels for 6 hours.

```

10 ____ s = 65
20 LET ____ = 6
30 PRINT "Speed of car is" ; ____ ; "miles per hour."
40 PRINT "Time spent travelling is" ; ____ ; "hours."
50 ____ "Length of journey is" ; ____ ; "miles."

```

12. Type in a program that uses variables for your:

- (a) age
- (b) weight
- (c) height
- (d) size of shoes

and then prints out this information in a descriptive paragraph.

13. Type in a program that uses variables for all the different ages of people in your family and then calculates and prints out:

- (a) the sum of the ages of your family
- (b) the average age of your family

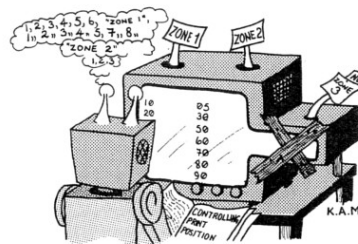
14. A lorry carries 1154 kilograms of sand in one journey.

It makes 19 journeys every four weeks. Write a program that calculates and prints out:

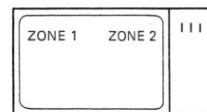
- (a) the amount of sand carried every four weeks
- (b) the amount of sand carried in 48 weeks.

15. A loaf of bread costs 44 pence, a pint of milk 22 pence and a pound of bacon 104 pence. Write a program that calculates and prints out the price of 3 loaves, 5 pints of milk and half a pound of bacon and then gives the total cost of all the purchases.

Unit 12: Controlling the print position



There are two print zones on the Spectrum screen. Zone 1 is on the left and Zone 2 on the right. Zone 2 begins in the 16th space across the screen.



To move a PRINT statement into Zone 2 a comma is used.

TYPE: PRINT "Zone 1", "Zone 2"

Press ENTER and notice the result.

Now TYPE: PRINT "Zone 1", "Zone 2"

Press ENTER and notice the result.

TYPE: PRINT 1; 2; 3; 4; 5; 6

Press ENTER

Result on the screen is: 123456

TYPE: PRINT 1, 2, 3, 4, 5, 6

Press ENTER


```
Result on the screen is: 1      2
                        3      4
                        5      6
```

Notice that the comma after 1 moves the 2 into Zone 2 but that the comma after 2 moves the 3 back into Zone 1. In other words a comma does not just move PRINT statements into Zone 2 but into the next available zone.

TYPE: PRINT 1,, 2,, 3,, 4
Press ENTER

```
Result on the screen is: 1
                        2
                        3
                        4
```

Can you see why this has happened?

Activities

1. Say what will appear on the screen when you type the following:

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

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

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

2. Write a PRINT statement for the following result on the screen:

```
1      2
45     3
```

3. Write a PRINT statement for the following result on the screen:

```
12     34
5       6
```

Check your answers to the questions above by typing them into the computer.

Sometimes we will want to leave spaces between lines. This is achieved by using just the PRINT statement on its own.

TYPE:

```
10 PRINT "I am on the top line"
20 PRINT
30 PRINT "I am on the third line"
40 PRINT
50 PRINT "I am on the fifth line"
```

RUN

The result is: I am on the top line

```

I am on the third line
I am on the fifth line
      blank lines
      left in between
```

On other occasions we may need to move the print position across the screen but not necessarily into the next zone, so we cannot use a comma. To move the print position we use the function TAB. This is an abbreviation for *tabulation* which means to arrange in table-form. TAB is found above key P and is obtained by putting the computer into extended mode (press CAPS SHIFT and SYMBOL SHIFT at the same time). TAB is used with a number; the number indicates how many spaces we want to move across the screen. Think of the spaces across the screen as columns beginning at 0 on the left-hand side and finishing at 31 on the right-hand side. This makes 32 columns or printing positions. For example:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      ↑           ↑           ↑           ↑
      5th column 11th column 19th column 27th column
      across    across      across      across
```

Notice that as the columns begin at 0, TAB 4 moves the printing position to column 5 and TAB 10 moves it to column 11.

TYPE:

```
10 PRINT "01234567890123456789"
20 PRINT "a";TAB 5;"b";TAB 10;
" c";TAB 15;"d";TAB 20;"e";TAB 25;
"f";TAB 30;"g"
```

RUN

The result is:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
a      b      c      d      e      f      g
```

So the important thing to remember is that when you type TAB 'n' (n represents any number), the printing will start one column further across.

The next program shows how a table of information can be produced.

TYPE:

```
10 PRINT "NAME";TAB 10;"AGE";TAB 10;"HEIGHT"
20 PRINT "Sarah";TAB 10;"13";TAB 10;"153cms"
```

```

40 PRINT "Jane";TAB 10;"14";TA
B 20;"155cms"
50 PRINT "Susan";TAB 10;"12";T
AB 20;"147cms"
RUN

```

The result is a list of names, ages and heights in three columns under appropriate headings. Notice in this program and the previous one the use of the semi-colons (;) to separate the TAB instructions from the words in quotation marks.

Activity

4. Write a program that prints out the information in table-form as shown below.

Day	Temperature (degrees C)	Rainfall (inches)
Monday	10	0.2
Tuesday	11	0.4
Wednesday	13	0.6
Thursday	11	1.0
Friday	6	0

Unit 13: Beginnings and ends of programs

Each program should begin with a *remark* statement which is abbreviated to REM (key E). The REM statement simply states what the program is about and gives it a name which can be referred to later. The computer ignores REM statements but it is good programming practice to include them. From now on each of our programs will start with a REM statement. For example:

```

10 REM Area of triangles
    or
10 REM Names of countries

```

REM statements can also be used anywhere in a program to show the beginning of a new section. For example

```

10 REM Areas of triangles
20 LET height=10
30 LET base=8
40 LET area=base/2*height
50 PRINT area
60 REM Areas of rectangles
70 LET width=15
80 LET length=25
90 LET area=width*length
100 PRINT area

```

It is also good programming practice to let the computer know when a program has finished. We do this by adding a STOP statement (shifted A). For example

```

110 STOP

```

Unit 14: INPUT statement

Another way to give the computer information is with the keyword INPUT. This lets you put information in while the program is running and enables you to use different information each time that the program is run. The keyword INPUT is found on key I and like PRINT and LET can be used only when the flashing cursor is a K (keyword mode).

TYPE:

```
10 REM Illustrating input
20 PRINT "Type me a number"
30 INPUT x
40 PRINT "You just typed ";x
50 STOP
```

RUN the program.

On the screen you will see: 'Type me a number'.

At the bottom of the screen you will see the L cursor — this is the computer asking you for information; the computer just waits until you type in a number.

Type in: 63 Press ENTER

On the screen now: Type me a number
 You just typed 63

RUN the program several times putting in your own numbers.

INPUT can be used with string variables as well as numeric variables.

TYPE:

```
10 REM Input and string variables
20 PRINT "What is your name?"
30 INPUT n$
40 PRINT "My name is ";n$
50 PRINT "Where do you live?"
60 INPUT a$
70 PRINT "I live in ";a$
80 STOP
```

RUN the program. Notice that the cursor L is enclosed in inverted commas when information is being asked for, so reminding you that a string is required.

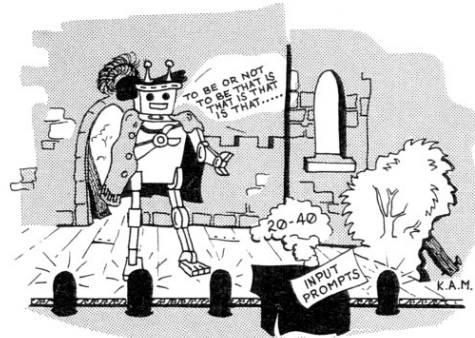
Using INPUT means that you can run the program as many times as you wish and keep changing the content of the variables. In unit 11 you wrote a program to find the area and perimeter of a rectangle. In this program the LET statement was used so it was possible to run the program only once with one set of measurements. However, by using INPUT the program can be run any number of times using different measurements each time.

TYPE:

```
10 REM Area and perimeter of
rectangles
20 PRINT "Give the length of a
rectangle"
30 INPUT l
40 PRINT "Give the breadth of
a rectangle"
50 INPUT b
60 PRINT "Area of rectangle is
";l*b;" sq.cm"
70 PRINT "Perimeter of rectang
le is ";(l+b)*2;" cm"
80 STOP
```

RUN this program several times putting in different measurements.

Lines 20 and 40 in the previous program are known as *input prompts*; in other words they are telling you what information the computer wants.



An input prompt can be written as a separate PRINT statement as in the previous program or as part of the INPUT statement as illustrated in the two examples below.

TYPE:

```
10 REM Input prompts
20 INPUT "Type in a number ";n
1
30 INPUT "Type in a second num
ber ";n2
40 PRINT n1;" + ";n2;" = ";n1+
n2
50 STOP
```

RUN the program several times. Notice that the input prompt appears at the bottom of the screen with the cursor and not at the top of the screen as in the previous example.

TYPE:

```
10 REM Writing a poem
20 INPUT "What is your name? "
;n$
30 PRINT "A poem by ";n$
40 INPUT "Type in a word that
rhymes with ME ";a$
50 PRINT "Here is the poem"
60 PRINT "Computers used to fr
ighten me"
70 PRINT "But now I'm as happy
as a ";a$
80 STOP
```

RUN the program several times altering the variables n\$ and a\$.

You can imagine that in the not too distant future your TV set, telephone and home computer will be linked together to form one 'intelligent' system so that on seeing an attractive advertisement on TV you could dial a number and get the following response.

TYPE:

```
10 REM Information recording s
ervice
20 PRINT "Hello"
30 PRINT "This is an informati
on recording service"
40 PRINT
50 PRINT "Please enter informa
tion as requested"
60 PRINT
70 PRINT "Your name? ";
80 INPUT n$
90 PRINT n$
100 PRINT "Your telephone numbe
r? ";
110 INPUT t$
120 PRINT t$
130 PRINT "Number or name of yo
ur house? ";
```

```
140 INPUT h$
150 PRINT h$
160 PRINT "Name of your road? "
;
170 INPUT r$
180 PRINT r$
190 PRINT "Name of your town or
city? ";
200 INPUT c$
210 PRINT c$
220 PRINT "Name of your county?
";
230 INPUT g$
240 PRINT g$
250 PRINT
260 PRINT
270 PRINT
280 PRINT "Thank you for your e
nquiry "
290 PRINT "Details of our servi
ces and products will be sent to
you"
300 PRINT
310 PRINT "Your personal detail
s will remain confidential"
320 STOP
```

Activities

Write programs for the problems given below. Test your programs on the computer.

1. Input four numbers; print them; find their sum and their average. Print the results.
2. Using the variables listed below write a letter asking a person to attend an interview. Use input so that the letter can contain different information each time it is run.
Variables
n\$ = name of person for interview
d\$ = date of interview
t\$ = time of interview
p\$ = place of interview
m\$ = name of interviewer
s\$ = name of person who has sent the letter
l\$ = date of letter.
3. Input a sum of money that you wish to save. Input the amount of money that you can save each week. Calculate the number of weeks that it will take to save the money. Change the number of weeks into months. Print out all the information.

Unit 15: Program descriptions— algorithms

An *algorithm* is a written description of a program. It is a way of expressing a solution to a problem from which a program can then be written. There are two main ways of writing algorithms

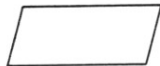
1. by using words
2. by drawing a plan or flowchart.

It is method 2 that we are concerned with in this unit.

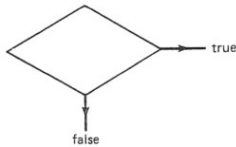
Flowcharts

A *flowchart* uses different shaped boxes.

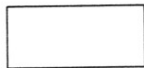
- (a) For *input* (information going in) and *output* (information coming out).



- (b) For *decisions* (deciding whether a statement is true or false).



- (c) For *calculations* (working out the arithmetic expressions).



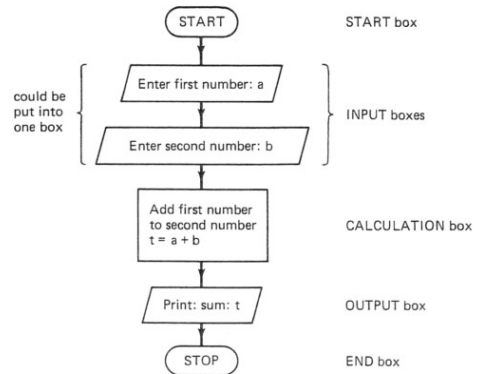
46

- (d) For *start* (beginning of program) and *stop* (end of program).



To see how these boxes are used let us take the problem of entering two numbers into the computer, adding them together and printing the result.

FLOWCHART



A program can then be written using this flowchart as a guide.

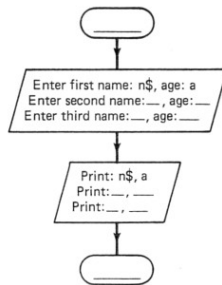
```
PROGRAM 10 REM Adding Numbers
20 INPUT "Enter first number"; a
30 INPUT "Enter second number"; b
40 LET t = a + b
50 PRINT t
60 STOP
```

Activities

1. Enter the names of three girls and their ages and print the information in a table.

Variables to use: n\$: first name
 g\$: second name
 h\$: third name.
 a : first girl's age
 b : second girl's age
 c : third girl's age.

Copy and complete the flowchart



Copy and complete the program

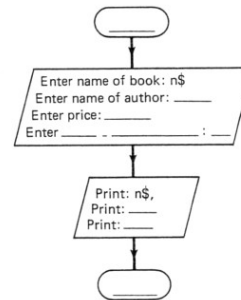
```

10 REM Names and ages
20 INPUT "Name of first girl?"; n$
30 INPUT "Age of first girl?"; -
40 INPUT "Name of second girl?"; -
50 - "Age of - girl?"; -
  INPUT - Name of third girl? -; -
70 - " - - - ?"; -
80 PRINT "NAME", "AGE"
90 PRINT n$, a
100 - -, -
  - -, -
120 -
  
```

2. Enter the name of a book, its author, its price and its date of publication. Print out this information.

Variables to use: n\$: name of book
 a\$: name of author.
 p : price
 d : date of publication.

Copy and complete the flowchart



Copy and complete program

```

10 REM Book and author
20 - "Name of book?"; -
30 INPUT " - - - ?"; -
40 - "Price of book?" - p
  - " - - - "; -
60 PRINT -; "by"; -
70 - "Price of book"; -
80 - " - - - "; -
  - -
  
```

3. List the variables, draw flowcharts and write programs for each of the problems below. Supply your own information where necessary. *Test your programs on the computer.*
- Calculate the area of a triangle and print out the result (area of a triangle = $\frac{1}{2}$ base \times height).
 - Enter the names of three people and their dates of birth and print out this information. (Will the dates of birth be numeric or string variables – think about this before you start.)
 - Enter a word of at least seven letters and print three other words using only the letters in the original word.
 - Enter the names of three towns and their populations. Print this information. Calculate the total population and print the result.
 - A boy has 216 marbles which he shares equally among his twelve friends. Print the number of marbles each friend receives.

- (f) Enter a length measured in feet and change it into centimetres (1 foot = 30.5 centimetres).
- (g) A girl has £95 to spend. She buys a skirt for £26, a dress for £44 and a pair of shoes for £16. Calculate the amount she spends and the amount she has left. Print out all the information.

Unit 16: Loops—using GOTO

GOTO is a command or keyword found on key G. As for other commands, it can only be used when the cursor is a **K** (the computer is in *keyword mode*).

```
TYPE: 10 PRINT "Hello"
      20 GOTO 10
```

RUN

The word 'Hello' will fill the screen and then the message 'scroll?' will be printed. Press Y for yes.

The screen fills again with the same word. This can be repeated as many times as you wish.

Press N for no.

The program will now stop with the message:

```
D BREAK- CONT repeats, 10:1
```

Press ENTER.

The program returns to the screen.

Jumping back to another part of the program is known as *looping*. The example shown above could go on for ever so it is known as an *infinite loop*. The GOTO command can be used to force the computer to carry out any line of the program.

TYPE:

```
10 LET x=0
20 LET x=x+1
30 PRINT x
40 GO TO 20
```

RUN the program.

The screen fills with the numbers 1 to 22.

Press Y.

The screen fills with the numbers 23 to 44. As in the previous example this is another *infinite loop* as it can be repeated any number of times.

How does the program work? Can you write an explanation?

Press N for 'no' and remove the program from the computer.

TYPE:

```
10 REM Running total of number
S
20 LET total=0
30 INPUT "Enter a number ";n
40 LET total=total+n
50 PRINT n,total
60 GO TO 30
```

RUN

This is another infinite loop so when you have entered several numbers

press STOP (shifted A) followed by ENTER which will produce the message:

H STOP in INPUT, 30 : 1

Press ENTER and the program returns to the screen.

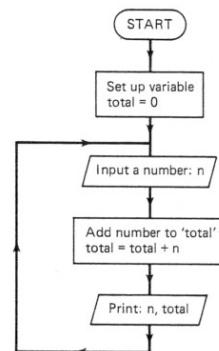
This program is rather like the previous one where x began by being 0 and then 1 was added every time the program was sent back to line 20 by the GOTO command in line 40. In this program the variable called 'total' also begins by being equal to 0 but in line 30 an input prompt asks for a number which is then added to 'total' in line 40. The variable 'total', therefore, changes its contents and is no longer 0. Line 50 prints the input number in one column and the contents of 'total' in the other column. The computer is then told to loop back to line 30 for another number which once again is added to 'total' so the contents of 'total' changes again.

Let us consider this again by actually putting the numbers 5, 10 and 15 into the program.

Result of program

Line 20: variable called 'total' is set up and is equal to 0
 Line 30: the number 5 is entered
 Line 40: the variable called 'total' is changed as it becomes $0 + 5 = 5$
 Line 50: prints 5 5
 Line 60: program loops back to line 30
 Line 30: the number 10 is entered
 Line 40: the variable 'total' is changed again and becomes $5 + 10 = 15$
 Line 50: prints 10 15
 Line 60: program loops back to line 30
 Line 30: the number 15 is entered
 Line 40: the variable 'total' is changed yet again and becomes $15 + 15 = 30$
 Line 50: prints 15 30
 Line 60: program loops back to line 30

Flowchart for program



Notice that there is no box for the GOTO statement. The command is indicated by the direction of the arrows.

The GOTO statement can also be used to jump forward in a program; whole sections of programs can be missed out if required. The next program is an example of this.

TYPE:

```
10 PRINT "The Spectrum is the
";
20 GO TO 40
30 PRINT "worst ";
40 PRINT "greatest ";
50 PRINT "computer"
```

RUN the program. What has happened to line 30?

GOTO is always used with a line number. The line number can be written as an arithmetic expression. For example

GOTO 20 + 40 means GOTO line 60

This can be quite useful if you wish to add a numeric variable to a number line; see the following program.

TYPE:

```

10 INPUT "Enter a month of the
year as a number between 1 and
12 ";m
20 PRINT "Month ";m;" is ";
30 GO TO 50+n ←
50 GO TO 10 ←
51 PRINT "January": GO TO 10
52 PRINT "February": GO TO 10
53 PRINT "March": GO TO 10
54 PRINT "April": GO TO 10
55 PRINT "May": GO TO 10
56 PRINT "June": GO TO 10
57 PRINT "July": GO TO 10
58 PRINT "August": GO TO 10
59 PRINT "September": GO TO 10
60 PRINT "October": GO TO 10
61 PRINT "November": GO TO 10
62 PRINT "December": GO TO 10

```

Notice that each of these lines has two keywords or commands — PRINT and GOTO. This is made possible by using the colon (:) which separates one statement from the next. After entering the colon the cursor automatically becomes a K again.

RUN

To escape from the program enter STOP followed by ENTER.

In this program, line 30 will cause a jump to line 51 if you enter the number '1', to line 52 if you enter the number '2', and so on.

If the number 0 is entered by mistake, how does the computer cope with this? If a number above 12 is entered by mistake what happens to the program?

Notice in this program that in lines 51 to 62 it was possible to give the computer two commands or instructions in each line. For example, PRINT a word and then GOTO another line. This was done by using the colon (:) which simply separates one command from another. The colon is a signal to the computer to return to keyword mode K, so that another keyword or command can be entered.

Activities

1. Draw a flowchart for the program about:
"the Spectrum is the greatest computer"
2. Write a program that inputs two separate numbers, adds them together, prints the result and then jumps back to the beginning of the program, forming an infinite loop. Test your program on the computer.
3. A man has three different saving accounts which contain the following amounts: £381.20, £623.65 and £127.84. He decides to transfer his savings to one account and wishes to know the total amount that he has saved. He

continues to save by adding money to his new account each month but he does not save the same amount of money each month. His bank sends him an account at the end of each month which states the date, how much he has saved each month and the total. This arrangement continues for many years. Draw a flowchart and write a program that displays and calculates all this information. Test your program on the computer.

Unit 17: Comparisons

A computer, unlike a calculator, can compare two pieces of information. It can decide whether a statement is true or false and then act accordingly. In order to do this the computer uses the command or keyword IF, found on key U, followed by the word THEN, on shifted key G. IF, like other commands, can be used only when the computer is in *keyword mode*; the cursor is a K. The command is simply saying that IF a certain statement is true THEN perform the next instruction in that line. See the example below which compares information by using the sign for equals (=) and the sign for not equals (<>).

TYPE:

```

10 PRINT "10+15=";
20 INPUT "Enter an answer ";x
30 PRINT x
40 IF x=10+15 THEN PRINT "Correct"
50 IF x<>10+15 THEN PRINT "Wrong"
60 STOP

```

As you type in these lines notice what happens to the cursor after you have pressed in the word THEN

key U

shifted key G

the content of numeric variable x is being compared to the sum of 10 + 15

next instruction

shifted key W (do not use shifted T and shifted R)

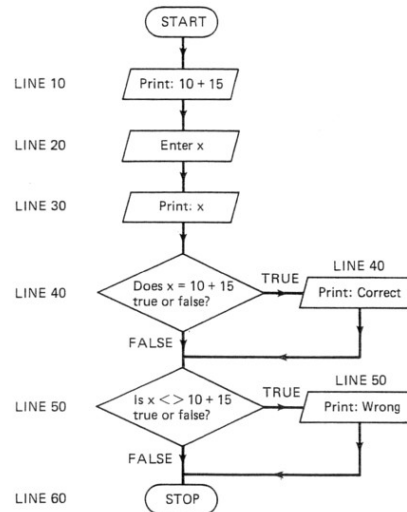
RUN the program; once with the correct answer and once with the wrong answer.

In this program line 40 means that if it is true that x equals 10 + 15 then print 'correct'. If it is not true and x does not equal 10 + 15 then ignore the rest of the instructions in that line. The program then passes on to line 50 which tests

the statement again; if it is true that x does not equal 10 + 15 then print 'wrong', the answer is 10 + 15'. If the statement is not true then ignore the rest of the line.

Question: Why was it possible to use the command PRINT in the middle of lines 40 and 50? Why did not the letter P appear?

Flowchart for this program



This next example shows that it is not only numbers contained in numeric variables that can be compared but also the contents of string variables.

TYPE:

```

10 PRINT "What is my name? ";
20 INPUT "Enter a name ";n$
30 PRINT n$
40 IF n$="Floribunda" THEN GO TO 70
50 PRINT "No, it is Floribunda"
60 STOP

```

```

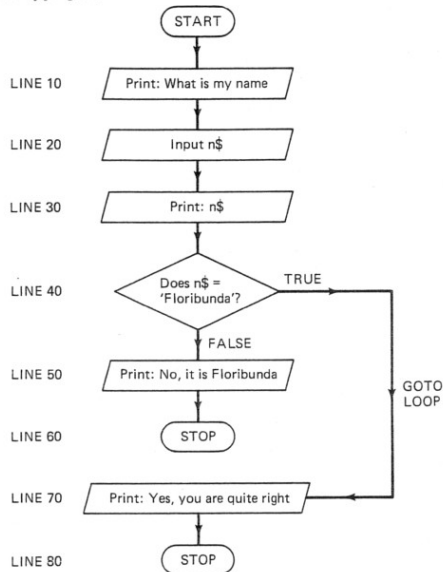
70 PRINT "Yes, you are quite ri
ght"
80 STOP

```

RUN the program several times trying different names.

This program illustrates a GOTO loop which is set up in line 40. This line means that if it is true that n\$ equals 'Floribunda' then go to line 70. If it is not true then ignore the instruction to go to line 70 and pass on to line 50.

Flowchart of program



Question: Why is the STOP statement in line 60 necessary?

Activities

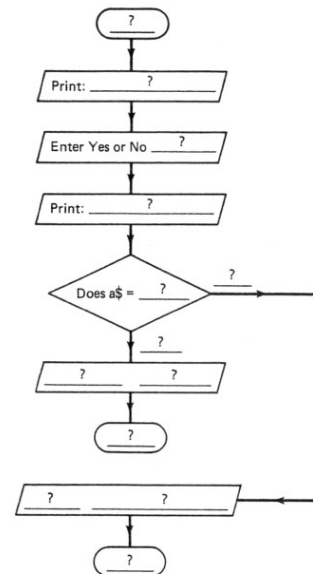
1. Study the program that follows and then copy out and complete the flowchart.

```

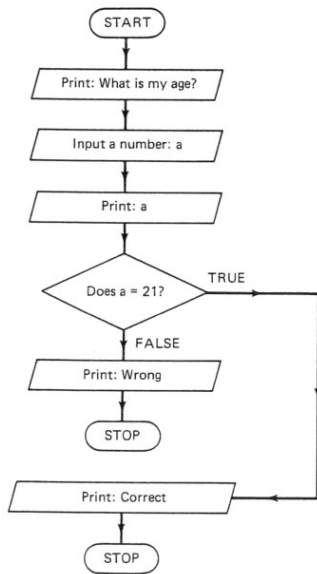
10 PRINT "Do you like cakes an
d biscuits? ";
20 INPUT "Enter yes or no ";a$
30 PRINT a$
40 IF a$="yes" THEN GO TO 70
50 PRINT "You will stay slim"
60 STOP
70 PRINT "You will get fat!!"
80 STOP

```

Flowchart



2. Study the flowchart below and then write the program.



3. Draw a flowchart and write a program for the following problem.
Print an arithmetic expression where one number is subtracted from another. Enter and print the answer. Print whether the answer is right or wrong. If the answer is wrong print out the correct answer. Test your program on the computer.
4. Write a program that asks a person the colour of your eyes. Use a string variable to represent the colour of your eyes. Print out whether they guess rightly or wrongly. Test your program on the computer.

The next program is a little more complicated as it compares information in lines 30 and 80 and uses several GOTO loops.

TYPE:

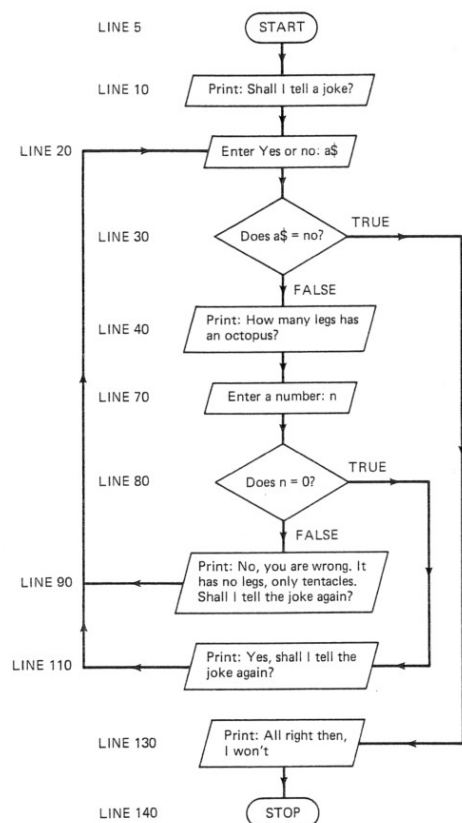
```

5 REM TELL A JOKE
10 PRINT "Shall I tell a joke?"
"
20 INPUT "Enter yes or no ";a$
30 IF a$="no" THEN GO TO 130
40 PRINT "How many legs has an octopus?"
50 PRINT
60 PRINT
70 INPUT "Enter a number ";n
80 IF n=0 THEN GO TO 110
90 PRINT "No,you are wrong.It has no legs only tentacles.Shall I tell the joke again?"
100 GO TO 20
110 PRINT "Yes,shall I tell the joke again?"
120 GO TO 20
130 PRINT "All right then,I won't"
140 STOP
  
```

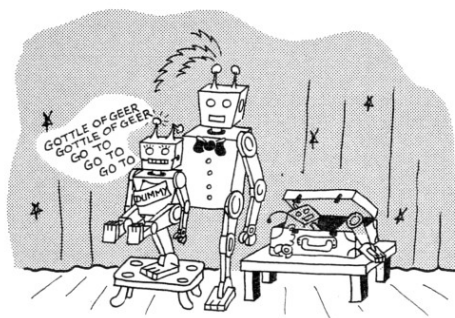
RUN the program several times changing the input in lines 20 and 70.

How does the program work? Use the flowchart that follows to help you.

Flowchart



Another useful technique can be learned here. It is the use of a *dummy value* to stop an infinite loop which has been set up by a GOTO statement.



The next program, which changes marks gained by pupils in an examination into percentages, illustrates this point.

TYPE:

```

10 REM CALCULATING PERCENTAGES
20 PRINT "Enter total marks of
examination ";
30 INPUT t
40 PRINT t
50 PRINT
60 PRINT "Enter mark gained by
pupil ";
70 INPUT m
80 IF m=-9999 THEN GO TO 140
90 PRINT m
100 LET p=(m/t)*100
110 PRINT p;"%"
120 PRINT
130 GO TO 60
140 STOP
  
```

RUN the program and when you have entered enough marks enter -9999 and notice what happens.

As you can see, if the dummy value -9999 is entered in line 70 the computer does not continue on through the program but jumps to line 140 and stops. If line 80 did not exist the computer would continue to perform the program indefinitely as the loop set up by the GOTO statement in line 130 cannot be stopped. The dummy value does not have to be -9999, it can be any value

which is unlikely to be entered as a pupil's mark. It would probably be useful to add an extra line to the program, telling the user of the program what the dummy value is. For example

```
65 PRINT "or -9999 if you wish the program to stop"
```

The dummy value does not have to be a number; a word can be used if the program is using string variables. For example

```
10 REM LIST OF NAMES
20 INPUT "Enter a person's nam
e ";n$
30 IF n$="sausages" THEN GO TO
60
40 PRINT n$
50 GO TO 20
60 STOP
```

Using a dummy value is not the only way of stopping an infinite loop, you will learn another method in unit 19.

Activities

- Write a program that continually asks for the names of towns in Great Britain and prints them out in a list. Include a dummy value so that the program can be stopped. Test your program on the computer.
- Draw a flowchart and write a program that tests a person's multiplication tables (such as, $6 \times 7 = 42$, $4 \times 5 = 20$ etc.). Numbers, using numeric variables, must be entered to make up the sum and the sum printed. An answer must then be entered and the person told whether they are right or wrong. If the answer is wrong the person is allowed another try; therefore the correct answer is not printed. If the answer is right the person should be given the choice of having another turn so the computer needs to return to the beginning of the program where another sum will be set up. Test your program on the computer.

Unit 18: More comparisons

Apart from equals (=) and not equals (<>) there are other ways of comparing information in an IF-THEN statement.

Meaning	Symbol	Key
greater than	>	shifted T
less than	<	shifted R
greater than or equal to	>=	shifted E
less than or equal to	<=	shifted Q

These four new symbols are used in the examples below.

Examples

- | | |
|------------------------|------------------------|
| (1) Is $8 > 4$? Yes. | (5) Is $8 > 9$? No. |
| (2) Is $8 < 4$? No. | (6) Is $8 < 8$? Yes. |
| (3) Is $8 >= 8$? Yes. | (7) Is $8 <= 6$? No. |
| (4) Is $8 >= 7$? Yes. | (8) Is $8 <= 9$? Yes. |

Activities

- Give 'Yes' or 'No' answers to the following:

- | | |
|-------------------------------|-------------------------------------|
| (a) Is $10 > 12$? | (n) Is $a - b <= 6$? |
| (b) Is $12 < 15$? | (a = 10, b = 4) |
| (c) Is $24 >= 24$? | (o) Is $r * s >= 7$? |
| (d) Is $32 >= 31$? | (r = 3, s = 2) |
| (e) Is $15 > 9$? | (p) Is $(10 + 2)/6 > 0$? |
| (f) Is $17 <= 17$? | (q) Is $(w + z) * 2 > 40$? |
| (g) Is $102 >= 105$? | (w = 5, z = 10) |
| (h) Is $33 <= 43$? | (r) Is $n * n < > 64$? |
| (i) Is $47 = 52$? | (n = 8) |
| (j) Is $53 < > 54$? | (s) Is $c + d + e < 24$? |
| (k) Is $3 * x = 15$? (x = 5) | (c = 7, d = 9, e = 11) |
| (l) Is $4 + 5 > 9$? | (t) Is $(f + g) * (10 - 3) <= 56$? |
| (m) Is $6 + y < 19$? (y = 6) | (f = 2, g = 6) |

2. Complete the following statements by putting in the correct symbol (do not use 'not equals' (<>) or 'equals' (=)).

Example 13 ___ 12 should be $13 > 12$
 5 ___ 5 should be $5 \leq 5$ or $5 \geq 5$
 12 ___ 13 should be $12 < 13$

- | | |
|---------------------------------------|---|
| (a) 16 ___ 23 | (g) $x/2$ ___ $y + 4$
($x = 6, y = 2$) |
| (b) 14 ___ 10 | (h) $x * x$ ___ $y/2$
($x = 10, y = 150$) |
| (c) 10 ___ 10 | (i) $(a + b) * 3$ ___ $a + b * 3$
($a = 3, b = 5$) |
| (d) $5 + 6$ ___ $11 - 3$ | (j) $a + b - c$ ___ $a + c - 10$
($a = 1, b = 6, c = 4$) |
| (e) $3 * 7$ ___ $13 + 8$ | |
| (f) x ___ y
($x = 0, y = 1$) | |

By studying the following flowcharts and programs it is possible to see how to use these various symbols. You will notice that they are included with the IF-THEN statements as in the previous unit.

TYPE:

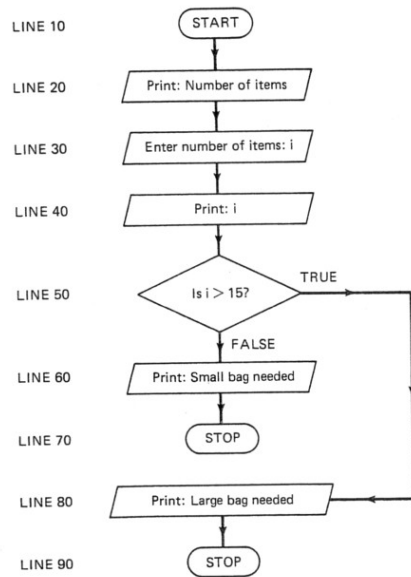
```

10 REM SIZE OF SHOPPING BAG
20 PRINT "Number of items bought is ";
30 INPUT i
40 PRINT i
50 IF i>15 THEN GO TO 80
60 PRINT "Small bag needed"
70 STOP
80 PRINT "Large bag needed"
90 STOP

```

RUN the program entering different numbers and study the flowchart that follows.

Flowchart for previous program



The next program involves entering the names of three children and the amount of pocket money that they receive. It then calculates the average amount of pocket money and compares this with £10 and prints out an appropriate message.

TYPE:

```

10 REM POCKET MONEY
20 INPUT "Enter name of first
child";f$
30 INPUT "Enter name of second
child";s$
40 INPUT "Enter name of third
child";t$
50 INPUT "Enter pocket money o
f first child";p1
60 INPUT "Enter pocket money o
f second child";p2
70 INPUT "Enter pocket money o
f third child";p3
80 PRINT "NAME", "POCKET MONEY"
90 PRINT TAB 16; "IN POUNDS"
100 PRINT f$, p1
110 PRINT s$, p2
120 PRINT t$, p3
130 PRINT
140 PRINT "Average pocket money
is £";
150 LET a=(p1+p2+p3)/3
160 PRINT a
170 IF a>6 THEN PRINT "Too much
money"
180 IF a<=2 THEN PRINT "Too lit
tle money"
190 STOP

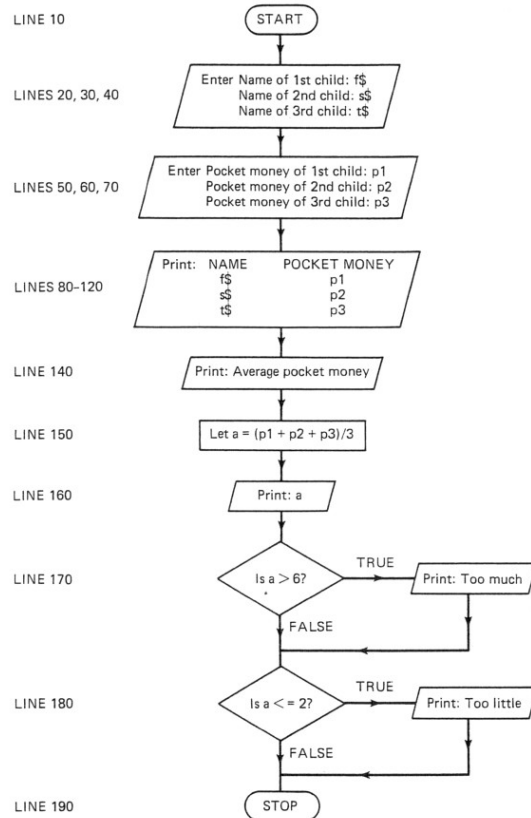
```

RUN the program several times using different names and different amounts of pocket money.

See the flowchart that follows.

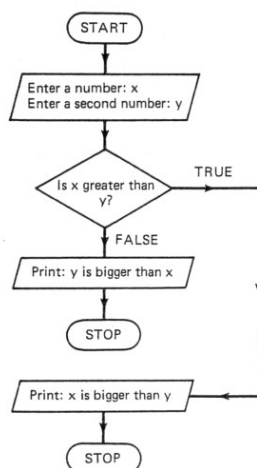
Lines 20 to 120 just receive the necessary information and then print it out in two columns which are given the headings 'NAME' and 'POCKET MONEY IN POUNDS'. Line 150 calculates the average amount of money while lines 170 and 180 compare this with £6 and £2 respectively.

Flowchart for previous program

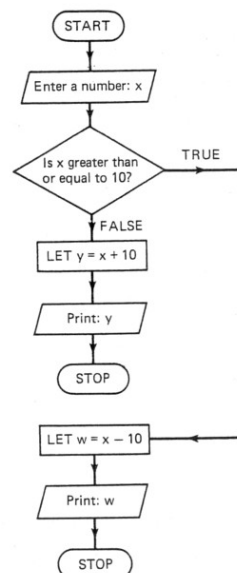


Activities

3. Write a program for this flowchart.



4. Write a program for this flowchart.



5. A salesman is paid a weekly salary of £65. If he sells goods to the value of £200 or more he is given an extra bonus of £10. The program below calculates the salesman's final salary using the following variables:

s = salary of £65
b = bonus of £10
x = value of goods sold

Draw a flowchart for this program:

```

10 REM SALESMAN'S SALARY
20 LET s=65
30 LET b=10
40 PRINT "Value of goods sold
  by a salesman in a week ";
50 INPUT "Enter value of goods
  ";x
60 PRINT x;" pounds"
70 IF x>200 THEN GO TO 100
80 PRINT "No bonus added.Final
  salary is ";s;" pounds"
90 GO TO 120
100 LET s=s+b
110 PRINT "Bonus added.Final sa
  lary is ";s;" pounds"
120 STOP

```

List the variables, draw flowcharts and write programs for the problems in 6-11. Test each program on the computer.

6. Enter the date of a year (for example, 1983) and compare it to the number 2000. If the year entered is greater or equal to 2000 then print 'Next century'; if the year entered is less than 2000 then print 'This century'.
7. Enter two separate numbers. If their product is greater than 81 then print 'Too great'; if it is less than 81 then print 'Too little'; and if it is equal to 81 print 'Just right'.
8. If a boy earns less than £50 a week his father gives him an extra £10. If he earns £70 or more he has to give his mother £15. Enter the boy's salary and calculate and print the boy's final weekly income.
9. A car holds 33 litres of petrol. A litre of petrol costs 45 pence. A car uses 1 litre of petrol to travel 14 km. A man puts some petrol in his car; calculate the cost of the petrol and the distance he can travel on that amount of petrol; print out this information. If he can travel 380 km without refilling his tank he will take a holiday in Cornwall, otherwise he will go to Dorset. Print out where he goes.
10. A baker's oven will hold 15 loaves of bread which take 52 minutes to bake. He fills his oven 4 times a day. Calculate the number of loaves he bakes in a day and the time it takes in hours. Print this information. The baker's bread is sold in the local supermarket. If all the bread is sold, a message is sent to the baker to bake more. Print out the message if required.
11. A milkman delivers milk to several blocks of flats. 18 families have 3 pints a day, 17 families have 2 pints a day and 9 families have four pints a day. 15 families vary the amount they have each day. A milk crate contains 12 pints of milk. Calculate the number of pints he delivers each day and the number of crates he has to load on to his milk van. Print this information. If more than 13 crates are required he will have to return to the depot for more, otherwise he can go home. Print out where he goes.

Sometimes we need to compare two or more pieces of information in the same IF-THEN statement. For example, if one thing is true and another thing is true then do so and so, which translated into BASIC could be:

```
IF y = 10 AND x = 20 THEN GOTO 200
```

or

```
IF a$ = "John" AND b$ = "McEnroe" THEN PRINT "Tennis star"
```

The only word we have added to the statement is AND (shifted Y).

TYPE:

```

10 REM Using AND
20 INPUT "Enter a number betwe
  en 1 and 100 ";n
30 INPUT "Enter another number
  between 1 and 100 ";n2
40 IF n>50 AND n2<50 THEN GO T
  O 90
50 PRINT n+n2
60 LET c=5
70 PRINT n*c
80 STOP
90 PRINT n-n2
100 STOP

```

RUN. Enter 20 and 70

Result is: 90

100

RUN. Enter 70 and 20

Result is: 50

RUN. Enter 70 and 70

Result is: 140

350

RUN. Enter 20 and 20

Result is: 40

100

Line 40 tests two conditions, both of which have to be true before the computer will obey the rest of the instruction. Therefore, when $n = 20$ and $n2 = 70$, 'n' is not greater than 50 and $n2$ is not less than 50 so the computer ignores the GOTO statement and carries on to lines 50 to 80. When $n = 70$ and $n2 = 20$, 'n' is greater than 50 and $n2$ is less than 50 so the GOTO statement is obeyed and the computer jumps to line 90. When $n = 70$ and $n2 = 70$, 'n' is greater than 50 but $n2$ is not less than 50, therefore both conditions are not true so the computer continues to line 60. When n is 20 and $n2$ is 20, 'n' is not greater than 50 although $n2$ is less than 50 so once again both conditions are not true so the next line to be executed is line 50.

Activity

12. Using the previous program give the results if the following numbers were entered:
- (a) $n = 10, n2 = 20$
 - (b) $n = 80, n2 = 40$
 - (c) $n = 80, n2 = 60$
 - (d) $n = 20, n2 = 10$
- Check your answers on the computer.

AND can be used equally well with string variables which can be seen in the next program.

TYPE:

```
10 REM Using AND
20 INPUT "Enter the Christian
name of a male tennis star ";c$
30 INPUT "Enter the surname of
the same tennis star ";s$
40 IF c$="John" AND s$="McEnroe"
THEN PRINT "Number 1 player in
the world"
50 IF c$="Jimmy" AND s$="Connors"
THEN PRINT "Number 2 player
in the world"
60 STOP
```

RUN. Enter John and McEnroe

Result is: Number 1 player in the world

RUN. Enter Jimmy and Connors

Result is: Number 2 player in the world

RUN. Enter John and Connors

Result is: neither message is printed.

This program works on exactly the same principle as the previous one; the message will be printed only if both conditions are true.

The word OR (shifted key U) works in much the same way except that only one of the conditions has to be true.

TYPE:

```
10 REM Using OR
20 INPUT "Enter your height in
centimetres ";h
30 INPUT "Enter the size of your
shoes ";s
40 IF h>165 OR s>6 THEN PRINT
"Too large-you must shrink": GO
TO 60
50 PRINT "Too small-you must grow"
60 STOP
```

RUN. Enter 155 and 5

Result is: Too small – you must grow

RUN. Enter 266 and 5

Result is: Too large – you must shrink

RUN. Enter 155 and 12

Result is: Too large – you must shrink.

When $h = 155$ and $s = 5$ neither of the conditions were true so the computer ignored the rest of the statement but when $h = 266$ and $s = 5$ one of the conditions was true, 266 was greater than 165, so "Too large – you must shrink" was printed. It did not matter that 5 was not larger than 6. Similarly when $h = 155$ and $s = 12$ one of the conditions was true.

Activities

13. Write a program that inputs a number which is tested to see if it is greater or equal to 100 but less than or equal to 200. If the number fits the condition then the message "Number between 100 and 200" should be printed. If the number does not fit the condition then the message "Number not between 100 and 200" should be printed.
14. Write out the result of this program if the following numbers were entered in line 30: (a) $n = 40$, (b) $n = 60$, (c) $n = 80$.

```
10 LET x=50
20 LET y=70
30 INPUT "Enter a number ";n
40 IF n>x AND n<y THEN GO TO 7
50 PRINT n+n
60 STOP
70 PRINT x+y+n
80 STOP
```

15. Write out the result of this program if the following ages were entered in line 30: (a) $a = 13$, (b) $a = 21$, (c) $a = 17$, (d) $a = 6$, (e) $a = 19$.

```
10 LET y=19
20 LET x=13
30 INPUT "Enter your age ";a
40 IF a>=13 AND a<=19 THEN PRINT "Teenager"
50 IF a<13 THEN PRINT "Child"
60 IF a>19 THEN PRINT "Adult"
70 STOP
```

16. Write out the result of this program if:

- (a) $w\$ = \text{hot}, r\$ = \text{dry}$. (b) $w\$ = \text{warm}, r\$ = \text{wet}$.
- (c) $w\$ = \text{cold}, r\$ = \text{dry}$. (d) $w\$ = \text{warm}, r\$ = \text{dry}$.

```

10 LET a$="hot"
20 LET b$="cold"
30 LET c$="wet"
40 LET d$="dry"
50 LET e$="warm"
60 INPUT "Is the weather hot,w
arm or cold? ";w$
70 INPUT "Is the weather wet o
r dry? ";r$
80 IF w$a$ OR w$b$ OR r$c$
THEN PRINT "No walk today": GO T
O 100
90 PRINT "Let's go for a walk"
100 STOP

```

Unit 19: Adding a counter

As you have already seen, a computer can be made to perform a program over and over again by using a GOTO loop. Sometimes it is difficult to break out of such a loop as the computer goes on indefinitely and this is not particularly useful. However, by adding a counter we can make the computer execute a program for a set number of times and then stop. This is illustrated in the following program, which is performed five times.

TYPE:

```

10 REM COUNTING
20 LET counter=0
30 INPUT "Enter the name of a
car ";a$
40 PRINT a$
50 LET counter=counter+1
60 IF counter=5 THEN GO TO 80
70 GO TO 30
80 STOP

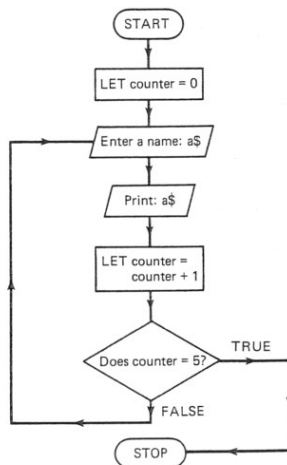
```

RUN the program. *Do not remove from the computer.*

Line 20 sets the numeric variable called 'counter' to 0. A name is then entered and printed. Line 50 adds 1 to the counter and so the variable now has the value 1. Line 60 asks whether the value of 'counter' is equal to 5; as it is not the computer ignores the rest of the line and continues on to line 70 where it is told to go back to line 30 and start all over again. The program is then executed a second time and the value of 'counter' changes to 2 which once again is not equal to 5 so the computer returns to line 30. This continues until 'counter' is equal to 5 and then the computer jumps to line 80 and stops.

Question: Why did line 70 say GOTO 30 and not to line 20?

Flowchart



It is not necessary to state the number of times a program is to be performed when the program is written. This can be entered into the program while it is being run. Modify the previous program by adding a line and by altering line 60.

ADD: 15 INPUT "Enter the number of times you wish to perform the program ";n
 ALTER: 60 IF counter=n THEN GO TO 80

RUN the program again, entering a reasonably low number otherwise you will have to input many names.

To make typing quicker the variable 'counter' is usually just called 'c' which is what we will use in future.

A counter is not only used to count the number of times a program is performed, it can also be useful in a quiz program to count the number of correct or incorrect answers. See the example below.

TYPE:

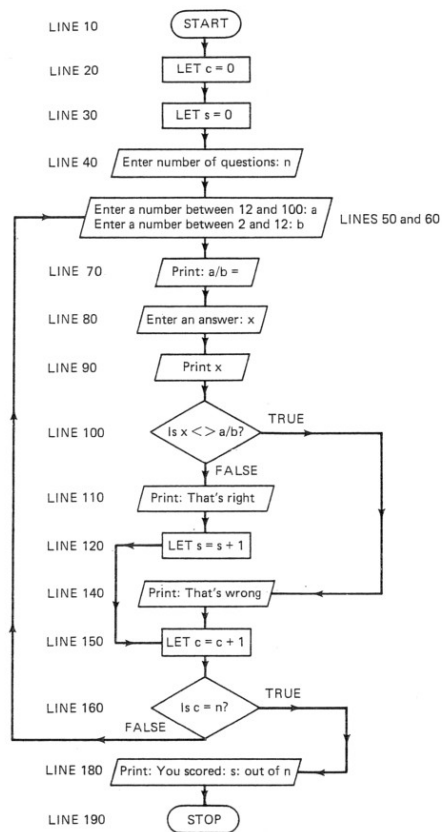
```

10 REM PRACTISING DIVISION
20 LET c=0
30 LET s=0
40 INPUT "Enter the number of questions to be given ";n
50 INPUT "Enter a number between 12 and 100 ";a
60 INPUT "Enter a number between 2 and 12 ";b
70 PRINT a;" / ";b;" = ";
80 INPUT "Enter an answer ";x
90 PRINT x
100 IF x<>a/b THEN GO TO 140
110 PRINT "That's right"
120 LET s=s+1
130 GO TO 150
140 PRINT "That's wrong. The answer is ";a/b
150 LET c=c+1
160 IF c=n THEN GO TO 180
170 GO TO 50
180 PRINT "You scored ";s;" out of ";n
190 STOP
  
```

RUN

Lines 20, 40, 150 and 160 set up and deal with the counter which counts the number of times that the program is performed. Line 30 sets up the counter 's' for keeping the score, in this case the number of correct answers. Variable 's' begins at 0 as no questions have been asked. If the first sum is answered correctly then 1 is added to the score in line 120 and therefore variable 's' becomes 1. If the sum is answered wrongly then line 100 sends the computer to line 140, thus missing out line 120 where 1 was added to 's'. This sequence of events is repeated each time that a new sum is answered and so 's' will increase by 1 every time that a sum is answered correctly. When the program has run the required number of times line 180 prints the final score. The following flowchart will help you to understand this program.

Flowchart



Activities

Remember to try your programs on the computer.

1. Write a program that asks for the names of six flowers and prints them out in a list.
2. Modify the program that you have written for activity 1 so that the number of flowers to be listed is decided as the program is run.
3. Draw a flowchart and write a program where a person guesses a number between 0 and 20. If the guess is wrong, then another attempt must be made until the right answer is given. The computer must record and print out the number of guesses made. Make certain that the person making the guesses is told not to look when the number to be guessed is being entered!

Unit 20: FOR-NEXT loops

Using FOR-NEXT statements is another way of controlling the number of times a program or part of a program is performed. FOR is a keyword found on key F and NEXT is another keyword on key N; neither can be used unless the computer is in *keyword mode K*.

TYPE:

```

10 FOR a=1 TO 10
20 PRINT a
30 NEXT a

```

Notice that these two variables must be the same

shifted F (do not type T and O as separate letters)

RUN

As you can see, the result of this program is to print out a list of numbers from 1 to 10. In line 10 the FOR statement sets the counter to start at 1 and finish at 10. Line 20 then prints the value of 'a' which on the first time round is 1. Line 30 then sends the computer back to the FOR statement to collect the NEXT value of 'a' which will be 2. This loop is repeated until 'a' has the value 10 and then it automatically stops. The loop is performed 10 times.

TYPE:

```

10 FOR b=11 TO 20
20 PRINT b
30 NEXT b

```

RUN

This program will also run 10 times, the only difference is that the value of 'b' will start at 11 and finish at 20.

TYPE:

```

10 FOR c=1 TO 6
20 PRINT "Hello"
30 NEXT c

```

RUN

This FOR-NEXT loop will run 6 times. Line 20 no longer asks the computer to print out the value of 'c' but to print the word 'Hello', so this will appear on the screen six times. You can see that any lines that lie between the FOR and the NEXT statements are part of the loop and will be performed each time the computer passes through the loop.

Activities

- Study the programs below and write out what would appear on the screen when they are run.

(a) 10 FOR x=1 TO 12 20 PRINT x 30 NEXT x	(c) 70 FOR w=1 TO 4 80 PRINT "Goodbye" 90 NEXT w
(b) 40 FOR y=4 TO 16 50 PRINT y 60 NEXT y	(d) 100 FOR c=1 TO 3 110 PRINT c 120 PRINT "This is a loop" 130 NEXT c

If you wanted a large space in a program between two written statements then a FOR-NEXT loop could be used to create this space. See the program below.

TYPE:

```

10 PRINT "I'm at the top of the
screen"
20 FOR a=1 TO 9
30 PRINT
40 NEXT a
50 PRINT "I'm in the middle of
the screen"
60 FOR b=1 TO 9
70 PRINT
80 NEXT b
90 PRINT "I'm at the bottom of
the screen"
100 STOP

```

RUN

In this program the loop set up by lines 20 and 40 is repeated nine times, so nine empty lines are printed, and then the same thing happens again because of the loop set up by lines 60 and 80. This saves typing out nine PRINT statements. For example

```

10 PRINT "I'm at the top of the screen"
20 PRINT
30 PRINT
40 PRINT
etc.

```

A loop is set up by using a numeric variable, such as

```
FOR a (numeric variable) = 1 to 10
```

but other numeric variables may also be used in the statement — see below.

TYPE:

```
10 INPUT "Enter number of times you wish loop to be repeated "
;n
20 FOR x=1 TO n
30 PRINT x
40 NEXT x
```

RUN the program several times, entering different numbers, and notice the results. If you enter a number greater than 22 the screen will fill up and you will be asked to 'scroll'.

ADD to the program:

```
5 INPUT "Enter a number at which you wish the loop to start "
;s
```

ALTER line 10:

```
10 INPUT "Enter a number at which you wish the loop to finish "
;n
```

ALTER line 20:

```
20 FOR x=s TO n
```

RUN the program making certain that the first number entered is lower than the second.

This last program shows that the whole of the FOR statement can be made up of numeric variables. This enables the programmer to enter different numbers while the program is running so that the loop can be repeated any number of times.

Usually when a FOR-NEXT loop is used the counting is done in ones but it is possible to make the computer count in twos, threes, fours etc. by using the word STEP which is shifted D. The following programs illustrate this.

TYPE:

```
10 FOR x=1 TO 10 STEP 2
20 PRINT x
30 NEXT x
```

RUN

The result is: 1 ↑
3 Only the odd numbers are printed.
5 Notice that although the loop

7 should finish at 10 this is not
9 printed as after 9 a step of two
would make the next number 11.

ALTER line 10:

```
10 FOR x=3 TO 21 STEP 3
```

RUN

The result is: 3 ↑
6
9 This program begins at 3 and then
12 counts in threes until 21 is
15 reached.
18
21

ALTER line 10:

```
10 FOR x=10 TO 1 STEP -1
```

RUN

The result is: 10 ↑
9
8 This program prints the highest number
7 first and then decreases by one each
6 time until the lowest number is reached.
5 This program cannot be written:
4 FOR x = 1 TO 10 STEP -1
3
2
1

ALTER line 10:

```
10 FOR x=20 TO -5 STEP -5
```

RUN the program.

The result is: 20 ↑
15 This loop starts at 20 and
10 decreases by 5 each time
5 until -5 is reached.
0
-5

ALTER line 10:

```
10 FOR x=-10 TO -20 STEP -2
```

RUN

The result is: -10 ↑
 -12 This is a similar program but
 -14 remember that -10 is a higher
 -16 number than -20.
 -18
 -20

Activities

2. Study the programs below and then write out the result of each.

- (a) 10 FOR x=2 TO 12 STEP 2
 20 PRINT x
 30 NEXT x
- (b) 10 FOR x=15 TO 25 STEP 4
 20 PRINT x
 30 NEXT x
- (c) 10 FOR x=0 TO 100 STEP 20
 20 PRINT x
 30 NEXT x
- (d) 10 FOR x=50 TO -25 STEP -25
 20 PRINT x
 30 NEXT x
- (e) 10 FOR x=-2 TO -12 STEP -3
 20 PRINT x
 30 NEXT x
- (f) 10 FOR x=-10 TO 26 STEP 5
 20 PRINT x
 30 PRINT "Stepping up"
 40 NEXT x

3. Correct the following programs:

- (a) 10 FOR x = 1 TO 10
 20 PRINT x
 30 NEXT y
- (b) 10 FOR x = 1 TO 10 STEP -1
 20 PRINT x
 30 NEXT x
- (c) 10 FOR x\$ = 1 TO 6 STEP 2
 20 PRINT x\$
 30 NEXT x\$
- (d) 10 FOR x = x TO 10
 20 PRINT x
 30 NEXT x
- (e) 10 FOR x = 1 TO x
 20 PRINT x
 30 NEXT x
- (f) 10 LET x = 1 TO 10
 20 PRINT x
 30 NEXT x

Variations on a theme.

TYPE:

```
10 FOR x=1 TO 6
20 LET a=x+5
30 PRINT x,a
40 NEXT x
```

RUN

The result is: 1 6
 2 7
 3 8
 4 9
 5 10
 6 11

Lines 10 and 40 set up the loop to be repeated 6 times. Line 20 introduces a new variable 'a' which is given the same value as 'x' plus 5. Line 30 then prints out the two variables in two columns. The first time round the loop 'x' is 1 and therefore 'a' is 6; the second time round 'x' is 2, therefore 'a' is 7. This continues until the last value of x is reached.

TYPE:

```
10 FOR x=3 TO 12 STEP 3
20 PRINT x,x*x
30 NEXT x
```

RUN

The result is: 3 9
 6 36
 9 81
 12 144

Explain how this program works.

Activities

4. Study the programs and write out the results.

- (a) 10 FOR x=2 TO 4
 20 PRINT x,x+4
 30 NEXT x
- (b) 10 FOR x=10 TO 16 STEP 2
 20 LET a=x+2
 30 PRINT x,a
 40 NEXT x
- (c) 10 FOR x=12 TO 3 STEP -3
 20 LET a=10
 30 PRINT x,a
 40 NEXT x

```
(d) 10 FOR x=100 TO 200 STEP 25
     20 LET a=x-10
     30 LET b=x+10
     40 PRINT x; "times"; a; "times"; b
     50 NEXT x
```

Using a FOR-NEXT loop to print out multiplication tables:

TYPE:

```
10 REM MULTIPLICATION TABLES
20 FOR x=1 TO 12
30 PRINT x;"times 8=";x*8
40 NEXT x
```

RUN the program. The result is the eight times table.

This program can be altered so that it will give any table.

ADD to program:

```
15 INPUT "Enter the number of
the table you require ";n
```

ALTER line 30:

```
30 PRINT x;"times";n;"=";x*n
```

RUN the program; if the number 5 is entered the five times table will be printed; if the number 9 is entered the nine times table will be printed and so on.

Activities

Write programs for each of the following and test them on the computer.

5. Print out all the numbers from 13 to 23.
6. Print out all the odd numbers from 3 to 37.
7. Print out the numbers from 5 to 35 counting in fives.
8. Print out all the even numbers from 10 to -6.
9. Print out the seven times table.
10. Print out three sentences which are five blank lines apart.
11. Print out the word 'Spectrum' eight times.
12. Write a program that prints out all the leap years in this century. Leap years occur every four years and the first one in this century was 1904.
13. Print out the value of x that begins at 10 and ends at 20. Print out in a separate column the value of $x + 3$.

The next two programs show that the activity that lies in the loop between the FOR-NEXT statements can be anything. It may be input statements asking for and printing out information or calculations or both.

TYPE:

```
10 FOR x=1 TO 5
20 INPUT "Enter the name of a
country ";c$
30 INPUT "Enter the population
of the country ";p
40 PRINT "COUNTRY","POPULATION"
"
50 PRINT c$,p
60 PRINT
70 NEXT x
80 STOP
```

RUN the program using the information listed below:

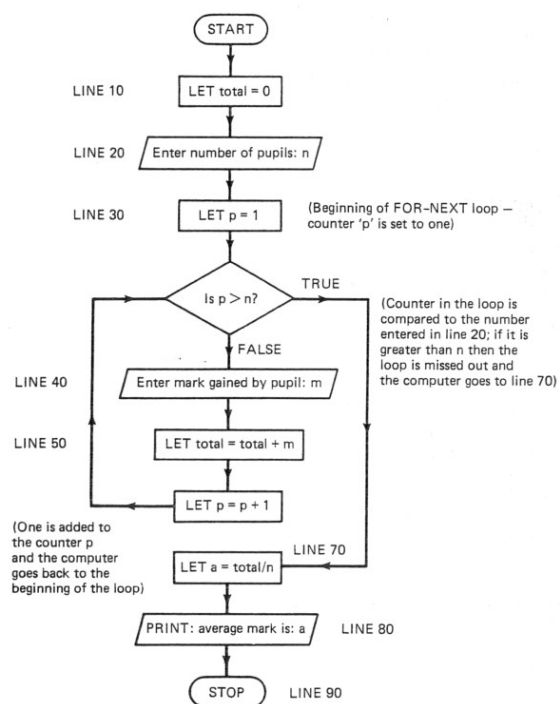
Country	Population	
Denmark	5 000 000	(Do not put commas
Sweden	8 000 000	between the set of
U.K.	56 000 000	noughts. Do you
France	53 000 000	know why?)
Belgium	10 000 000	

TYPE:

```
10 LET total=0
20 INPUT "Enter number of pupils ";n
30 FOR p=1 TO n
40 INPUT "Enter mark gained by
pupil ";m
50 LET total=total+m
60 NEXT p
70 LET a=total/n
80 PRINT "The average mark is
";a
90 STOP
```

RUN the program varying the number of pupils entered in line 20 and the list of marks.

Flowchart for previous program



As you can see, the FOR-NEXT loop appears in the flowchart just like an ordinary counter and so looks very similar to the diagrams drawn in unit 19.

Activities

Write programs for each of the following and test them on the computer.

14. Input the names of six people and their ages. Print this out in two columns with headings.
15. Using the information given below input and print out the names of four cars and their prices. Calculate and print the average price.

Car	Price
Ford Fiesta	£4600
Vauxhall Nova	£5000
Nissan Micra	£4550
Talbot Horizon	£5200

16. The birth and death rates per 1000 people, for various countries, are given below. Input this information and calculate the rate of natural population increase (birth rate minus death rate). Print out all the information.

	Birth rate	Death rate
India	43	17
Japan	19	7
Britain	15	12
U.S.A.	16	9
Malawi	49	25
Ghana	47	18
Bolivia	44	19

Unit 21: READ and DATA statements

READ and DATA statements can be used only when the computer is in extended mode (press CAPS SHIFT and SYMBOL SHIFT together). READ is above key A and DATA above key D.

READ and DATA statements provide another way of putting information into a program. However, unlike the INPUT statement where information is fed into the program while it is running, information used in a DATA statement is actually written into the program at the beginning and cannot be altered as the program is run. In this respect it is rather like a LET statement.

A program will best illustrate how these new statements are used.

TYPE:

```
10 READ a,b,c
20 DATA 10,20,30
30 PRINT a,b,c
40 PRINT b,c,a
50 STOP
```

RUN

The result is:	10	20
	30	
	20	30
	10	

DO NOT REMOVE

The READ statement gives the names of the variables; in this case numeric variables called 'a', 'b' and 'c'. The DATA statement supplies the information to go into these variables, in this example the numbers 10, 20 and 30.

As soon as the computer reaches the first variable in a READ statement it looks for the first piece of information in the first DATA statement. Thus the value 10 is stored in 'a'. As the computer reaches the second variable it searches for the second piece of information, therefore 'b' takes the value 20. This continues until all the variables have been given values.

Line 30 proves that the variable 'a' contains 10, the variable 'b' 20 and variable 'c' 30. The contents of these variables do not change, as can be seen in line 40, when the computer is asked to print out the variables in a different order.

Each variable in the READ statement must have a matching piece of information in the DATA statement. Try the following program:

Modify line 10

```
10 READ a,b,c,d
```

RUN

The result is the error message:

E Out of DATA, 10 : 1

So even though we did not ask the computer to print out the contents of variable 'd' in line 30, because the computer could not find a value to put into 'd', the program will not run.

String variables can also be used in READ and DATA statements.

TYPE:

```
10 READ a$,b$,c$
20 DATA "Happy", "New", "Year"
30 PRINT a$;b$;c$
40 STOP
```

RUN

The result is:

HappyNewYear

Exactly the same has happened as in the previous program; the only difference is that in the DATA statement the information has to be in quotes, as you would expect as we are now using string variables.

Notice that in all the programs the variables in the READ statement and the pieces of information (strings or numbers) in the DATA statement are separated by commas (.). This is necessary as it is part of BASIC language.

READ statements can contain both string and numeric variables provided that they match up with the correct types of information in the DATA statement.

TYPE:

```
10 READ a$,b$,c$,a
20 DATA "Battle", "of", "Hasting
s", 1066
30 PRINT a$; " "; b$; " "; c$; " ";
a
40 STOP
```

RUN

The result is: Battle of Hastings 1066

(Notice that spaces were added to line 30 to make the result of the PRINT statement more readable.)

There may be more than one READ statement in a program and more than one DATA statement.

TYPE:

```

10 REM BOOKS
20 READ x,a$,b$,c$,d$
30 READ e$,f$,y,g$
40 READ h$
50 DATA 2001,"Space","Odyssey"
,"Little","Women"
60 DATA "Great","Expectations"
,1984,"Animal","Farm"
70 PRINT "NAME OF BOOK" (dashes - NOT spaces
80 PRINT "-----" shifted J)
90 PRINT x;" ";a$;" ";b$
100 PRINT c$;" ";d$
110 PRINT e$;" ";f$
120 PRINT y
130 PRINT g$;" ";h$
140 STOP

```

RUN DO NOT REMOVE

As expected the result is a list of book titles but there are several points to notice about the program. There are three READ statements but only two DATA statements. This does not matter provided that the variables and information match up correctly and there are enough pieces of information for each variable. Variables g\$ (Animal) and h\$ (Farm) belong together but they have not been written in the same READ statement; the important point is that they correspond to the correct information in the DATA statement. This program can be rewritten in a different order to prove that, provided the variables and information match up, the result will be the same.

Alter lines 20 to 40 of the previous program.

```

20 READ g$,b$,y,e$,c$
30 READ a$,h$,f$,d$
40 READ x

```

RUN

Result: error message:

C Nonsense in BASIC, 20 : 1

This is because the variables no longer match up; for example, 'g\$' tries to take the value of '2001' which is obviously not acceptable.

Now alter lines 50 and 60

```

50 DATA "Animal","Odyssey",198
4,"Great","Little"
60 DATA "Space","Farm","Expect
ations","Women",2001

```

Leave the rest of the program as it is.

RUN

The list of books reappears on the screen, so provided that the variable fits the piece of information the actual order is unimportant.

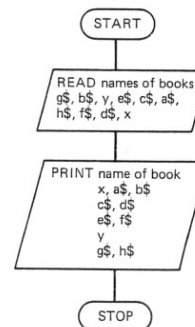
DATA statements often follow READ statements but they can be placed at the end of a program.

In a flowchart the READ statements are represented by input boxes



although the DATA statements are not represented but just assumed to be present. The flowchart that follows is for the program we have just used.

Flowchart



The following program shows how one READ statement may be used several times if included in a GOTO loop.

TYPE:

```

10 REM READ STATEMENT IN A GOT
O LOOP
20 LET c=0
30 INPUT "Enter a number betwe
en 0 and 12";n
40 READ a
50 DATA 3,4,7,11,1,0,6,8,9,10
60 DATA 2,0,12,7,5,6,3,2,1,12
70 DATA 1,5,3,7,9,3,11,2,4,-1
80 IF a<0 THEN GO TO 110
90 IF a=n THEN LET c=c+1
100 GO TO 40
110 PRINT "The number of times
";n;" appears is ";c
120 STOP

```

RUN the program several times entering different numbers.

This program inputs a number and calculates the number of times it occurs in a list of numbers in the DATA statement. There is only one variable in the READ statement so this changes its content every time the loop is completed. The first time round 'a' takes the value '3' which is then compared in line 90 with the number 'n' which was input in line 30. If 'a' = 'n' then one is added to the counter 'c'; if they are not equal then the remainder of line 90 is ignored. The computer is then sent back to the READ statement and as this is the second time round the loop 'a' takes the value '4' which is again compared to 'n'. This sequence of events is repeated until the last value, -1, is reached. When 'a' is less than 0 the computer jumps to line 110 where the result is printed out. The jump from line 80 to line 110 means that the GOTO statement in line 100 is missed out so the computer does not pass through the loop again and the program stops.

Question: What would happen to this program if the value -1 was not included as the last piece of information in the last DATA statement?
Remove this number from line 70 and see if you were right.

READ and DATA statements can also be used in FOR-NEXT loops.

```
TYPE:
10 FOR x=1 TO 7
20 READ y
30 DATA 5,10,15,20,25,30,35
40 PRINT y
50 NEXT x
60 STOP
RUN
```

This program passes through the FOR-NEXT 'x' loop seven times. Each time through the program the variable 'y' takes a new value beginning at 5 and working through the data in the correct order until 35 is reached. As the program is executed seven times there must be seven pieces of information, each being a number since 'y' is a numeric variable, in the DATA statement.

A similar loop can be used with string variables.

```
TYPE:
10 REM NAMES
20 FOR x=1 TO 5
30 READ y$
40 DATA "John", "Mary", "Anne", "
Paul", "Harry"
50 PRINT y$
60 NEXT x
70 STOP
RUN          DO NOT REMOVE
```

The program prints a list of five names. The DATA statement now contains strings so the variable used in the READ statement must also be a string.

It is possible to combine numeric and string variables in a loop but once again the data must match.

Alter lines 30, 40 and 50

```
30 READ y$,w
40 DATA "John",6,"Mary",7,"Anne",3,"Paul",10,"Harry",8
50 PRINT y$,w
```

Leave the rest of the program as it is.

RUN

```
The result is:  John    6
                Mary    7
                Anne    3
                Paul   10
                Harry    8
```

This program has a string variable 'y\$' followed by a numeric variable 'w' in the READ statement, therefore in the DATA statement the information is arranged accordingly with a string first followed by a number. As the computer will read 'y\$' first the first piece of data is a string 'John'. The computer will then read 'w' which takes the next item in the DATA list which is the number '6'. This sequence is repeated each time that the loop is executed.

Activities

Rewrite the following programs correcting the mistakes.

- 10 READ a\$, b, c\$
20 PRINT a\$, b, c\$
30 DATA 100, 200, 300
- 10 READ x, y, z\$
20 PRINT x, y, z\$
30 DATA cat, dog, fish
- 10 READ a\$, b\$, c
20 PRINT c; a\$; b\$
30 DATA 3, "Blind", "mice"
- 10 READ x\$ y b\$
20 PRINT y; b\$; x\$
30 DATA bottles, "10", green
- 10 FOR n = 1 TO 5
20 READ x
30 PRINT x
40 DATA 1, 2, 3, 4
50 NEXT n

6. 10 FOR n = 1 TO 3
20 READ x
30 PRINT x
40 DATA "man", "woman", "child"
50 NEXT n
7. 10 FOR n = 1 TO 3
20 READ a\$, b
30 PRINT a\$, b
40 DATA 3, "ships", 2, "cars", 1, "train"
50 NEXT n
8. What would be the contents of variables 'a\$', 'b\$' and 'c\$' at the end of this program?

10 READ a\$, b\$, c\$
20 DATA "Tom", "Dick", "Harry", "Jack", "Tim"
30 PRINT a\$, b\$, c\$
9. What would be the contents of variables 'a' and 'b\$' at the end of this program?

10 FOR n=1 TO 3
20 READ a, b\$
30 PRINT a, b\$
40 DATA 1, "France", 2, "Britain", 3, "Norway"
50 NEXT n
10. What would be the contents of variables 'x', 'y' and 'z' at the end of the third loop?

10 FOR n=1 TO 6
20 READ x, y, z
30 PRINT x, y, z
40 DATA 35, 4, 6, 8, 21, 42
50 DATA 10, 3, 17, 9, 12, 23
60 DATA 25, 15, 7, 33, 14, 11
70 NEXT n

Write programs for the following using READ and DATA statements.

11. Print out in a list the seven days of the week.
12. Print out a list of 10 numbers taken from anywhere between 35 and 64.
13. Input the sentence "The wicked witch cast many magic spells" using separate variables for each word. Print out three different sentences using some or all of these words.
14. Input and print a list of the following items of food and their prices (in pence):

bread 36, soup 19, jam 58, beans 17, eggs 42, rice 88

Calculate and print the total cost of all these items in pounds.

Unit 22: Nested loops



This unit also deals with FOR-NEXT loops, but for those cases where one loop is placed or nested inside another.

TYPE:

```

10 REM NESTED LOOPS
20 FOR n=1 TO 4
30 PRINT n;" OUTER LOOP"
40 FOR x=1 TO 3
50 PRINT x;" inner loop"
60 NEXT x
70 PRINT "-----"
80 NEXT n (dashes - NOT spaces)
90 STOP (shifted J)

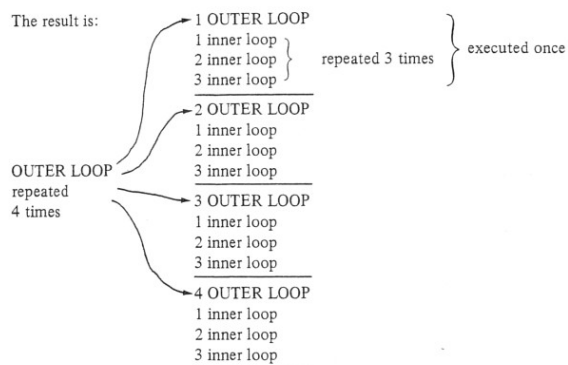
```

OUTER LOOP.
Repeated
4 times

Inner or nested loop.
Inside the outer loop.
Repeated 3 times every
time the computer
passes through the
outer loop once

RUN

The result is:



Notice that lines 30 and 70 are printed only 4 times because they are part of the outer loop.

An inner or nested loop must be contained entirely within the outer loop. You cannot write:

```
20 FOR n = 1 TO 4
30 FOR x = 1 TO 3
40 PRINT x
50 NEXT n
60 NEXT x
```

Loops overlap — not allowed

The following program fills the screen with numbers.

```
TYPE: 10 REM SCREEN OF NUMBERS
      20 FOR n=1 TO 22
      30 FOR x=1 TO 20
      40 PRINT x;
      50 NEXT x
      60 PRINT
      70 NEXT n
      80 STOP
RUN
```

Outer loop: lines 20 to 80. Inner or nested loop: lines 30 to 50.

The nested loop 'x', lines 30 to 50, prints out the values of x from 1 to 20 across the screen. The semi-colon (;) in line 40 makes sure that the numbers are printed next to each other. The outer loop 'n' is repeated 22 times so there are 22 lines of the numbers 1 to 20. Line 60 makes certain that each time the outer loop is executed a new line is started — see what happens if you remove line 60.

The next program shows how a whole set of multiplication tables can be printed out using nested loops.

TYPE:

```
10 REM MULTIPLICATION TABLES
20 FOR n=2 TO 12
30 PRINT n;" TIMES TABLE"
40 PRINT "-----" (dashes — shifted J) (not spaces)
50 FOR x=1 TO 12
60 PRINT x;"*";n;"=";x*n
70 NEXT x
80 NEXT n
90 STOP
```

Annotations: The inner loop 'x' is repeated 12 times for each 'n'. The outer loop 'n' is repeated 11 times.

RUN the program. You will have to 'scroll' several times.

The inner loop 'x' gives the number by which the table is to be multiplied; the first time round the loop it will give the 2 TIMES TABLE, the second time round the loop the 3 TIMES TABLE, and so on up to the 12 TIMES TABLE. The inner loop 'x' gives the number by which the table is to be multiplied; for example, the first value of x is 1 so the line printed out is $1 \times 2 = 2$, the second value of x is 2 so the next time round the loop $2 \times 2 = 4$ is printed. The inner loop finishes when x is 12 and the last line of the table $12 \times 2 = 24$ is printed. Control then goes back to the outer loop, the value of n is increased by 1 so the 3 TIMES TABLE is printed. The whole program finishes when the 12 TIMES TABLE has been printed.

We can now see how nested loops can be applied to a problem. A shop allows items to be purchased on special credit terms provided that 15 per cent of the price is paid as a deposit. The buyer may then pay off the remaining amount in monthly instalments over a 6, 9 or 12 month period. The program calculates the deposits required on items ranging from £5 to £50 for every £5 and then calculates the monthly instalments. All the information is printed out. The variables used are as follows:

p = price
d = deposit
n = number of months to pay off cost
m = amount of monthly instalment.

TYPE:

```

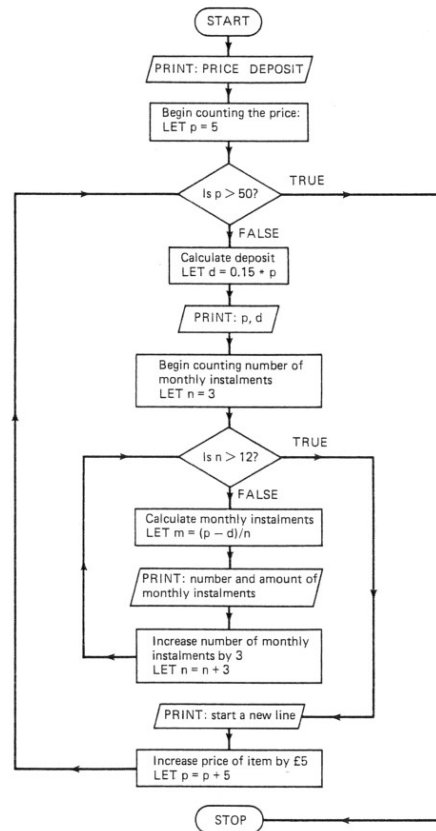
10 REM CREDIT TERMS
20 PRINT "PRICE", "DEPOSIT"
30 FOR p=5 TO 50 STEP 5
40 LET d=0.15*p (formula for calculating 15 per cent)
50 PRINT p,d
60 FOR n=6 TO 12 STEP 3 (3 monthly intervals)
70 LET m=(p-d)/n (formula for calculating monthly payments)
80 PRINT "The monthly instalme
nts over ";n;" months is ";m;" p
ounds"
90 NEXT n
100 PRINT
110 NEXT p
120 STOP

```

RUN

The outer loop, using the variable 'p' for price, calculates and prints the deposit needed for each item beginning at £5. Each time the computer passes through this loop £5 is added to the price (STEP 5) until £50 is reached. Line 40 is responsible for the actual calculation of the deposit at 15 per cent ($0.15 \times \text{price}$). In the inner loop, lines 60 to 90, the variable 'n' is used for the length of time over which the monthly instalments can be paid. The amount of time allowed varies between 3, 6 and 12 months. STEP 3 is used because the difference in time is 3 months. Line 70 calculates the amount of each monthly instalment – the price minus the deposit divided by the number of months, $(p - d)/n$, which is then printed out in line 80. The computer passes through this inner loop 3 times, and so calculates and prints three different amounts for 'm'.

Flowchart for the previous program



Activities

Use nested loops to write the following programs. Test them on the computer.

1. Print out 10 rows of 10 stars (*).
2. Input the names of 5 people and the amount of money each has saved in a bank account. Print out this information. Calculate the amount of interest each would have gained if the interest rates were 6 per cent, 8 per cent and 10 per cent. Print out the amount of interest and the total amount saved on each occasion. Use the following variables:
 s = sum saved $n\$$ = name of person
 r = interest rate
 i = interest gained = $\frac{\text{sum saved} \times \text{interest rate}}{100}$
 n = counter for 5 people
3. Print out five rectangles of stars (*). The length and width of the rectangles are to be entered while the program is being used. (This program needs three FOR-NEXT loops — each nested inside the other.)

Unit 23: Graphics using PRINT statements, colour and moving graphics

Until now we have controlled where the computer is to print a message or number by using the semi-colon, the comma and the TAB statement. We have also learned how to leave spaces in a program and how to start a new line by just using the word PRINT (for example, 50 PRINT). However, we can control the print position much more exactly by using PRINT AT. The word AT is on key I and is obtained by using SYMBOL SHIFT. If you look at diagram 23.1, you

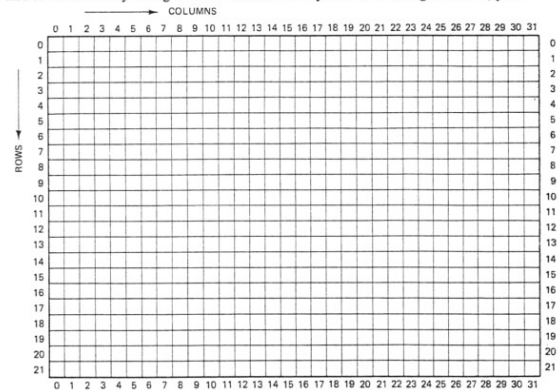


Diagram 23.1 To show the number of rows and columns on a Spectrum screen for use with PRINT AT statements.

will see that the screen is divided into twenty-two rows numbered 0 to 21 down the screen and into thirty-two columns numbered 0 to 31 across the screen. By using these numbers, rows first followed by columns, we can make the computer print anywhere on the screen. For example

10 PRINT AT 10,16;"*"

will print a star almost in the centre of the screen. Notice that the two numbers are separated by a comma and that a semi-colon separates the second number from the rest of the instruction. The program below prints stars at different positions on the screen.

TYPE:

```
10 REM STARS
20 INPUT "Enter a number betwe
en 0 and 21";a
30 INPUT "Enter another number
between 0 and 31";b
40 PRINT AT a,b;"*"
50 GO TO 20
```

RUN this program a few times, it will go on indefinitely because of the GOTO statement. To end the program enter STOP.

Activity

1. On a copy of diagram 23.1 draw a simple picture or pattern in stars and then produce it on the screen using the previous program. When your picture is complete enter STOP.

Now if we are drawing pictures, we do not want to be confined to using stars. So let us now look at the graphic characters. They are on the keys 1 to 8 as follows:



Only the parts in black will appear on the screen; so, for example, number 8 will produce a blank space. To use these graphic characters we must move into *graphics mode*; that is, change the flashing L cursor into a flashing G cursor. This is done by holding down CAPS SHIFT and then pressing key 9 which has GRAPHICS written above it. To move out of *graphics mode* just press key 9 again.

TYPE:

```
10 PRINT AT 10,9;" "
" " " " " " " " " "
```

RUN

DO NOT REMOVE

These are not the only characters that we can produce. If, while in *graphics mode*, we press CAPS SHIFT and any of the numbers 1 to 8 we get the character reversed. In other words the white becomes black and the black, white. For example



Add to the previous program:

```
20 PRINT AT 12,9;" "
" " " " " " " " " "
```

RUN

We can now use these characters to draw pictures. The program below produces a bus.

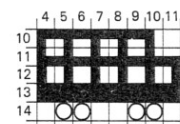
TYPE:

```
10 REM Drawing a bus
20 PRINT AT 10,4;" "
30 PRINT AT 11,4;" "
40 PRINT AT 12,4;" "
50 PRINT AT 13,4;" "
60 PRINT AT 14,5;"OO" } (capital letter O)
70 PRINT AT 14,9;"OO" }
```

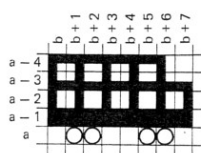
RUN

DO NOT REMOVE

You may think this is rather an odd bus but it gives you the idea of how to build up a picture using the graphic characters. It is possible to modify this program so that buses can be printed anywhere on the screen. Look at the following diagram of the bus:



The wheels of the bus are in row 14 (lines 60 and 70 of our program), so if we call row 14, row 'a', then row 13 will be 'a - 1' and row 12 will be 'a - 2' and so on. The left-hand edge of the bus is in column 4, so if we call this column 'b' then column 5 will be 'b + 1', column 6 will be 'b + 2' and so on. This is illustrated in the next diagram:



We can now change lines 20 to 70 in our bus program using variables 'a' and 'b'.

Alter the previous program:

```
10 REM Drawing a bus
20 PRINT AT a-4,b
30 PRINT AT a-3,b
40 PRINT AT a-2,b
50 PRINT AT a-1,b
60 PRINT AT a,b+1
70 PRINT AT a,b+5
```

The rest of these lines remain the same

Now all we need to do is to enter the numbers that 'a' and 'b' represent.

Add to the program:

```
12 INPUT "Enter a row-"; a
15 INPUT "Enter a column-"; b
80 GOTO 12
```

RUN the program so that nine buses fit on to the screen without touching each other or going off the edge of the screen. You will need to use diagram 23.1 to help you. If you go off the edge all sorts of odd things will happen. If you have done this successfully, deliberately type in numbers that will cause one bus to overlap another.

What happens?

DO NOT REMOVE

To colour the picture we use the commands:

INK (below key X) } both used in *extended mode*
 PAPER (below key C) } with *symbol shift*
 BORDER (keyword on key B) — entered in *keyword mode*

The colours that can be used are written above and operated by the number keys; for example, blue is number 1, red is number 2 and so on. We can tell the

computer to colour the border or the screen or both. The border is the part of the screen that is not printed on except for input statements.

Add to the previous program:

```
16 BORDER 1
```

Delete line 80.

RUN the program. The border will remain blue until another command is given.

Change line 16:

```
16 BORDER 2
```

RUN the program.

Try as many colours as you like but eventually decide on one colour that you would like to use.

DO NOT REMOVE

To colour the actual bus we need to change the ink. We will make the bus red and the wheels black.

Add to the program:

```
19 INK 2
55 INK 0
```

RUN

DO NOT REMOVE

Finally we can colour the background by changing the colour of the paper.

Add to the program:

```
17 PAPER 6
```

RUN the program.

Experiment with the INK and PAPER colours.

DO NOT REMOVE

It is also possible to make the bus move. We do this by using a FOR-NEXT loop.

Alter the program:

Delete line 10

Add:

```
5 FOR r = 0 TO 31
6 PRINT AT 12, r; " "
7 NEXT r
```

Prints out a road for the bus to travel along

Edit line 12:

```
12 LET a = 11
```

Delete line 15.

Add:

```
18 FOR b = 1 TO 23
80 NEXT b
```

Edit lines 20 to 70

```
10 REM Drawing a bus
20 PRINT AT a-4,b;"
30 PRINT AT a-3,b;"
40 PRINT AT a-2,b;"
50 PRINT AT a-1,b;"
60 PRINT AT a,b+1;"
70 PRINT AT a,b+5;"
```

Rest of line as before, but with the addition of a space before the first graphic character on each line

RUN DO NOT REMOVE

The bus will now rush across the screen from left to right. We can slow down the movement of the bus by using the command PAUSE on key M. For example

```
PAUSE 50 means wait for 1 second
PAUSE 150 means wait for 3 seconds (could also be written
PAUSE 50 * 3)
```

We do not want to slow the bus down too much so we will use only part of a second.

Add:

```
75 PAUSE 10
```

RUN DO NOT REMOVE

The bus now travels rather disjointedly across the screen. The FOR-NEXT loop is actually responsible for moving the bus. 'b' is given the value 1 at the beginning so the bus starts by having its left-hand edge in column 1; the next time round the loop 'b' is 2 so the bus moves along to the next column; the third time round the loop 'b' is 3 so again the bus is printed one column farther in, and so on until 'b' is 22. As the bus moves the previous left-hand edge of the bus is rubbed out by the space that was added to lines 20 to 70. If you want to alter the speed of the bus simply change line 75 - remember PAUSE 50 is one second.

Edit line 75:

```
75 PAUSE 0
```

RUN - the bus will not move until you press any key except the two shift keys.

REMOVE THE PROGRAM

Activity

2. (a) Write a program that prints a picture of one of the following: car, train,

ship, aeroplane. Use different colours for the border, paper and ink. Make your vehicle move across the screen.

- (b) Alter your program so that your vehicle crosses the screen six times. Each time it crosses it should be printed in a different colour (set the PAPER colour to white).

We will now look at moving objects in more detail. The next program moves a black square all the way round the edges of the screen. The only complicated part of the program is making sure that each square is rubbed out by a space as the next square is printed. The two variables used are 'a' for the rows and 'b' for the columns. Each section of the program begins with a PRINT statement, which tells you where the square will be moving. Lines 40, 130, 200 and 280 use a new command CLS (key V) which removes the writing off the screen before the square begins to move. The PAUSE statement before each of these lines ensures that the message stays for 2 seconds, just long enough to be read, before being cleared. (Do not confuse CLEAR (key X) with CLS. CLS stands for CLear Screen, and just wipes away what is printed on the screen, whereas CLEAR removes all the variables from the computer's memory!)

TYPE:

```
10 REM Moving square
20 PRINT "Square moves down th
e left-hand edge"
30 PAUSE 100
40 CLS
50 LET b=0
60 FOR a=0 TO 18
70 PRINT AT a,b;" "
80 PRINT AT a+1,b;"■"
90 PAUSE 10
100 NEXT a
110 PRINT "Square moves along t
he bottom of the screen from lef
t to right"
120 PAUSE 100
130 CLS
140 FOR b=0 TO 30
150 PRINT AT a,b;" "
160 PAUSE 10
170 NEXT b
180 PRINT "Square moves up righ
t-hand edge"
190 PAUSE 100
200 CLS
210 FOR a=19 TO 1 STEP -1
220 PRINT AT a-1,b;"■"
230 PRINT AT a,b;" "
240 PAUSE 10
250 NEXT a
260 PRINT "Square moves across
```

```

the top of screen from right to
left"
270 PAUSE 100
280 CLS
290 FOR b=30 TO 0 STEP -1
300 PRINT AT a,b;"■"
310 PAUSE 10
320 NEXT b
330 STOP

```

RUN

Moving the square downwards is done by adding 1 to the value of 'a' (the rows) every time the FOR-NEXT loop 'a' is executed. 'b' (the columns) remains constant as the column does not change, so 'b' is set to 0 and remains at 0. Line 70 prints a space which will follow the square all the way down the edge, so rubbing out the previous square. The first space is printed in 0, 0 ($a = 0$, $b = 0$) and therefore the first square is printed in 1, 0 ($a + 1 = 1$, $b = 0$). The second time round the loop 'a' is 1, so the space will be in 1, 0 ($a = 1$, $b = 0$) and the square in 2, 0 ($a + 1 = 2$, $b = 0$).

Moving the square across the screen from left to right has already been discussed in the bus program. Moving the square back up the screen is a matter of working backwards; 'a' now begins at 19, so 1 is taken off instead of being added on each time through the loop (STEP -1 in line 210). The space must continue to follow the square, so now it is placed below the square and is printed in 19, 30 ($a = 19$, $b = 30$) as the square has moved from column 0 to 30 in the previous section of the program) and the first square is printed in 18, 30 ($a - 1 = 18$, $b = 30$). Finally the square moves back across the screen from right to left by taking 1 from the value of 'b' which is now 30. The row does not change so 'a' remains constant. The space still follows on behind, so it is printed on the right-hand side of the square.

The next program moves the square diagonally from the top left-hand corner to near the bottom right-hand corner.

TYPE:

```

10 REM Square moves diagonally
20 FOR n=0 TO 20
30 PRINT AT n,n;" "
40 PRINT AT n+1,n+1;"■"
50 PAUSE 10
60 NEXT n
70 STOP

```

RUN

Now the rows and columns are represented by 'n' so both begin at 0. The space is first printed in 0, 0 (n , n) and the square in 1, 1 ($n + 1$, $n + 1$). See the diagram that follows.



The second time round the loop 'n' has the value 1 so the space is printed in 1, 1 (n , n) which rubs out the first square and the second square is printed in 2, 2 ($n + 1$, $n + 1$).

Activities

- Study the program below and decide where the square is moving to and from. Draw a diagram to show where the first space and square will be printed.

```

10 FOR n=0 TO 19
20 PRINT AT n,20-n;" "
30 PRINT AT n+1,19-n;"■"
40 PAUSE 10
50 NEXT n
60 STOP

```

- Write a program that moves a black square diagonally across the screen from near the bottom right-hand corner to the top left-hand corner. Begin at row 20 and column 20.

The following program enables you to draw a pattern by moving the square in any direction that you wish. It uses the number keys 5, 6, 7 and 8 which show the direction of movement by the arrows drawn above them. In order to do this the program uses the function INKEY\$ (above key N-extended mode). This function allows us to press a key on the keyboard and get an immediate result.

TYPE:

```

10 REM Drawing with a square
20 LET a=11 (a = rows, b = columns)
30 LET b=16
40 PRINT AT a,b;"■"
50 IF INKEY$="5" THEN LET b=b-
1
60 IF b<0 THEN LET b=0
70 IF INKEY$="6" THEN LET a=a+
1
80 IF a>21 THEN LET a=21
90 IF INKEY$="7" THEN LET a=a-
1
100 IF a<0 THEN LET a=0

```

```

110 IF INKEY$="8" THEN LET b=b+
1
120 IF b>31 THEN LET b=31
130 GO TO 40

```

RUN

The only way to stop this program is to press CAPS SHIFT and BREAK at the same time, followed by ENTER which returns the program listing to the screen.

This program draws a continuous thick black line because we have not rubbed out the square as we have moved along. The first square is printed in 11, 16 as these are the values given to 'a' and 'b' at the beginning of the program. The value of 'a' and 'b' then changes according to the key pressed. If '5' is pressed the square moves one position to the left and therefore the number of the column must decrease, so 'b' becomes 'b - 1'; 'a' remains the same since the square is in the same row. If '6' is pressed the square moves down one place so the number of the row increases by 1 and therefore 'a' becomes 'a + 1'. If '7' is pressed the square moves up one place so the number of the row decreases and 'a' becomes 'a - 1'. If '8' is pressed the square moves to the right so 'b' becomes 'b + 1'. Lines 80 and 100 just make certain that 'a' never gets smaller than 0 or larger than 21, and lines 60 and 120 make sure that 'b' never gets smaller than 0 or larger than 31. This keeps the square on the screen and stops an error message appearing on the screen.

Activity

- Alter the program so that as the square moves the previous one is rubbed out. You will need to put spaces all round your square so that whatever direction it is moved in there is a space behind it to rub out the square. Also, since you will have an 'invisible' area all round your square you will need to adjust the size of your screen otherwise your spaces will be off the screen and error messages will appear.

This idea of moving an object or character around the screen is a common feature of most computer games.

To conclude this unit we will look at three other commands:

FLASH (key V)	} All used in extended mode with SYMBOL SHIFT
BRIGHT (key B)	
INVERSE (key M)	

FLASH simply means, make the characters printed flash from black ink to white ink on a black background, just like the flashing cursors used by this computer. Number 1 starts the flashing and number 0 stops it.

TYPE:

```

10 FLASH 1
20 PRINT "These characters are
flashing"
30 FLASH 0
40 PRINT "These characters are
not flashing"
50 STOP

```

RUN

BRIGHT means make the characters appear on a brighter background, it is like adding highlights. It operates in the same way as FLASH.

TYPE:

```

10 BRIGHT 1
20 PRINT "I am bright"
30 BRIGHT 0
40 PRINT "I am dull"
50 STOP

```

RUN

INVERSE reverses the inks just like the FLASH command but there is no flashing from one to the other. Again the numbers 1 and 0 are used to turn the command on and off.

TYPE:

```

10 INVERSE 1
20 PRINT "My characters are wh
ite on a black background - I'm
inverted"
30 INVERSE 0
40 PRINT "My characters are no
rmal"
50 STOP

```

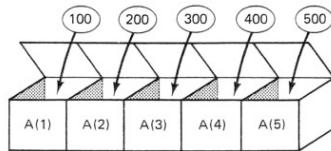
RUN

If FLASH, BRIGHT and INVERSE are not turned off at the end of the program they will continue to affect everything that follows (for example, the listing of the program).

Unit 24: Arrays—DIM statement

An array is a set or group of storage spaces all with the same name. The name of the array is just one letter of the alphabet (a lower or upper case letter). It is rather like a variable but instead of a single storage space, it is several, like a box with many compartments.

Each storage space in the array is called an element and is referred to by a number. For example, if we use an array named 'A' to store five numbers then the first number will be stored in the first element or storage space called A(1), the second number in A(2) and so on until the last number is stored in A(5). This would then be a numeric array with five elements — it will contain five numeric variables.



Numeric array with five elements

The array illustrated above shows that the number 100 is stored in A(1), the number 200 in A(2) and so on. The computer can call on any of these numbers by simply referring to the name and number of its storage space. For example, type PRINT A(1) and 100 appears on the screen.

TYPE:

```
20 LET A(1)=100
30 LET A(2)=200
40 LET A(3)=300
50 LET A(4)=400
60 LET A(5)=500
70 PRINT A(1),A(2),A(3),A(4),A(5)
```

RUN!

You will be surprised to find that the program does not work although each element in the array has been given a value. The error message

2 variable not found, 20 : 1

has appeared on the screen. This is because, before an array can be used, the computer has to be told the name and size of the array so that it knows how many storage spaces are needed. This is done by using the DIM statement which is an abbreviation for *dimension*. DIM is found on key D.

TYPE: 10 DIM A(5)

↑ ↑ size of array — number of elements
name of array

Now RUN

DO NOT REMOVE

The result is now what you would expect:

```
100
200
300
400
500
```

However, it probably has occurred to you that this result could have been achieved just as easily by using five different variables. For example

```
20 LET A = 100
30 LET B = 200
40 LET C = 300
```

etc.

```
70 PRINT A,, B,, C.....
```

So the next program uses the same array but only prints the number required by the person running the program. This could not be done if ordinary numeric variables had been used.

Leave lines 10 to 60 as they are.

Delete line 70.

TYPE:

```
70 INPUT "Enter a number between 1 and 5—"; n
80 PRINT A(n)
90 STOP
```

RUN the program several times using different numbers.

As you can see, the number 'n' entered in line 70 is then used with the name of the array 'A' in line 80. So if number 2 is entered then A(n) is really A(2) and 200 will be printed.

Another use of an array is to enter the numbers into the array while the program is running.

data into the five elements of the array. The message to be printed is decided by the number entered in line 60.

The next program called a "FRUIT MACHINE GAME" uses two string arrays and many of the techniques learned in the last few units.

TYPE:

```

10 REM Fruit Machine Game
20 DIM a$(10,6)
30 DIM b$(10,6)
40 FOR n=1 TO 10
50 READ a$(n)
60 NEXT n
70 FOR m=1 TO 10
80 READ b$(m)
90 NEXT m
100 DATA "orange","lemon","orange","orange",
"cherry","lemon","bell","orange","cherry",
"lemon","bell","orange","cherry","cherry",
"bell","lemon","cherry","orange","bell",
"lemon","orange","lemon","orange",
"bell","lemon"
110 LET s=0
120 PRINT "FRUIT MACHINE GAME"
130 PRINT "RULES-you cannot enter the same pair of numbers twice"
140 PRINT
150 FOR a=1 TO 5
160 INPUT "Enter a number between 1 and 10 ";n
170 PRINT a$(n);" ";
180 INPUT "Enter another number between 1 and 10 ";m
190 PRINT b$(m)
200 IF a$(n)<>b$(m) THEN PRINT "You lose - no points scored": GOTO 230
210 IF a$(n)=b$(m) THEN PRINT "you win - 5 points scored"
220 LET s=s+5
230 PRINT
240 NEXT a
250 PRINT
260 PRINT
270 PRINT "You scored ";s;" points"
280 STOP

```

RUN

Lines 20 to 100 put the information into the elements of the two arrays called 'a\$' and 'b\$'. Each array has storage spaces for 10 strings with a maximum length of 6 characters. Two FOR-NEXT loops 'n' and 'm', lines 40 to 60 and

70 to 90, feed the information in using READ and DATA statements. Notice that all the strings are contained in one DATA statement, this is possible because the first loop 'n' takes the first ten items out of the DATA statement and puts them into a\$(1), a\$(2) etc. and then the second loop 'm' takes the next ten items and stores them in b\$(1), b\$(2) etc.

The counter 's', to keep the score, is set up in line 110. The title and the rules of the game are printed by lines 120 and 130. Another FOR-NEXT loop using the variable 'a' (in no way connected with array 'a\$') is then set up to count to 5, the number of turns the player is allowed to have. In lines 160 to 190 numbers are entered which refer to two different storage spaces, one in each of the two arrays. The contents of these storage spaces are printed out, 'a\$(n)' and 'b\$(m)'. Lines 200 and 210 compare the contents of 'a\$(n)' and 'b\$(m)', if they are not the same then the player loses and no points are scored and the computer jumps to line 230. If the contents are the same then the player wins and scores 5 points which are added to the score 's' in line 220. The computer then continues through the program by repeating the loop until it has been executed five times. The program finishes by printing out the final score.

Another important thing to note about string arrays is that, as the maximum length of any of the strings to be stored has to be given, then any string put into one of the elements of the array will automatically be padded out with spaces if it is less than the maximum length. See the example that follows.

If we set up a string array: DIM a\$ (6, 7) then the computer's memory will look like this:

Array name and number of element	Length of string						
	1	2	3	4	5	6	7
a\$(1)							
a\$(2)							
a\$(3)							
a\$(4)							
a\$(5)							
a\$(6)							

This array is waiting to be filled with 6 strings, each of length 7 characters. If the following strings are entered

dog, giraffe, camel, fish, hamster, kitten

the array will look like this (a dash represents a space)

Array name and number of element	Length of string						
	1	2	3	4	5	6	7
a\$(1)	d	o	g	—	—	—	—
a\$(2)	g	i	r	a	f	f	e
a\$(3)	c	a	m	e	l	—	—
a\$(4)	f	i	s	h	—	—	—
a\$(5)	h	a	m	s	t	e	r
a\$(6)	k	i	t	t	e	n	—

So what appears to be nothing is really a space, which is a character. Hence all of these strings are the same length. It is necessary to remember this because if you were writing a program and wished to enter a word and then compare it with words in an array, they have to be the same length. For example, imagine we have set up the array as drawn in the last diagram and the program goes as follows:

```
50 INPUT "Enter the name of an animal-"; b$
60 IF b$ = a$(1) THEN PRINT "Woof, woof"
```

Even if the word 'dog' is entered the computer will ignore the instruction to print "Woof, woof" because 'dog' contained in 'b\$' is only three characters long whereas the 'dog' in a\$(1) is seven characters long, therefore they are not the same string. In the Fruit Machine Game program, lines 200 and 210, the elements of the two different arrays were compared but as both arrays had the same maximum length (6) this was possible. If the second array had been called 'b\$(10, 7)' this program would not have worked. We can prove this point in the following program.

TYPE:

```
10 DIM a$(3,5)
20 LET b$="hi"
30 LET a$(1)=b$
40 IF a$(1)=b$ THEN PRINT "O K"
```

RUN

DO NOT REMOVE

Nothing happens except that the message O O K, 40 : 1 appears. This is because it is not possible for a\$(1) to contain the word 'hi' which is only two characters, since the 'a\$' array states that all the elements must have five characters.

Alter line 20

```
20 LET b$ = "hi _ _ _"
```

RUN

DO NOT REMOVE

The computer now prints 'O K' because 'b\$' has been padded out with three spaces which will now fit into a\$(1). However, this is not a good way to overcome the problem; it is much better to let b\$ have an array of its own, also with a maximum length of five characters.

Modify the program.

```
15 DIM b$(1,5)
20 LET b$(1)="hi"
30 LET a$(1)=b$(1)
40 IF a$(1)=b$(1) THEN PRINT "
O K"
```

RUN the program

'b\$(1)' is now automatically five characters long as the word 'hi' will be padded out with three spaces and so 'O K' will be printed. Let us see if this will work in a more complicated program.

TYPE:

```
10 REM COMPUTERS
20 DIM a$(12,9)
30 DIM x$(1,9)
40 PRINT "COMPUTERS"
50 INPUT "Enter the name of a
computer ";x$(1)
60 FOR n=1 TO 12
70 READ a$(n)
80 IF x$(1)=a$(n) THEN PRINT "
Yes,there is a computer called t
he ";x$(1): GO TO 120
90 NEXT n
100 PRINT "No,there is not a co
mputer called the ";x$(1)
110 DATA "Spectrum","Oric","B.B
.C","Apple","Electron","Atari","
ZX81","Research","Acorn","Commod
ore","I.B.M","Dragon"
120 STOP
```

RUN

The name entered in line 50, stored in the array x\$(1,9), is compared with the contents of array 'a\$(12,9)'. This is possible because both arrays contain strings that are nine characters long. As array 'a\$' has twelve elements the content of x\$(1) is compared with each element in turn, beginning with a\$(1). The GOTO statement in line 80 causes the computer to jump out of the FOR-NEXT loop 'n' which is something we have not tried before. Can you think why this was necessary?

A program can contain both numeric and string arrays but these must be separate arrays. It is not possible to have a mixture of numeric and string variables in the same array.

TYPE:

```

10 REM MOUNTAINS
20 DIM a$(6,11)
30 DIM x(6)
40 FOR n=1 TO 6
50 READ a$(n)
60 DATA "Everest","Kilimanjaro",
"Table","Blanc","Matterhorn","
Ben Nevis"
70 NEXT n
80 FOR m=1 TO 6
90 READ x(m)
100 DATA 8848,5895,1087,4807,44
78,1347
110 NEXT m
120 PRINT "MOUNTAIN","HEIGHT IN
METRES"
130 PRINT
140 FOR y=1 TO 6
150 PRINT a$(y),x(y)
160 NEXT y
170 STOP

```

RUN

DO NOT REMOVE

Lines 20 and 30 set up the two arrays, a\$(6,11) for strings and x(6) for numbers. The first FOR-NEXT loop 'n' enters the names of the mountains into six elements of the string array. The second FOR-NEXT loop 'm' enters the heights of the mountains into six elements of the numeric array. Notice that there is no connection between the two arrays. Line 120 prints out the headings for two columns. The third FOR-NEXT loop 'y' prints out the information in two columns.

Why is the variable 'y' used with a\$ and x in this last loop?

Activities

1. (a) Add another string array to the previous program to store the names of the countries in which the mountains are situated. Use the information given below.

Mountain	Country
Everest	Nepal-Tibet
Kilimanjaro	Tanzania
Table	South Africa
Blanc	France
Matterhorn	Switzerland
Ben Nevis	Scotland

- (b) Add another section to the program, in order to print this information in two columns. Each column must have a heading.
RUN your program.
2. Write a program that inputs eight numbers and then prints them out in reverse order.
3. Write a program that inputs twelve numbers and then prints out every other number.
4. (a) Write a program that stores the names of the days of the week and then prints the name of the day according to the number entered by the user.
(b) Modify the program so that if a number less than 1 is entered then "number too low" is printed or if a number above 7 is entered then "number too high" is printed and the computer goes back to the line where another number can be entered.
(c) Modify the program so that it runs four times before stopping.
5. Write a program that inputs a word which is checked to see if it is the name of a month. If it is then the program should print out the number of that month (for example, May is month 5). If the word is not a month then the message "Not a month" should be printed. (You need two arrays, one to store the names of the months and the other to store the word that is entered.)
6. Write a program that stores and prints the following information.

HOW ELECTRICITY IS PRODUCED IN THE U.K.

Source of electricity	Percentage of total produced
Coal	79
Oil	8
Nuclear	11
Water	2

Test your programs on the computer.

All the arrays used so far are known as *one-dimensional arrays*. This is because they contain only one set or list of elements. However, it is possible to have *two-dimensional arrays*. For example, DIM a(6, 3) means that the array called 'a' has eighteen elements, six rows of three columns. The names of the elements in the array are:

	Column 1	Column 2	Column 3
Row 1	a(1, 1)	a(1, 2)	a(1, 3)
Row 2	a(2, 1)	a(2, 2)	a(2, 3)
Row 3	a(3, 1)	a(3, 2)	a(3, 3)
Row 4	a(4, 1)	a(4, 2)	a(4, 3)
Row 5	a(5, 1)	a(5, 2)	a(5, 3)
Row 6	a(6, 1)	a(6, 2)	a(6, 3)

Notice that the first figure in the brackets refers to the row and the second figure to the column:

```

DIM a (6, 3)
      ↑   ↑   ↑
      Name Number Number
      of   of   of
      numeric row column
      array
          6 × 3 = an 18
          element array

```

The following program uses a two-dimensional numeric array.

```

TYPE:
10 REM TRAIN TIMES
20 DIM a(6,3)
30 FOR n=1 TO 6
40 FOR m=1 TO 3
50 READ a(n,m)
60 DATA 7.01,7.48,8.04
70 DATA 7.14,8.03,8.19
80 DATA 7.32,8.19,8.33
90 DATA 7.45,8.32,8.49
100 DATA 8.01,8.54,9.07
110 DATA 8.13,9.03,9.19
120 NEXT m
130 NEXT n
140 PRINT "BRIGHTON   E.CROYDON
VICTORIA"
150 PRINT "DEPART     ARRIVE
ARRIVE"
160 FOR x=1 TO 6
170 FOR y=1 TO 3
180 PRINT a(x,y); " ";
190 NEXT y
200 PRINT
210 NEXT x
220 STOP
RUN

```

Points to note:

- Line 20 – sets up two-dimensional numeric array.
- Line 30 – counts the number of rows – 1 to 6.
- Line 40 – counts the number of columns – 1 to 3.
- Line 50 – reads the information into the elements of the array

```

a (n, m)
name, row, column

```

Lines 60–110 – supply the train times. First time round the loop the first element is a(1, 1), and this takes the value 7.01; the inner loop is then repeated so the second element is a(1, 2), and this takes

the value 7.48; the inner loop is repeated again so the third element is a(1, 3) which takes the value 8.04; the first row of the array is now complete. Control then returns to the outer loop so 'n' becomes 2 and 'm' returns to 1 so the next element is a(2, 1), and this takes the value 7.14. This pattern continues until the outer loop has run six times and then all the elements will contain numbers.

Lines 140 and 150 – print out the headings in three columns.

Lines 160 to 210 – two nested loops print out the information in columns and rows. Loop 'x' counts the rows and loop 'y' the columns hence the name of each element is now a(x, y).

The next program stores the names of six pupils and their examination marks for English, French and Maths. It prints this information in a table and also calculates and prints the pupil's total marks and average marks.

```

TYPE:
10 REM CLASS MARKS
20 DIM a(6,3)
30 DIM n$(6,7)
40 FOR n=1 TO 6
50 FOR m=1 TO 3
60 READ a(n,m)
70 DATA 42,52,61,12,23,34,67,7
2,63,78,82,91,53,62,58,73,69,85
80 NEXT m
90 NEXT n
100 FOR z=1 TO 6
110 READ n$(z)
120 DATA "Smith","Allen","Brown
ey","Davies","Cotton","Baker"
130 NEXT z
140 PRINT "NAME   ENGLISH   FRENC
H   MATHS"
150 FOR x=1 TO 6
160 PRINT n$(x); " ";
170 LET t=0
180 LET v=0
190 FOR y=1 TO 3
200 PRINT a(x,y); " ";
210 LET t=t+a(x,y)
220 NEXT y
230 LET v=t/3
240 PRINT
250 PRINT "TOTAL MARKS ";t
260 PRINT "AVERAGE MARK ";v
270 PRINT
280 NEXT x
290 STOP
RUN the program.
Study the result and the program.
Write a description of how the program works.

```

We can now write a program to play the game of battleships. Battleships consists of a board 8 squares by 8 squares on which are secretly placed 4 submarines, 2 destroyers, 2 cruisers and 1 battleship, as drawn on the diagram below. Notice the shapes of the ships, it is important to remember these when playing the game.

	1	2	3	4	5	6	7	8
1							D	D
2		C	C					
3			C			B		
4	S					B		S
5			D	D				
6		S				B		
7				C				S
8			C	C				

S = submarine
D = destroyer
C = cruiser
B = battleship

The position of these ships has to be stored in a numeric array so it is necessary to change the letters to numbers. Therefore submarines will be 1, destroyers 2, cruisers 3 and the battleship 4. Any square without a ship is a 0. So the board now looks like this.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	2	2
2	0	3	3	0	0	0	0	0
3	0	0	3	0	0	4	0	0
4	1	0	0	0	0	4	0	1
5	0	0	2	2	0	4	0	0
6	0	1	0	0	0	4	0	0
7	0	0	0	3	0	0	0	1
8	0	0	3	3	0	0	0	0

0 = no ship
1 = submarine
2 = destroyer
3 = cruiser
4 = battleship

This information can now be fed into a two-dimensional numeric array with 64 elements – 8 columns and 8 rows, we will call this 'a(8,8)'. This is done by using nested loops, the outer loop 'p' counts the rows and the inner loop 'q' counts the columns. The inner loop will run eight times every time that the outer loop is executed once. While this information is being read into the array it is necessary to count the number of squares that are 0; this gives us the number of squares that contain ships which will be needed later to see when all the ships are sunk and the game ended. Variable 'n' is used for this and then each element of the array, 'a(q,p)', will be compared with 0; if it does not equal 0 then 1 is added to the counter 'n'. We can now write the first part of the game.

TYPE:

```
10 REM BATTLESHPIS
20 DIM a(8,8)
30 LET n=0
40 FOR p=1 TO 8
50 FOR q=1 TO 8
60 READ a(p,q)
70 IF a(p,q)<>0 THEN LET n=n+1
80 NEXT q
90 NEXT p
100 DATA 0,0,0,0,0,0,2,2
110 DATA 0,3,3,0,0,0,0,0
120 DATA 0,0,3,0,0,4,0,0
130 DATA 1,0,0,0,0,4,0,1
140 DATA 0,0,2,2,0,4,0,0
150 DATA 0,1,0,0,0,4,0,0
160 DATA 0,0,0,3,0,0,0,1
170 DATA 0,0,3,3,0,0,0,0
```

The board for the battle is now stored in the array 'a(8,8)'; for example, the number 0 is stored in a(3,1) and the number 4 in a(3,6). The counter 'n' has kept a record of the number of squares that have ships on them. Now the computer needs to know what 1, 2, 3 and 4 represent because if you hit a ship you need to know what you have hit and to give a number as an answer is not good enough. So we will use a one-dimensional string array, s\$(4,10), to store this information. The array has four elements consisting of ten characters each.

TYPE:

```
180 DIM s$(4,10)
190 FOR b=1 TO 4
200 READ s$(b)
210 DATA "submarine","destroyer",
"","cruiser","battleship"
220 NEXT b
```

The computer now knows that s\$(1) is a submarine, s\$(2) is a destroyer, s\$(3) is a cruiser and s\$(4) a battleship.

To play Battleships it is necessary to enter two numbers, these will be called 'x' and 'y' in the program. The first number 'x' refers to the columns and the second number 'y' refers to the rows. Numbers entered must be between 1 and 8, anything smaller than 1 or larger than 8 is not allowed. The computer will check this and print an appropriate message if necessary. If the numbers 5 and 6 are entered then 'a(5,6)' is a 0 (see the next diagram), so the computer reports that you have missed and then we need to return for another turn. If on the other hand you enter 2 and 6 you will be told that you have hit a submarine (see the next diagram). (When playing the game it is as well to keep a blank copy of the board so that you can record your hits and misses.)

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	2	2
2	0	3	3	0	0	0	0	0
3	0	0	3	0	0	4	0	0
4	1	0	0	0	0	4	0	1
5	0	0	2	2	0	4	0	0
6	0	X	0	0	X	4	0	0
7	0	0	0	3	0	0	0	1
8	0	0	3	3	0	0	0	0

a(5, 6) = 0 therefore a
'miss' is recorded
a(2, 6) = 1 therefore a
'hit' is recorded

Now let us write this part of the program.

TYPE:

```
230 INPUT "Fire at a ship by en
entering two numbers between 1 and
8. Press ENTER after each number
";x,y
240 IF x<1 OR x>8 OR y<1 OR y>8
THEN PRINT "Numbers not allowed
": GO TO 230
250 IF a(x,y)=0 THEN PRINT "You
missed": GO TO 230
260 PRINT "You hit a ";s$(a(x,y
))
```

If you hit a ship all that now remains to be done is to take 1 off the counter 'n' which is storing the number of squares with ships on them. The counter must then be compared with 0 to see if there are any more ships to sink. Once 'n' equals 0 then the game must be over and all we need is a finishing message.

TYPE:

```
270 LET n=n-1
280 IF n>0 THEN GO TO 230
290 PRINT
300 PRINT "Well done, all ships
sunk"
310 STOP
```

Now play the game; hopefully you will not remember too well where the ships are. Do not look back at any of the diagrams.

Finally we need to look at a two-dimensional string array. For example

```
DIM a$( 4, 2, 6)
↑      ↑      ↑
Name   ↑      ↑
of      Number of
array  columns
        Number of
        rows      Maximum
                  length of
                  strings
```

This is used like a two-dimensional numeric array with nested loops to feed in the information.

TYPE:

```
10 REM NAMES
20 DIM a$(4,2,6)
30 PRINT "GIRL", "BOY"
40 FOR n=1 TO 4
50 FOR m=1 TO 2
60 READ a$(n,m)
70 DATA "Anne", "Paul", "Jane", "
Robert", "Sarah", "James", "Susan",
"Mark"
80 PRINT a$(n,m),
90 NEXT m
100 PRINT
110 NEXT n
120 PRINT
130 PRINT "Examples of what som
e elements contain"
140 PRINT "a$(1,1)=";a$(1,1)
150 PRINT "a$(3,2)=";a$(3,2)
160 PRINT "a$(4,1)=";a$(4,1)
170 STOP
```

RUN

As there are two columns in this array the FOR-NEXT loop 'm' operates twice, picking up the first two names, every time the loop 'n' operates once. Loop 'n' is performed four times as there are four rows in the array. The names are printed (line 80) as the loops operate but just to prove that the elements of the array do retain the names throughout the program lines 140 to 160 print three examples.

Activities

- Using the program just used what names are contained in the elements: a\$(1, 2), a\$(3, 1) and a\$(4, 2)?
- Using a two-dimensional string array print out in two columns, with headings, the information given below.

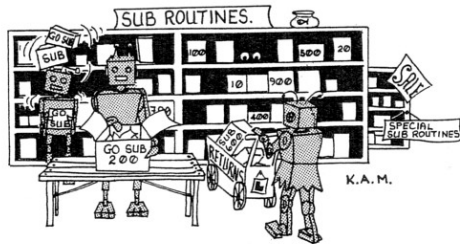
Name	Sport
J. Connors	Tennis
S. Cram	Athletics
S. Davis	Snooker
L. Piggott	Horse racing
F. Bruno	Boxing

- Using a two-dimensional numeric array print in three columns, with headings, the information below.

Year	Share of world trade (per cent)	
	Developed countries	Developing countries
1950	68	32
1960	79	21
1970	82	18

Test your programs on the computer.

Unit 25: Subroutines

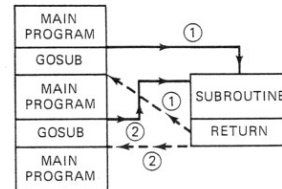


As programs tend to get longer there may be several lines of the program that are needed more than once. It is not necessary to keep typing in these lines as they can be put into a *subroutine*, or *subprogram* as it is sometimes called. A subroutine, therefore, consists of several program lines making up one section of the main program which may be called upon more than once while the program is running. To use or call a subroutine the statement GOSUB, on key H, is used. For example

GOSUB 200

This means go to the subroutine that starts on line 200. When the lines in the subroutine have been executed they must always finish with the statement, RETURN, on key Y. This sends the computer back to the place where it has just come from. For example, if GOSUB 200 was line 50 in a program then the computer will RETURN to this line and continue with the next line of the program. Look at the following illustration:

132



Therefore, when the computer is sent to another part of the program by a GOSUB statement it remembers where it has come from and returns to that place. Subroutines are usually placed towards the end of the main program. There is no special word to indicate the beginning of a subroutine, so it is good practice to include a REM statement at that point; this has been done in the following program.

TYPE:

```

10 REM Use of subroutines
20 INPUT "Enter your name ";n$
30 PRINT n$
40 GO SUB 200
50 INPUT "Enter first line of
address ";a$
60 INPUT "Enter second line of
address ";b$
70 INPUT "Enter third line of
address ";c$
80 PRINT a$,,b$,,c$
90 GO SUB 200
100 INPUT "Enter age in years(n
umber only) ";a
110 PRINT "Age is ";a;" years"
120 GO SUB 200
130 INPUT "Enter date of birth
";d$
140 PRINT "Date of birth is ";d$
$
150 GO SUB 200
160 STOP
200 REM Subroutine to draw a li
ne
210 FOR j=1 TO 32
220 PRINT "- ";      (a dash - not a space)
230 NEXT j
240 PRINT
250 RETURN

```

RUN, entering in the information as necessary.

This program prints out the name, address, age and date of birth of a person but separates each piece of information with a dotted line. As the line is required a number of times it has been placed in a subroutine beginning at line 200. Line 40 calls the subroutine for the first time, so when it has been executed the computer returns to this point and continues to line 50. Line 90 calls the subroutine for the second time and so this time the computer returns to line 90 and continues to line 100. The subroutine is called twice more, by lines 120 and 150. The STOP statement, in line 160, is very important because if it was not there the computer would automatically fall into the subroutine again; on reaching the RETURN statement it would be confused, as it would not know where to return to since it was not sent by a GOSUB statement.

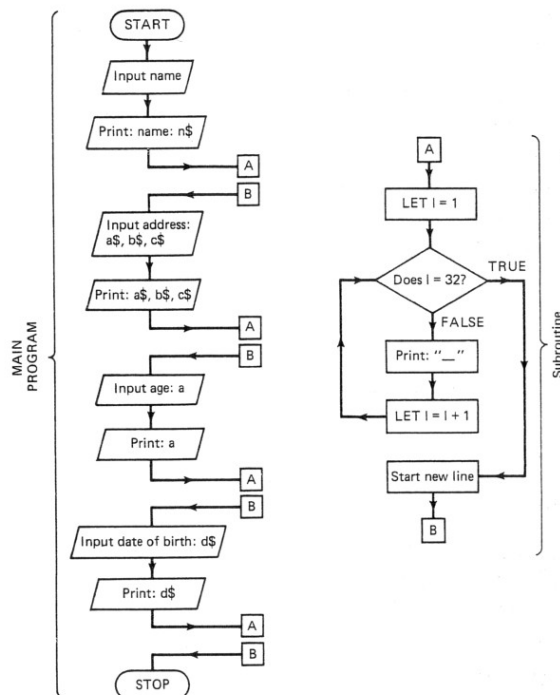
Delete line 160.

RUN

As a result of this an extra line is drawn as the computer falls into the subroutine again but then the error message

7 RETURN without GOSUB, 250 : 1

appears. The flowchart of this program shows that the subroutine is drawn as a separate section.



The next program is about a machine which is operated by an automatic time switch; it goes on and off every 5 seconds. The program gives a message that tells whether the machine is 'on' or 'off'. The 5-second intervals are controlled by a FOR-NEXT loop which is written in a subroutine. The main program is executed five times; this is counted by the variable 'c'.

TYPE:

```

10 REM Machine on or off
20 LET c=0
30 GO SUB 100
40 PRINT "MACHINE ON"
50 GO SUB 100
60 PRINT "MACHINE OFF"
70 LET c=c+1
80 IF c=5 THEN GO TO 140
90 GO TO 30
100 REM Subroutine for timing i
    ntervals
110 FOR x=1 TO 900
120 NEXT x
130 RETURN
140 STOP

```

RUN

Activity

1. Draw a flowchart for the previous program.

Sometimes a subroutine is called only if certain conditions are true and ignored if the conditions are not true. The next program is an example of this.

TYPE:

```

10 REM Subroutine used if n is
4 or 7
20 INPUT "Enter a number betwe
en 1 and 10 ";n
30 IF n=4 OR n=7 THEN GO SUB 1
00
40 LET n=n+2
50 PRINT n
60 STOP
100 REM Subroutine
110 LET n=n*n
120 RETURN

```

Before you RUN the program try the following question.

2. What is printed in line 50 when the following numbers are entered: (a) 4, (b) 2, (c) 7. Check your answers by running the program.

Activities

3. What is the value of x after this program has been run if: (a) n = 5, (b) n = 2.

```

10 INPUT "Enter a number ";n
20 IF n<5 THEN GO TO 40
30 GO SUB 70
40 LET x=n*n
50 PRINT x
60 STOP
70 REM subroutine
80 LET n=n+z
90 RETURN

```

Draw a flowchart for this program.

4. Write a program that asks the users four questions about their home. The question and the answer should be printed on the screen and separated from the next by a row of stars, which are printed out by a subroutine.
5. Enter a number between 1 and 100, 'n'. If 'n' is less than 50 send the computer to a subroutine where 25 is added to 'n'. On returning to the main program the numbers from 0 to 'n' are printed out in steps of five.
6. Write a program for a number guessing game. Enter a number between 1 and 100 without the player looking. The player should then enter a guess and an appropriate message should be printed out if they are right. The program should return to the beginning for another turn. If the guess is too high then the computer should go to a subroutine that prints this fact and asks the player to have another guess. If the guess is too low then the computer should go to another subroutine that prints this and asks the player to have another guess. The main program should form an infinite loop.
7. A robot has two sets of instructions for opening a wine bottle: one for a bottle with a cork and another for a bottle with a screw top. Enter the type of bottle the robot is to open and then go to the right set of instructions that are to be followed. The instructions should be printed on the screen. (Two subroutines will be required, one for each set of instructions.) After the bottle has been opened the computer should return to the main program and print these messages:
 - "Bottle opened"
 - "Shall I pour a glass?"
 - "Cheers!!"

If the wrong type of bottle top is entered then the computer should print an appropriate message and send the user back to the beginning to enter another bottle top.

8. It takes 0.15 litres of paint to paint one square metre of wall. Write a program that:
 - (a) inputs the number of walls to be painted
 - (b) inputs the length and height of each wall
 - (c) calculates the area of each wall
 - (d) calculates the total area of all the walls to be painted
 - (e) calculates the amount of paint required - total area multiplied by 0.15

(f) prints out the amount of paint required.
Unfortunately all the measurements for the walls are entered in feet so these have to be converted to metres; this is done in a subroutine. The mathematical formula required is:

number of metres (m) = number of feet (f) divided by 3.208

In BASIC this is

$m = f/3.208$

The following variables should be used:

r = litres of paint required to paint one square metre
n = number of walls to be painted
f = length of wall in feet
f = height of wall in feet } also used in the subroutine
a = area of each wall ($l * h$)
t = total area of all the walls ($t + a$)
m = length of wall in metres } to be used in the subroutine
m = height of wall in metres }
l = length of wall in metres } to be used in the main program
h = height of wall in metres }
p = quantity of paint required ($t * r$)
w = variable to count the number of walls.

The final example in this unit shows how even the RETURN statement can be subject to a condition. In line 320 the computer RETURNS to the main program only if c(1) = t(x) . If it does not the computer ignores the RETURN statement and carries on through more of the subroutine until it reaches another RETURN statement in line 350.

TYPE:

```
10 REM Distances between cities
S
20 DIM t$(7,10)
30 DIM d(7,7)
40 DIM c$(1,10)
50 FOR n=1 TO 7
60 READ t$(n)
70 DATA "London","Taunton","Plymouth",
"Nottingham","Liverpool",
"Leeds","Hull"
80 NEXT n
90 FOR m=1 TO 7
100 FOR s=1 TO 7
110 READ d(m,s)
```

```
120 DATA 0,144,211,122,197,190,
168,144,0,74,180,203,237,247,211,
74,0,254,278,312,321,122,180,25
4,0,97,67,73,197,203,278,97,0,73
,128,190,237,312,67,73,0,55,168,
247,321,73,128,55,0
130 NEXT s
140 NEXT m

150 INPUT "Enter city you are travelling to ";c$(1)
160 GO SUB 300
170 IF x=0 THEN PRINT "No information about ";c$(1): GO TO 150
180 LET y=x
190 INPUT "Enter city you are travelling from ";c$(1)
200 GO SUB 300
210 IF x=0 THEN PRINT "No information about ";c$(1): GO TO 190
220 PRINT "The distance between ";t$(y);" and ";t$(x);" is ";d(y,x);" miles"
230 STOP
300 REM Subroutine
310 FOR x=1 TO 7
320 IF c$(1)=t$(x) THEN RETURN
330 NEXT x
340 LET x=0
350 RETURN
```

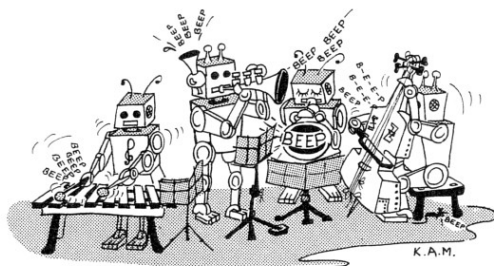
RUN the program several times trying different cities.

All the techniques used in this program except for the GOSUB and RETURN statements have already been met in previous lessons so study the program carefully and:

Activity

9. Write an explanation of how the program above works.

Unit 26: Making music



As you may already know, the Spectrum has a loudspeaker built into it. This can be made to play a single note by using the BEEP statement below key Z (extended mode and symbol shift). The program that follows gives you the opportunity to hear the loudspeaker play.

TYPE:

```
10 FOR n=50 TO -30 STEP -1
20 BEEP 0.25,n
30 NEXT n
```

RUN DO NOT REMOVE

Look at line 20. BEEP, the command to play a note, is followed by two numbers. The first number is the length or *duration* of the note; the number 1 represents a note one second long so in this program each note is a quarter of a second long (0.25). The second number is the actual note or *pitch*; this begins here at 50, which is very high, and finishes at -30, which is very low. So the larger the number the higher the note. The whole program plays 80 notes, each 0.25 second long, going down the scale one note at a time. It would be just as easy to play these notes going up the scale.

Change line 10

```
10 FOR n = -30 TO 50
```

RUN DO NOT REMOVE

It would also be possible to make the computer play alternate notes by altering the size of the STEP.

Change line 10

```
10 FOR n = -30 TO 50 STEP 2
```

RUN. Experiment with this line by altering the size of the STEP.

So far all we have produced is a whole series of notes but we do not know what notes we are playing. For this we need to look at the piano keys to see which note has which number. There is not enough space to show all the keys so the diagram just concentrates on the notes around middle C, the note which is almost in the centre of the keyboard.

				D \flat C \sharp -11	E \natural D \sharp -9		G \flat F \sharp -6	A \flat G \sharp -4	B \flat A \sharp -2		D \flat C \sharp 1	E \natural D \sharp 3		G \flat F \sharp 6	A \flat G \sharp 8	B \flat A \sharp 10		D \flat C \sharp 13																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							</
--	--	--	--	--------------------------------	----------------------------------	--	-------------------------------	-------------------------------	-------------------------------	--	------------------------------	---------------------------------	--	------------------------------	------------------------------	-------------------------------	--	-------------------------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

b = flat

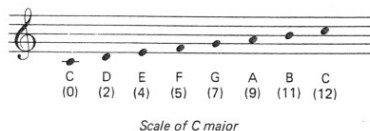
♯ = sharp

As you can see, middle C is 0; all notes above this have positive numbers and all those below have negative numbers. (Positive numbers have a plus sign although this is not usually written and all negative numbers have a minus sign.)

Activity

1. What numbers would the top E and the bottom B, marked on the diagram with question marks, have?

It is all very well having the numbers on the piano keys but if you are not a musician you may not know where these notes are on a written piece of music. The next diagram should help; it shows the key of C major (no sharps or flats) beginning on middle C and ending on the next C up. The scale consists of eight notes, one octave.



The name of the note is written below the note and the corresponding number for the computer is written in brackets. The following program plays this scale; each note is half a second long.

TYPE:

```
10 REM Scale of C major
20 BEEP .5,0: BEEP .5,2: BEEP
.5,4: BEEP .5,5: BEEP .5,7: BEEP
.5,9: BEEP .5,11: BEEP .5,12
30 STOP
```

RUN

DO NOT REMOVE

Notice that each BEEP statement is separated from the next by a colon (:), as each one is a new command.

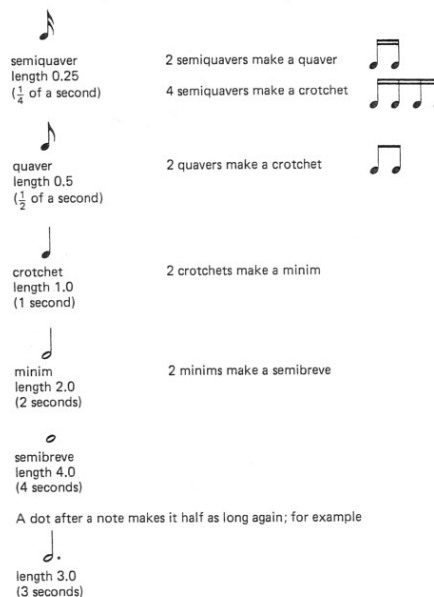
Activities

- Alter the program so that it plays the scale three times.
- Add a PAUSE statement so that there is an interval of 3 seconds between each time the scale is played.

DO NOT REMOVE

We are almost ready to play an actual piece of music but as yet we have kept each note the same length, which rarely happens. We now need to know the length of different notes, so another music lesson is required. See the following diagram, which shows the various lengths of notes (there are additional notes to those detailed in this diagram, but we do not need to know about these at this stage).

Diagram to show the different lengths of notes



Return to the program of the scale of C major.

Alter line 20

```
20 BEEP .5,0: BEEP .25,2: BEEP
.5,4: BEEP .25,5: BEEP .5,7: BE
EP .25,9: BEEP .5,11: BEEP .25,1
2
```

RUN

DO NOT REMOVE

The program now plays the first note as a quaver, the second note as a semi-quaver, the third note as a quaver and so on.

Activity

4. Using the C major scale program:

- make all the notes in the scale semiquavers.
- make the scale play like this



- make the scale play like this



We will now try a proper tune — *The Vicar of Bray*. It is written in the scale of C major but uses an extra note, top D, which if you look back at the diagram of the piano keys you will see is number 14. Each bar has been numbered and this corresponds to a line in the program, so that you can see what is happening. For example, bar 3 is line 230 and bar 9 is line 70. The two dots at the end of bar 5 mean go back to the beginning.



Using the diagram of the scale of C major on page 142 we can write out the number for each note.

Bar 1	Bar 2	Bar 3	Bar 4	Bar 5	Bar 6
7	12, 11, 9, 7, 9	5, 7, 4, 5	7, 0, 5, 4	2, 0	7
Repeated					
Bar 7	Bar 8	Bar 9	Bar 10	Bar 11	
12, 9, 11, 7	12, 11, 9, 11, 7	12, 11, 12, 14, 12, 11	9, 7, 7	12, 11, 9, 7, 9	
Bar 12	Bar 13	Bar 14			
5, 7, 4, 5	7, 0, 5, 4	2, 0			
Same as bars 1 to 5					

The next thing to notice is that one section of the tune is played three times, bars 1 to 5. These bars are played twice at the beginning and then if you look carefully you will see that from the last note of bar 10 to the end of bar 14 is exactly the same as bars 1 to 5. This means that this section of the tune can be put into a subroutine and will therefore only have to be typed in once. We now need to calculate the length of each note and add this to our computer music.

	Bar 1	Bar 2	Bar 3	Bar 4	Bar 5	Bar 6
P	7	12, 11, 9, 7, 9	5, 7, 4, 5	7, 0, 5, 4	2, 0	7
D	1	1, -5, -5, 1, 1	1, 1, 1, 1	1, 1, 1, 1	2, 1	1

	Bar 7	Bar 8	Bar 9	Bar 10	Bar 11
P	12, 9, 11, 7	12, 11, 9, 11, 7	12, 11, 12, 14, 12, 11	9, 7, 7	12, 11, 9, 7, 9
D	1, 1, 1, 1	1, -5, -5, 1, 1	1, -5, -5, 1, -5, -5	2, 1, 1	1, -5, -5, 1, 1

	Bar 12	Bar 13	Bar 14
P	5, 7, 4, 5	7, 0, 5, 4	2, 0
D	1, 1, 1, 1	1, 1, 1, 1	2, 1

P = Pitch
D = Duration (length)

TYPE:

```

10 REM The Vicar of Bray
20 GO SUB 200
30 GO SUB 200
Bar 6 40 BEEP 1,7
Bar 7 50 BEEP 1,12: BEEP 1,9: BEEP 1
      ,11: BEEP 1,7
Bar 8 60 BEEP 1,12: BEEP .5,11: BEEP
      .5,9: BEEP 1,11: BEEP 1,7
Bar 9 70 BEEP 1,12: BEEP .5,11: BEEP
      .5,12: BEEP 1,14: BEEP .5,12: B
      EEP .5,11
Bar 10 80 BEEP 2,9: BEEP 1,7 (only two notes as the other G (7)
      90 GO SUB 200 is the beginning of the subroutine)
      100 STOP
      200 REM Subroutine
Bar 1 210 BEEP 1,7
Bar 2 220 BEEP 1,12: BEEP .5,11: BEEP
and 11 .5,9: BEEP 1,7: BEEP 1,9
Bar 3 230 BEEP 1,5: BEEP 1,7: BEEP 1,
and 12 4: BEEP 1,5
Bar 4 240 BEEP 1,7: BEEP 1,0: BEEP 1,
and 13 5: BEEP 1,4
Bar 5 250 BEEP 2,2: BEEP 1,0
and 14 260 RETURN

```

RUN

The speed of the tune may seem a little slow, this is because we have taken a crotchet

to be one second long for convenience; it should be slightly shorter, probably 0.8 instead of 1. However, it is simpler to keep the crotchet as 1 as this makes it much easier to calculate the values of the other notes.

Activity

5. Write a program to play the tune *London's Burning*. The music is provided below. Notice that each little phrase is repeated, so to save typing use FOR-NEXT loops.



Let us look at one musical key that uses a sharp (#). Many pieces of music are written in the key of G major which has F sharp. Any tune written in this key will look like this:



This key starts and finishes on a G and F# is played instead of an ordinary F. If you look back to the diagram of the piano keys, you will see that the following numbers would play this scale beginning on the first G below middle C:

↓ middle C
G A B C D E F# G
-5, -3, -1, 0, 2, 4, 6, 7

Activities

6. (a) Type in a program to play the scale of G major.
(b) Write out the numbers for the next octave of the scale beginning on the first G above middle C.
(c) Add to your program the second octave of the scale.
7. Write a program for *Good King Wenceslas*; the music is given on the next page. Notice the key signature. Only notes that you used in activity 6 are needed. Notice that the first four bars are repeated and bars 9 and 15 are also the same. Can you save yourself some typing?



Typing in a line of music is rather tedious because of all the BEEP statements needed. The following program uses READ and DATA statements in a FOR-NEXT loop which cuts out all the BEEPs except one!

TYPE:

```
10 REM Nursery rhyme
20 FOR n=1 TO 38
30 READ a,b
40 BEEP a,b
50 NEXT n
60 DATA 1,0,1,0,1,7,1,7,.5,9,.
5,11,.5,12,.5,9,2,7,1,5,1,5,1,4,
1,4,1,2,1,2,2,0,1,7,.5,7,.5,7,1,
5,.5,5,.5,5,1,4,.5,4,.5,4,1,5,2,
.5,2,1,7,.5,7,.5,7,.25,5,.25,7,.
25,9,.25,5,1,4,.5,2,.5,2,2,0
70 STOP
```

RUN. Do you know the tune? DO NOT REMOVE

There are 38 notes in the tune so the computer goes round the FOR-NEXT loop 'n' 38 times, playing a different note each time. The READ statement picks up the first two values from the DATA statement which are 1 and 0, so the first note is a crotchet on middle C. The second time round the loop the values 1 and 0 are picked up again as these are the third and fourth values in the DATA statement. The third time round the loop 1 and 7 are picked up. So each time through the loop two values are taken from the DATA statement which make up one note. This continues until all the DATA have been used.

Add to the program:

```
15 FOR m = 1 TO 2
65 NEXT m
```

RUN. What happens and why?

An error message results because the computer has run out of data. The first time round the loop the READ statement uses up all the values in the DATA statement, so the second time round the 'm' loop there is no data left. It is possible to make the computer play the tune twice which is what we were hoping to do. What we want is to put the notes back into the DATA statement. We can do this by using the RESTORE statement above key S (extended mode).

Add to the program:

63 RESTORE

RUN the program again. DO NOT REMOVE

RESTORE is an instruction that tells the computer to begin READING from the start of the list of DATA. We can separate the two verses of the rhyme by using the PAUSE statement. (Remember that PAUSE 50 is 1 second.)

Add to the program:

64 PAUSE 3 * 50

RUN

There is now a 3-second interval between the two verses.

Activity

8. Write a program using READ and DATA statements for four verses of *The Holly and the Ivy*, with an interval of 2 seconds between the verses.



Using the INKEY\$ function, as we did in unit 23, we can make the keyboard keys act like piano keys. This final program uses the keys 1 to 8 for the notes of the C major scale beginning with C on key 1, D on key 2 and so on. The length of the note remains the same so variable 'a' is used to store this number.

TYPE:

```

10 REM Piano
20 LET a=.25
30 IF INKEY$="1" THEN BEEP a,0
40 IF INKEY$="2" THEN BEEP a,2
50 IF INKEY$="3" THEN BEEP a,4
60 IF INKEY$="4" THEN BEEP a,5
70 IF INKEY$="5" THEN BEEP a,7
80 IF INKEY$="6" THEN BEEP a,9
90 IF INKEY$="7" THEN BEEP a,1
100 IF INKEY$="8" THEN BEEP a,1
110 GO TO 30

```

RUN. Press any of the keys between 1 and 8. If you hold a key down too long the note just repeats, it does not produce a longer note. Can you play any tunes?

Try this tune: 1, 3, 2, 4, 3, 5, 3, 1
1, 3, 2, 4, 3, 1
1, 3, 2, 4, 3, 5, 3, 1
6, 2, 4, 3, 1

The only way to stop this program is to press CAPS SHIFT and BREAK.

Unit 27: Some useful functions — RND, RAND and INT

Random, abbreviated to RND above key T, is a function of the computer that produces a mixed selection of numbers from a list of 65536 numbers. In other words it chooses a number or numbers at random from this list.

TYPE:

```
10 REM Random numbers
20 FOR n=1 TO 10
30 PRINT RND
40 NEXT n
50 STOP
```

RUN. Make a note of the first 3 numbers that appear on the screen.
DO NOT REMOVE.

The result is a list of ten numbers ranging from 0 to 0.99999999; you will never get the number 1.

RUN the program again — the list of numbers should be different from the previous one — check the three numbers that you made a note of.

Another list of numbers has appeared, once again none of the numbers will be below 0 and none above 0.99999999. The numbers look random enough but in fact the computer is following a set sequence, although it would be difficult to work this out. However, this means that we can make the computer pick the same group of numbers every time by making it start at the same point in the sequence. To do this we use the function RANDOMIZE which is abbreviated on key T to RAND. For example, RAND 35 will start picking numbers from the sequence starting at the 35th number.

Add to the program;

```
15 RAND 35 (notice the whole word is printed)
```

RUN. Make a note of the first three numbers.

RUN the program again. Check that the numbers are the same.

Having checked your numbers you can see that RAND does fix the starting point of the sequence. If RND is used on its own the computer will just start anywhere in the list of 65536 numbers.

Now you may be thinking that numbers from 0 to just under 1 are not very useful and are rarely used, and that normally a much wider range of numbers is required. We can produce higher numbers by multiplying RND by another number. For example

```
6 * RND gives numbers from 0 to 5.999999
64 * RND gives numbers from 0 to 63.999999
```

TYPE:

```
10 REM Random numbers
20 PRINT "6*RND", "64*RND"
30 FOR n=1 TO 10
40 PRINT 6*RND, 64*RND
50 NEXT n
60 STOP
```

RUN the program several times. Notice that the number 6 will never appear in the first column and the number 64 will never appear in the second column.

So far all the numbers that have appeared on the screen have looked rather complicated because of the figures that have appeared after the decimal point. Usually we want only whole numbers, so we need a function that will remove all these unnecessary decimal places. The function that we use is INT, written above key R (extended mode). INT is short for *integer* which means 'whole number'. INT chops off all the decimal places. For example

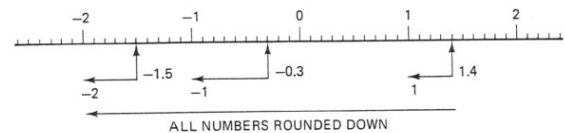
```
INT 1.4809143 becomes 1
INT 5.9621567 becomes 5
INT 62.5132789 becomes 62
```

Notice that all the numbers are being rounded down, even those that are nearer to the next highest number; 5.9621567 became 5 although it is nearer to 6. This also applies to negative numbers. For example

```
INT -1.2678957 becomes -2
INT -10.6723142 becomes -11
```

Do not think that these numbers have been rounded up, remember that -2 is lower than -1 and -11 is lower than -10.

Look at the number line below to convince yourself that all these numbers have been rounded down.



The next program proves this point.

TYPE:

```

10 REM Integers
20 PRINT INT 5.6
30 PRINT INT 62.9
40 PRINT INT 1.4
50 PRINT INT -1.2
60 PRINT INT -10.6
70 STOP

```

RUN

```

The result is:  5
               62
               1
              -2
             -11

```

Activity

1. What are the values of the following:

- | | |
|---------------|---------------|
| (a) INT 4.5 | (e) INT -87.1 |
| (b) INT 103.9 | (f) INT -0.87 |
| (c) INT 99.1 | (g) INT 0.23 |
| (d) INT -5.6 | |

We will now use INT and RND together to give us random whole numbers.

TYPE:

```

10 REM Random whole numbers
20 PRINT "INT (-10*RND)", "INT
(10*RND)"
30 FOR n=1 TO 10
40 PRINT INT (-10*RND), INT (10
*RND)
50 NEXT n
60 STOP

```

RUN several times. Notice that 0 can never appear in the first column of negative numbers and that 10 can never appear in the second column of positive numbers.

Producing random whole numbers can be useful when writing a games program. In units 19 and 25 you had to write a guessing-a-number game, where the number to be guessed had to be put into the program while it was running. This game would have been much better if the computer had secretly produced its own number that nobody could see. Using INT and RND we will now improve this game.

TYPE:

```

10 REM Guessing a number
20 LET c=1
30 LET x=INT (21*RND)
40 PRINT "Guess the number bet
ween 0 and 20"
50 INPUT "Enter a number ";n
60 IF n=x THEN GO TO 100
70 LET c=c+1
80 PRINT n;" Wrong,try again"
90 GO TO 50
100 PRINT "Correct,you took ";c
;" guesses"
110 STOP

```

RUN the program several times.

Line 30 produces a number between 0 and 20 which is then compared with 'n', the number entered in line 50.

Activity

2. Why was line 30 not written as follows?

```
30 LET x = INT (20 * RND)
```

Sometimes in a set of random numbers 0 is not required but the next highest number is. For example, if the numbers 1 to 6 are required at random, as in a dice-throwing game, INT (6 * RND) produces 0 to 5 and INT (7 * RND) produces 0 to 6, neither of which is required. To overcome this problem we use INT (6 * RND) + 1; by adding one the noughts generated become ones and the fives become sixes, giving us the range 1 to 6.

TYPE:

```

10 REM Random numbers 1 TO 6
20 FOR n=1 TO 10
30 PRINT INT (6*RND)+1
40 NEXT n
50 STOP

```

RUN the program several times, checking that the numbers generated fall only between 1 and 6.

The next program uses this idea to count the number of times that the number 3 occurs when a die (the singular of 'dice') is thrown a hundred times.

TYPE:

```

10 REM Throwing a die a 100 ti
mes
20 LET c=0
30 FOR x=1 TO 10

```

```

40 FOR y=1 TO 10
50 LET d=INT (6*RND)+1
60 PRINT d;" ";
70 IF d=3 THEN LET c=c+1
80 NEXT y
90 PRINT
100 NEXT x
110 PRINT
120 PRINT "The number of times
3 appeared was ";c
130 STOP

```

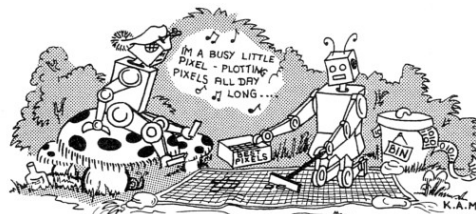
RUN the program. Check the answer.
DO NOT REMOVE.

No explanation of this program is required as all the other techniques used have been met before.

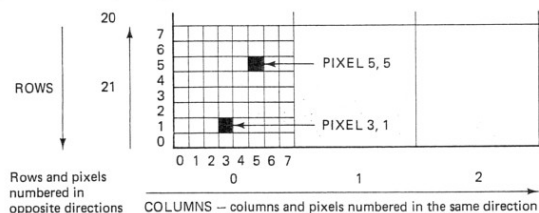
Activities

3. Modify the previous program so that users can enter the number they wish to be counted.
4. Write a program to print a list of 20 random whole numbers between 0 and 100.
5. Write a program to print a list of 15 random whole numbers between 1 and 20.
6. Write a program to print the same sequence of 10 random whole numbers from 0 to 10 every time that the program is run.
7. Write a program to print 100 random whole numbers from 1 to 5 in ten columns of ten rows and to record and print how many times each number occurs.
8. Write a program to test a person's multiplication tables between 2 and 12. The multiplication sum should be produced at random. The users should be told if they are correct; if they are incorrect, the right answer should be given.
9. (a) Write a program to throw two dice 50 times and to print out the number of times the total of 7 occurs.
(b) Add another loop so that the program is repeated 5 times and each time the total is added to the previous total to give a grand total. The program should finish by printing the average number of times that 7 occurs.
10. Write a program to count the number of times heads and tails occur when a coin is tossed 80 times. Let heads equal 0 and tails equal 1. The result should be printed.

Unit 28: High-resolution graphics — PLOT, DRAW, CIRCLE and INVERSE



High-resolution graphics involves the use of *pixels*. A pixel is a very small dot. Each character position on the screen is made up of eight pixels across and eight pixels up the side. The diagram that follows, which has been enlarged, shows the character position 21, 0 divided into 8×8 tiny squares — each tiny square is a pixel. Two examples of pixels have been marked in.



There are 32 (0-31) columns so there are $32 \times 8 = 256$ pixel positions across the screen; these are numbered in the same direction as the columns. There are 22 (0-21) rows so there are $22 \times 8 = 176$ pixel positions up the side of the screen. These are not numbered like the rows as they begin at 0 at the bottom of the screen. See diagram 28.1, which can be used as a guide for positioning pixels. When referring to a pixel 'x' and 'y' coordinates are used, as for plotting

points on a mathematical graph. 'x' represents the number across the screen, remember 0 is on the left-hand side, and 'y' represents the number up the screen. Remember that 0 is at the bottom of the screen. The 'x' coordinate is always given first, so the pixels drawn in diagram 28.1 are at 3, 1 and 5, 5. Obviously the screen is not visibly divided up into tiny squares, so before plotting pixels and drawing lines it is necessary to use a copy of diagram 28.1.

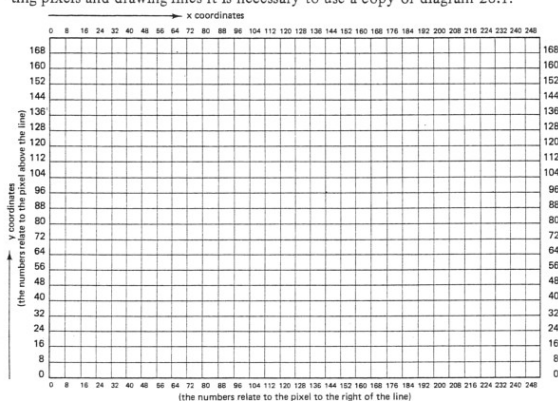
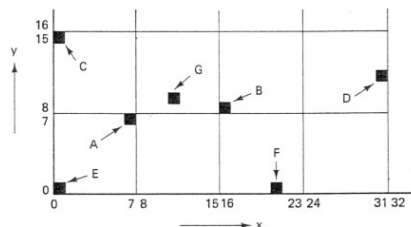


Diagram 28.1 Diagram to show the numbering of the pixels

Activity

1. Give the x and y coordinates of the pixels drawn below:



To place pixels on the screen we use the command PLOT (key Q).

TYPE:

```
10 REM Plotting pixels
20 INPUT "Enter x coordinate(0
-255) ";x
30 INPUT "Enter y coordinate(0
-175) ";y
40 PLOT x,y
50 GO TO 20
```

RUN. Enter numbers and watch the pixels being plotted. Notice how small they are.

If you enter an x coordinate larger than 255 or a y coordinate larger than 175, the program will crash and the error message

B Integer out of range, 40 : 1

will appear. To stop this happening add

```
33 IF x > 255 THEN LET x = 255
36 IF y > 175 THEN LET y = 175
```

This prevents the pixels going off the screen.

Notice that you cannot have negative coordinates.

Enter STOP to stop the program.

In the next program the computer plots 200 pixels at random all over the screen; it is rather like a person coming out in measles.

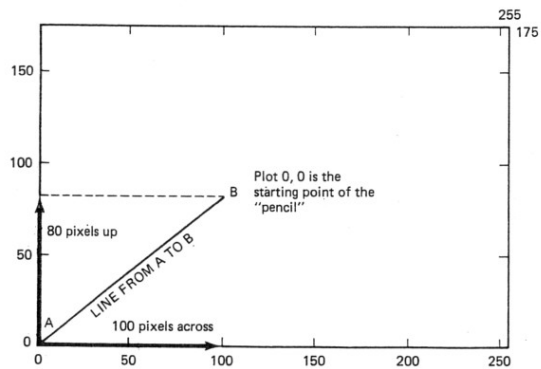
TYPE:

```
10 REM Plotting at random
20 FOR p=1 TO 200
30 LET x=INT (256*RND)
40 LET y=INT (176*RND)
50 PLOT x,y
60 NEXT p
70 STOP
```

RUN

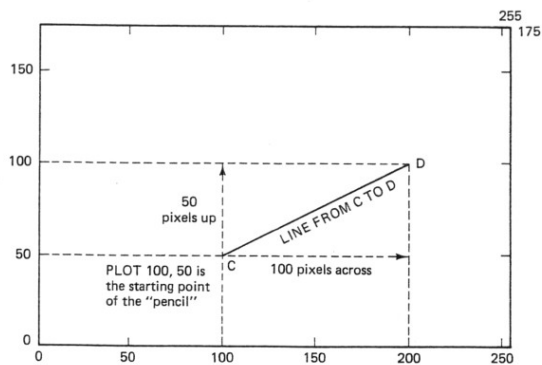
Using PLOT on its own is not very useful unless we are actually drawing a graph but we need to understand the command in order to draw lines. For example, when we draw a line we need to plot the point where the 'pencil' is going to start. The length and direction of the line is given by adding on the number of pixels across and up the screen, and by using the command DRAW (key W).

Example 1: Drawing a line from A to B



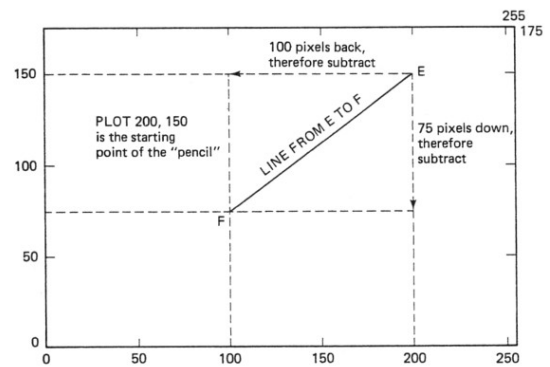
```
TYPE 10 PLOT 0, 0
      20 DRAW 100, 80
RUN
```

Example 2: Drawing a line from C to D



```
TYPE 10 PLOT 100, 50
      20 DRAW 100, 50 (notice that this is not the finishing point of the
                        line which if plotted would be 200, 100)
RUN
```

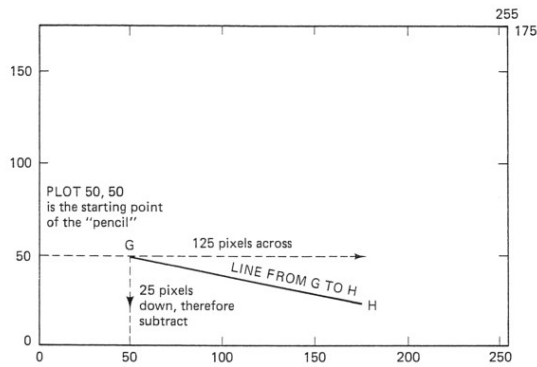
Example 3: Drawing a line from E to F



```
TYPE: 10 PLOT 200, 150
      20 DRAW -100, -75
RUN
```

This program shows that, unlike the PLOT statement, the DRAW statement does use negative numbers.

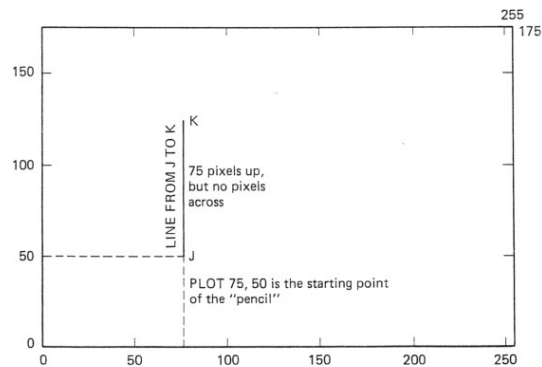
Example 4: Drawing a line from G to H



```
TYPE: 10 PLOT 50, 50
      20 DRAW 125, -25
```

```
RUN
```

Example 5: Drawing a line from J to K



```
TYPE 10 PLOT 75, 50
      20 DRAW 0, 75
RUN
```

Activity

2. (a) Diagram 28.2 shows a screen with six lines on it, A, B, C, D, E and F. The arrows on the lines show the direction in which the lines have been drawn. Write a program that draws these lines in approximately the correct positions.
- (b) Using PRINT AT, label each line so that its letter appears in an appropriate place (remember with PRINT AT that the rows are numbered differently and that the row and not the column is given first — see unit 23).

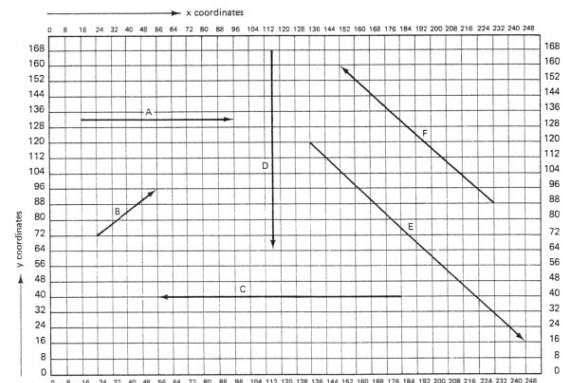
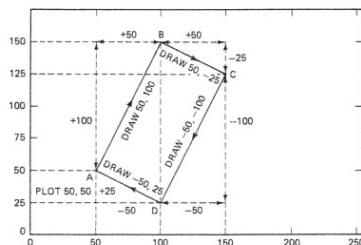


Diagram 28.2

If we want to draw a rectangle it is not necessary to keep plotting the point of the 'pencil' at the beginning of each side. After the first side of the rectangle has been drawn the 'pencil' remains at the end of that line, so we can automatically draw the next side from that point; all we need is another DRAW statement. The only point to be plotted is right at the beginning when we start the drawing, which is point A on the following diagram:



TYPE:

```

10 REM Rectangle ABCD
20 PLOT 50,50           (beginning of drawing - plots pencil point.)
30 DRAW 50,100          (draws AB, pencil now at B)
40 DRAW 50,-25          (draws BC, pencil now at C)
50 DRAW -50,-100        (draws CD, pencil now at D)
60 DRAW -50,25          (draws DA, pencil back at A)
70 STOP

```

RUN

Activity

3. Write programs that draw the shapes given on diagram 28.3. Print the correct letter in the centre of each shape.

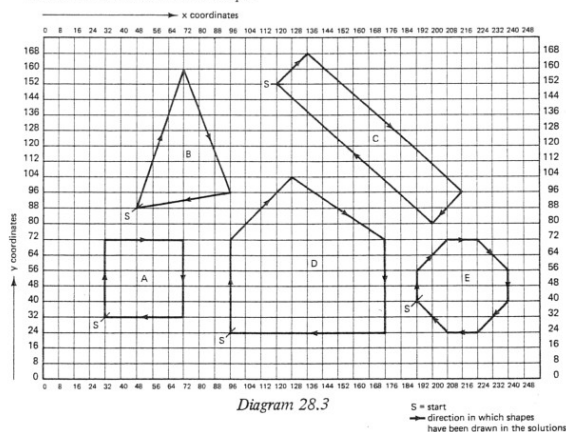


Diagram 28.3

S = start
 → direction in which shapes
 have been drawn in the solutions

We can draw any rectangle by putting the necessary information into the program as it is run. The only information required is the starting point of the 'pencil', and the width and the height of the rectangle. The next program enables the user to draw five rectangles.

TYPE:

```

10 REM Drawing any rectangle
20 FOR n=1 TO 5
30 INPUT "Enter x coordinate "
;x
40 IF x<0 OR x>255 THEN PRINT
"Not acceptable": GO TO 30
50 INPUT "Enter y coordinate "
;y
60 IF y<0 OR y>175 THEN PRINT
"Not acceptable": GO TO 50
70 INPUT "Enter width of recta
ngle ";w
80 IF x+w>255 THEN PRINT "Too
wide": GO TO 70
90 INPUT "Enter height of rect
angle ";h
100 IF y+h>175 THEN PRINT "Too
tall": GO TO 90
110 CLS
120 PLOT x,y
130 DRAW 0,h
140 DRAW w,0
150 DRAW 0,-h
160 DRAW -w,0
170 NEXT n
180 STOP

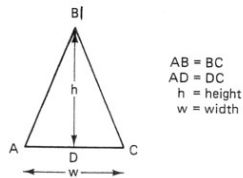
```

RUN, putting in the required information.

Lines 40 and 60 make certain that the coordinates x and y will fit on to the screen. Lines 80 and 100 check that the rectangle will fit on to the screen. For example, if x is 200 and then the user enters 100 for the width 'w', then obviously x + w must be greater than 255 and will therefore be off the screen. Line 110 clears any messages or previous drawings from the screen. The rectangle is drawn by lines 120 to 160.

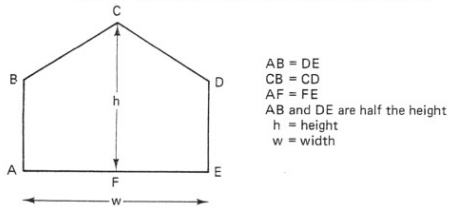
Activities

4. Write a program that allows the user to draw different sized isosceles triangles. (An isosceles triangle has two equal sides.) Use the diagram overleaf to help you.



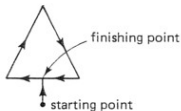
The program should check that the triangle will fit on to the screen. Begin the drawing at A.

5. Add a section to your program that asks users whether they would like to draw triangles or pentagons. If the answer is triangles, then the computer should be sent to the program that you have just written; if the answer is pentagons, then the computer should be directed to a new section of your program that will do this. The diagram below will help you to write this new section.



Notice that the information required is exactly the same as before (the starting point, the height and the width), so this part of the program will be exactly the same as before, and there is no need to write it twice.

6. Write a program that draws the picture shown in diagram 28.4. All the trees are the same size, so they can be drawn by using a FOR-NEXT loop. The only different numbers to be entered are the coordinates needed for the starting point of the 'pencil' for each tree; this information could be placed in a DATA statement. The trees can each be drawn with one continuous line, as shown in the diagram below.



The houses can be drawn in a similar way, as they are also all the same size. The roads are a little more tricky but they too can be entered by one continuous line, so it is necessary to plot the starting point only once.

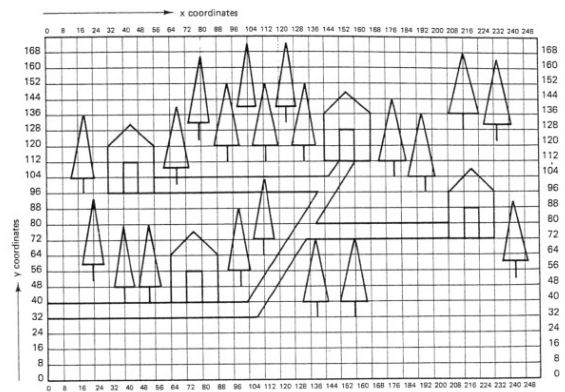
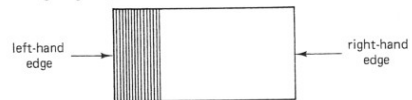


Diagram 28.4

7. Try writing a program for one of the following:
- a picture of your own using only straight lines
 - or
 - a pattern using lots of overlapping shapes.

Shading in a shape

It is possible to shade or colour in the inner area of a shape. It is easiest to see this with a rectangle. What we need to do is to draw in all the vertical lines that exist between the left-hand side and the right-hand side of the rectangle. Look at the following diagram;



If these vertical lines are placed closely together they give the impression of colouring in.

TYPE:

```
10 REM Shading in a rectangle
20 INPUT "Enter x coordinate "
;X
```



```

30 INPUT "Enter y coordinate "
:Y
40 INPUT "Enter width ";w
50 INPUT "Enter height ";h
60 PLOT x,y
70 DRAW 0,h
80 DRAW w,0
90 DRAW 0,-h
100 DRAW -w,0
110 REM Shading
120 FOR n=1 TO w-1           (number of vertical lines to be drawn)
130 PLOT x+n,y              (start of 'pencil' each time loop is
140 DRAW 0,h-1              executed - x is increased by 1 so the
150 NEXT n                  pencil moves across one place)
160 STOP

```

RUN. Be careful to enter suitable numbers as we have deliberately missed out the lines that check the size in order to make the program shorter.

DO NOT REMOVE

This program produces a black square as the vertical lines are placed so close together that they touch each other. However, to prove to yourself that the shading does consist of vertical lines, try this:

Edit line 120

```

120 FOR n=1 TO W-1 STEP 2
145 PAUSE 50

```

RUN

You can now see the vertical lines being drawn in. The rectangle could have been shaded in by using horizontal lines beginning at the bottom.

Activity

8. Try the suggestion made above — edit the previous program rather than rewrite the whole thing.

Drawing circles

For circles we use the command CIRCLE below key H (extended mode and then symbol shift). For example

CIRCLE 120, 88, 20

↑ ↑ ↑ radius of the circle — number of pixels
x and y (distance from the centre to the edge
coordinates of the circle)
for the
centre of
the circle

TYPE: 10 CIRCLE 120, 88, 20
RUN

This gives us one circle near the centre of the screen. We can alter this program so that the user can draw different sized circles anywhere on the screen, checking at the same time that they will fit on to the screen.

TYPE:

```

10 REM Circles
20 INPUT "Enter x coordinate f
or centre of circle ";x
30 IF x<0 OR x>255 THEN PRINT
"Not acceptable": GO TO 20
40 INPUT "Enter y coordinate
for centre of circle ";y
50 IF y<0 OR y>175 THEN PRINT
"Not acceptable": GO TO 40
60 INPUT "Enter radius of circ
le ";r
70 IF x+r>255 OR x-r<0 THEN PR
INT "Off sides": GO TO 60
80 IF y+r>175 OR y-r<0 THEN PR
INT "Off top or bottom": GO TO 6
0
90 CLS
100 CIRCLE x,y,r
110 GO TO 20

```

RUN. Enter several different numbers, sometimes deliberately entering numbers that you know will not be acceptable, and checking that all parts of the program work. Use diagram 28.1 as a guide.

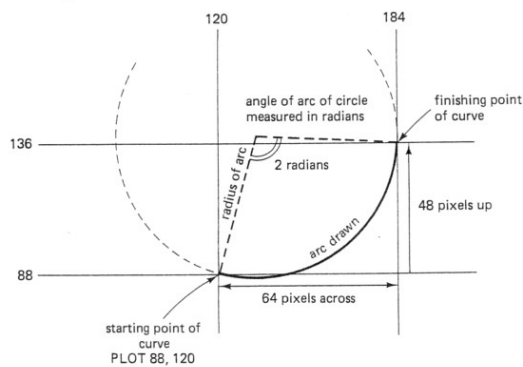
Drawing curves or arcs

Curves or arcs are parts of circles. They are not as simple to draw as circles. The commands PLOT and DRAW are used. For example

PLOT 120, 88 – gives the starting point of the ‘pencil’
 DRAW 64, 48, 2, ← angle of arc of circle, measured in radians
 ↑ ↑ – not degrees

number of pixels across	number of pixels up
----------------------------------	------------------------------

See the diagram below:



```
TYPE: 10 PLOT 120, 88
      20 DRAW 64, 48, 2
RUN
```

It is difficult to visualise an angle in radians but if you think of 5 radians as being almost equal to 360° , this gives you almost a full circle.

TYPE: 10 PLOT 120, 88
20 DRAW 64, 48, 5

RUN

You will not find it easy to judge the size and shape of an arc, so it is best just to experiment with the idea.

Activity

9. Write a program that draws the picture of the Viking ship shown in diagram 28.5.

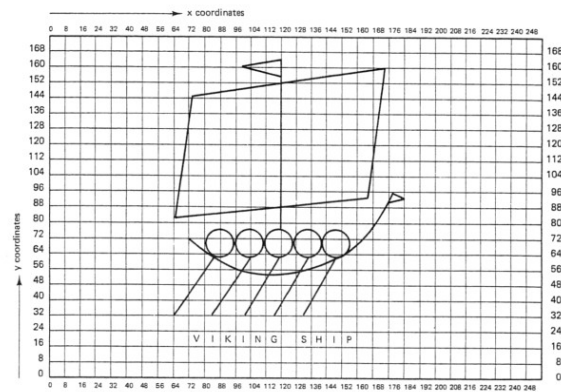


Diagram 28.5

Rubbing out lines

We can remove or rub out a line or circle by using the INVERSE command. INVERSE (key M – extended mode and then SYMBOL SHIFT) was used to reverse the ink and paper colours in unit 23 on low-resolution graphics. INVERSE 0 will produce ink dots in the normal way but INVERSE 1 will change the ink dots to paper dots (white) and so appear to rub out the ink dots.

TYPE:

```

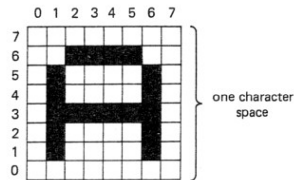
10 REM Rubbing out
20 PLOT 24,40
30 DRAW INVERSE 0;180,112
40 PAUSE 30
50 DRAW INVERSE 1;-180,-112
60 CIRCLE INVERSE 0;128,88,50
70 CIRCLE INVERSE 1;128,88,50
80 STOP

```

RUN

Unit 29: User-defined characters

There are 21 characters on the Spectrum that you can alter and make into new characters. The keys that can be altered are the letters A to U. As you know, each character space is made up of 8×8 pixels. For example, the letter A can be made up as in the diagram below:



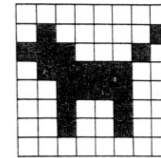
In order for this character to be understood by the computer the white spaces, known as *paper dots*, have to be given the value 0 and the black spaces, known as *ink dots*, have to be given the value 1. So to the computer the character A looks like this:

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	0
0	1	1	1	1	1	1	0
0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0

A mathematician would know that numbers consisting of eight noughts or ones are *binary* numbers. We do not need to understand binary arithmetic but the word, in its abbreviated form 'BIN', will occur in the next program.

To make our own character we need to design the character first. The diagram that follows shows a dog!

170



Translated into binary numbers this character looks like this:

0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
1	1	1	0	0	0	1	0
0	1	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0

We now have to put this into the computer's memory. We use three commands for this:

POKE (key O) which means put or poke something into the memory
 USR (above key L — extended mode) which tells the computer that the
 USER is going to make a character
 BIN (above key B — extended mode) which gives the computer binary
 numbers that it can understand.

The only other thing is to decide which letter between A and U we are going to use for our character. I have chosen D for obvious reasons.

TYPE:

```

10 REM Dog
20 POKE USR "D",BIN 00000000
30 POKE USR "D"+1,BIN 01000001
40 POKE USR "D"+2,BIN 11100010
50 POKE USR "D"+3,BIN 01111100
60 POKE USR "D"+4,BIN 00111100
70 POKE USR "D"+5,BIN 00100100
80 POKE USR "D"+6,BIN 00100100
90 POKE USR "D"+7,BIN 00000000
100 PRINT "D"      Enter 'D' in graphics mode
RUN

```

This program prints one dog. To print many dogs we must alter the program.

Edit line 100:

```

100 FOR n=1 TO 20

```

Add:

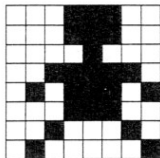
```
100 FOR n=1 TO 20
110 FOR m=1 TO 15
120 PRINT "D";" ";
130 NEXT m      Enter 'D' in graphics mode
140 PRINT      (dog character will appear)
150 NEXT n
160 STOP
RUN
```

Points to notice:

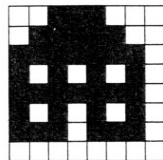
1. Letter D is in quotation marks and entered in graphics mode (line 120).
2. Each row in the character is given a number. Therefore the top row is "D" + 0, but is just written as "D", the second row is "D" + 1, the third row "D" + 2 and so on. D in these rows is entered in normal letter mode (lines 20 to 90).
3. In lines 20 to 90 there is a comma before the BIN statement.

Activities

1. Write programs that produce the following characters:
 - (a) A fat little man



- (b) A house



2. Design your own character of a car. Write the program for the character and then make the car travel across the screen.

Unit 30: Some interesting string functions

LEN

The function LEN, above key K — extended mode, is an abbreviation for *length* and, as the name suggests, calculates the length of strings.

TYPE:

```
10 REM Using LEN
20 LET a$="cat"
30 PRINT LEN a$
40 LET b$="hippopotamus"
50 PRINT LEN b$
60 LET c$=""
70 PRINT LEN c$
80 STOP
RUN
```

The result is: 3
12
0

So the PRINT statements in lines 30, 50 and 70 do not operate in the normal way by printing out the contents of the string variables, a\$, b\$ and c\$, but give instead the length or number of characters in the string. A string variable containing nothing, like c\$ in line 60, will return the value 0.

A program can be written to calculate the length of any string that is in a list of data.

TYPE:

```
10 REM Length of strings
20 FOR n=1 TO 6
30 READ a$
40 PRINT a$;" has ";LEN a$;" l
eters"
50 DATA "LEN","calculates","th
e","length","of","strings"
60 NEXT n
70 STOP
RUN
```

Activity

1. Write a program where the user can input any eight words, and where the word and its length is printed out.

The next program uses LEN to set the number of times a FOR-NEXT loop will run. It prints out the word that is entered and then prints out the same word by adding one letter at a time until the word is complete. Line 50 uses the technique of extracting parts of strings, met in unit 10.

TYPE:

```
10 REM Printing a word letter
   by letter
20 INPUT "Enter a word ";a$
30 PRINT a$
40 FOR n=1 TO LEN a$
50 PRINT a$(1 TO n)
60 PAUSE 50
70 NEXT n
80 STOP
```

RUN. Enter the word 'Spectrum'.

The result is: Spectrum
S
Sp
Spe
Spec
Spect
Spectr
Spectru
Spectrum

The word 'Spectrum' has eight characters so the FOR-NEXT loop 'n' will run eight times. Line 50 extracts the part of the string from the first character to the 'n'th character. The first time round the loop 'n' is 1 so 1 TO 1 is S, the second time round the loop 'n' is 2 so 1 TO 2 is Sp, and so on.

RUN the program again putting in different words.

We can print out words (or sentences or characters) in reverse, in the same way. Modify the previous program:

```
40 FOR n = LEN a$ TO 1 STEP - 1
50 PRINT a$(n TO n)
```

RUN the program again, entering different words or phrases.

CHR\$ and CODE

The function CHR\$ (above key U – extended mode), an abbreviation of *character*,

will give the character at any given number. There are 256 characters in the computer's memory and each one has a code between 0 and 255. There is a complete list of the characters on pages 183-188 of the Spectrum's manual.

TYPE:

```
10 REM Characters
20 PRINT CHR$ 37
30 PRINT CHR$ 101
40 PRINT CHR$ 135
50 PRINT CHR$ 198
60 PRINT CHR$ 237
70 STOP
RUN
```

The result is: %

e

AND

GO SUB

You can check in the manual if these are the correct characters for those numbers. Notice that keywords such as GO SUB are treated as single characters.

TYPE:

```
10 FOR n=32 TO 255
20 PRINT CHR$ n;
30 PAUSE 40
40 NEXT n
50 STOP
RUN
```

This program produces all the characters from 32 to 255. There are two points to notice:

- (1) Characters 0 to 31 were not included in the program because they do not produce anything printable as they are control characters. Character 32 is a space, so on the screen you have a gap at the beginning of the first row.
- (2) The upper case letters A to U appear twice; this is because, on their second appearance, they are representing the user-defined characters.

The CODE function (above key I – extended mode) does the opposite to CHR\$. It will give the number of the character.

TYPE:

```
10 REM Using CODE
20 PRINT CODE "A"
30 PRINT CODE "e"
40 PRINT CODE "INKEY$"
50 PRINT CODE " CIRCLE "
60 PRINT CODE "hello"
70 STOP
RUN
```

The result is: 65
96
166
216
104

Again you can check these in the manual if you wish. Notice that for CODE 'hello' the result 104 is the CODE for the letter 'h', so if a normal word is used only the CODE for the first letter is given. CODE 'INKEYS' and CODE 'CIRCLE' produce the CODE for the whole word as they are keywords.

VAL (not to be confused with VAL\$)

The function VAL (above key J — extended mode), an abbreviation for *evaluate*, can change a string to a number.

```
TYPE:
10 REM Using VAL
20 LET a$="1+2+3+4"
30 PRINT a$
40 PRINT VAL a$
50 LET b$="10*10"
60 PRINT b$
70 PRINT VAL b$
80 STOP
RUN
```

Lines 30 and 60 print out the contents of a\$ and b\$ in the normal way, but the VAL function in lines 40 and 70 makes the computer treat the contents of the string variables as numbers. Therefore VAL a\$ gives the value of 1 + 2 + 3 + 4 and VAL b\$ gives the value of 10 * 10.

Unit 31: Using the printer

The ZX Printer is fixed along the edge connector at the back of the computer. It should not be attached or detached while the computer is on. The keywords LPRINT, LLIST and COPY are used in connection with the printer.

LPRINT (above key C — extended mode)

```
TYPE: 10 LPRINT "The result of this program will be printed on paper"
      20 LPRINT "The result will not appear on the screen"
RUN      DO NOT REMOVE
```

The result is exactly as stated in the program so if for any reason you only want the result on paper then use LPRINT instead of PRINT.

LLIST (above key V — extended mode)

Add to the program

```
      30 LLIST
RUN      REMOVE PROGRAM
```

In response to LLIST the computer produces a list of the program on the paper.

```
TYPE: 10 PRINT "The result of this program will appear on the screen"
      20 PRINT "A list of this program will also be printed out on paper"
      30 LLIST
RUN      DO NOT REMOVE
```

Now the result is printed on the screen and the program list on the paper.

COPY (keyword on key Z)

Change lines 20 and 30

```
      20 PRINT "and will also be printed on paper"
      30 COPY
RUN
```

This time the result of the program is on the screen and on the paper. When using COPY the printer runs on, unnecessarily wasting valuable paper; this can be stopped by pressing CAPS SHIFT and BREAK.

It is possible to purchase special connectors (called *interfaces*) which allow the Spectrum to be connected to a wide variety of alternative printers. They are usually accompanied by a short computer program which has to be loaded into the computer in order to 'translate' the signals before sending them to the printer. In this way, quite sophisticated results can be obtained if a daisy-wheel printer or high-quality dot-matrix printer is used. You should decide on the make of printer you require, and then check that an interface can be obtained, before purchasing anything.

Solutions

Unit 3

```
1. 10 PRINT "Mr.J.Brown"
    20 PRINT "51,Bedford Square"
    30 PRINT "LONDON"
    40 PRINT "W.C.1."

2. 10 PRINT "My name is Mary Jones"
    20 PRINT "My age is 21 years"
    30 PRINT "My date of birth is 2nd May 1962"
```

```
3. 10 PRINT "There are five people in my family"
    20 PRINT "I have one brother and one sister"
    30 PRINT "My sister's name is Janette"
    40 PRINT "My brother's name is Keith"
    50 PRINT "My parents live in London"
```

Unit 8

```
1. 2. 3.
32 72 a) ABC f) Abc
32 dogs c) abc i) A2B2C
y 53 d) Two ABC j) a b c
```

Unit 9

```
1. 10 LET a$="John"
    20 LET s$="school"
    30 PRINT a$;" goes to ";s$
```

```
2. 10 LET m$="Mary"
    20 LET a$="Anne"
    30 LET b$="12 years"
    40 LET c$="13 years"
    50 PRINT m$;" is ";b$;" old"
    60 PRINT a$;" is ";c$;" old"
```

```
3. 10 LET a$="Robert"
    20 LET b$="Jane"
    30 LET c$="London"
    40 LET d$="Liverpool"
    50 PRINT a$;" lives in ";c$
    60 PRINT b$;" lives in ";d$

4. 10 LET n$="Ralph"
    20 LET d$="12th September"
    30 LET w$="Saturday"
    40 LET t$="8 o'clock"
    50 PRINT "Dear ";n$
    60 PRINT "Please come to a party on ";w$;" the ";d$;" at ";t$
    70 PRINT "love from"
    80 PRINT "Sally"
```

```
5. 10 LET a$="James"
    20 LET b$="Harriot"
    30 PRINT a$+b$
```

```
6. 10 LET a$="Wall"
    20 LET b$="ing"
    30 LET c$="ton"
    40 PRINT a$+b$+c$
```

```
7. 10 LET a$="THE"
    20 LET b$="GREEDY"
    30 LET c$="BOY"
    40 LET d$="ATE"
    50 LET e$="MANY"
    60 LET f$="CREAM"
    70 LET g$="BUNS"
    80 LET h$="AND"
    90 LET i$="JAM"
    100 LET j$="TARTS"
    110 PRINT a$;" ";c$;" ";d$;" ";g$
    120 PRINT a$;" ";b$;" ";c$;" ";d$;" ";e$;" ";j$
    130 PRINT a$;" ";c$;" ";d$;" ";f$;" ";j$
```

Unit 10

```
1. 30 PRINT b$;" ";p$
```

```

2. 60 PRINT p$(5 TO 6)+b$(4)
    70 PRINT p$(1 TO 2)+b$(5)
    80 PRINT b$(2)+p$(3)+b$(4)
    90 PRINT p$(1)+b$(2)+b$(5)
    100 PRINT b$(5)+b$(4)+p$(1 TO 3)
)

3. 10 LET a$ = "ladle"
    20 LET b$ = "spoon"
    30 PRINT b$(2)+a$(2 TO 3)+a$(3 TO 5)
    40 PRINT a$(1 TO 2)+b$(2)

```

Unit 11

```

1. a) 3+7 d) 5*2+8 g) 5*6*7
    b) 3*7 e) 24-4*3 h) 81-27/2
    c) 8/4 f) 30/3+2 i) 23/2*9

```

```

2. a) 17 c) 100 e) 5
    b) 200 d) 25 f) 3

3. a) 26 b) 13 c) 8 d) 1

```

```

4. a) 14 c) 4 e) 15
    b) 20 d) 24 f) 12

```

```

5. a) 17 c) 50 e) 7
    b) 3 d) 5 f) 20

```

```

6. a)

```

A	B	C
12		
12	5	
12	5	60
55	5	60

A	B	C
20		
20	60	
20	60	5
20	100	5

```

7. 10 LET a=7 } (any numbers here)
    20 LET b=3 }
    30 PRINT a+b
    40 PRINT a-b
    50 PRINT a*b
    60 PRINT a/b

```

```

8. 10 LET l=35
    20 LET b=16
    30 PRINT "The area of the rect
angle is ";l*b;" sq cm"
    40 PRINT "The perimeter of the
rectangle is ";l+l+b+b

```

```

9. 10 LET x$="Mr Blake"
    20 LET b=93
    30 LET c=9
    40 PRINT "A bag of cement cost
s ";b;" pence"
    50 PRINT x$;" ordered ";c;" ba
gs"
    60 PRINT "The cost of ";c;" ba
gs is ";b*c;" pence"

```

```

10. 10 LET a$="Susan Smith"
    20 LET b$="John Brown"
    30 LET c$="Mary Turner"
    40 LET a=43
    50 LET b=68
    60 LET c=73
    70 PRINT a$;" gained ";a;" mar
ks"
    80 PRINT b$;" gained ";b;" mar
ks"
    90 PRINT c$;" gained ";c;" mar
ks"
    100 PRINT "The average mark was
";(a+b+c)/3

```

```

11. 10 LET s=65
    20 LET t=6
    30 PRINT "Speed of car is ";s;
" miles per hour"
    40 PRINT "Time spent travellin
g is ";t;" hours"
    50 PRINT "Length of journey is
";s*t;" miles"

```

```

12. 10 LET a=21
    20 LET w=10
    30 LET h=160
    40 LET s=5
    50 PRINT "I am ";a;" years old
and I weigh ";w;" stone.I am ";
h;" centimetres tall and take si
ze ";s;" shoes"

```

```

13. 10 LET a=21 } ages of
    20 LET j=26 } children
    30 LET k=43 }
    40 LET m=65 } mother
    50 LET f=72 } father
    60 PRINT "The sum of the ages
of my family is ";a+j+k+m+f
    70 PRINT "The average age of m
y family is ";(a+j+k+m+f)/5

```

```

14. 10 LET l=1154
    20 LET j=19
    30 PRINT "The amount of sand c
arried by a lorry in four weeks
is ";l*j;" kilograms"
    40 PRINT "The amount carried i
n 48 weeks is ";l*j*48/4;" kilog
rams"

```

```

15. 10 LET b=44
    20 LET m=22
    30 LET ba=104
    40 PRINT "The price of three l
oaves is ";3*b;" pence"
    50 PRINT "The price of 5 pints
of milk is ";5*m;" pence"
    60 PRINT "The price of half a
pound of bacon is ";ba/2;" pence"
    70 PRINT "The total cost of th
e purchases is ";3*b+5*m+ba/2;"
pence"

```

Unit 12

```

1. 10 PRINT "
20 PRINT "DAY";TAB 10;"TEMPERA
TURE";TAB 23;"RAINFALL"
30 PRINT TAB 10;"DEGREES C";TAB
B 23;"INCHES"
40 PRINT "
50 PRINT "Monday";TAB 13;"10";
TAB 25;"0.2"
60 PRINT "Tuesday";TAB 13;"11"
;TAB 25;"0.4"
70 PRINT "Wednesday";TAB 13;"1"
3";TAB 25;"0.6"
80 PRINT "Thursday";TAB 13;"11"
;TAB 25;"1.0"
90 PRINT "Friday";TAB 13;"6";TAB
AB 25;"0"

```

Unit 14

```

1. 10 REM Numbers
    20 INPUT "Enter a number ";n
    30 PRINT n
    40 INPUT "Enter a second numbe
r ";n2
    50 PRINT n2
    60 INPUT "Enter a third number
";n3
    70 PRINT n3
    80 INPUT "Enter a fourth numbe
r ";n4
    90 PRINT n4
    100 LET s=n+n2+n3+n4
    110 PRINT "Sum of numbers=";s
    120 LET a=s/4
    130 PRINT "Average of numbers="
;a
    140 STOP

```

```

2. 10 REM Letter for interview
    20 INPUT "Enter name of interv
iewee ";n$
    30 INPUT "Enter date of interv
iew ";d$
    40 INPUT "Enter time of interv
iew ";t$
    50 INPUT "Enter place of inter
view ";p$
    60 INPUT "Enter name of interv
iewer ";m$
    70 INPUT "Enter the name of th
e person who is sending this let
ter ";s$
    80 INPUT "Enter date of letter
";l$
    90 PRINT TAB 18;"Cogland Ltd"
    100 PRINT TAB 20;"Machine Rd"

```



```

110 PRINT TAB 22;"Manchester"
120 PRINT TAB 18;l$
130 PRINT "Dear ";n$
140 PRINT TAB 15;"Thank you for
your application for the vacanc
y at our office.Please would you
attend an interview on ";d$;" a
t ";t$;
150 PRINT " The interview will
take place at ";p$;" and will be
conducted by ";m$
160 PRINT TAB 15;"Yours sincere
ly,"
170 PRINT TAB 18;s$
180 STOP

```

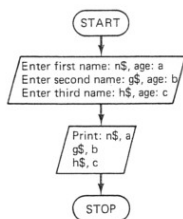
```

3.
10 REM Saving money
20 INPUT "Enter the amount of
money you wish to save ";s
30 INPUT "Enter the amount you
can save each week ";w
40 LET n=s/w
50 PRINT "The number of weeks
it will take to save £";s;" is "
;n
60 PRINT "The number of months
it will take to save £";s;" is
";n/4
70 STOP

```

Unit 15

1. Flowchart



```

10 REM Names and ages
20 INPUT "Name of first girl "
;n$
30 INPUT "Age of first girl ";
a
40 INPUT "Name of second girl
";g$
50 INPUT "Age of second girl "
;b
60 INPUT "Name of third girl "
;h$

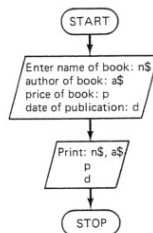
```

```

70 INPUT "Age of third girl ";
c
80 PRINT "NAME", "AGE"
90 PRINT n$,a
100 PRINT g$,b
110 PRINT h$,c
120 STOP

```

2. Flowchart



```

10 REM Book and Author
20 INPUT "Name of book? ";n$
30 INPUT "Author of book? ";a$
40 INPUT "Price of book? ";p
50 INPUT "Date of publication?
";d
60 PRINT n$;" by ";a$
70 PRINT "Price of book £";p
80 PRINT "Date of publication
";d
90 STOP

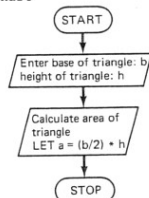
```

```

3. a) Variables:
b = base of triangle
h = height of triangle
a = area of triangle

```

Flowchart



```

10 REM Area of triangles
20 INPUT "Enter base of trian
gle ";b
30 INPUT "Enter height of tria
ngle ";h
40 LET a=b/2*h
50 PRINT "The area of the tria
ngle is ";a;" sq cm"
60 STOP

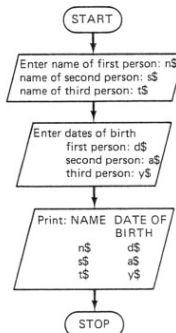
```

```

3. b) Variables:
n$ = name of first person
s$ = name of second person
t$ = name of third person
d$ = first date of birth
a$ = second date of birth
y$ = third date of birth

```

Flowchart



```

10 REM Names and dates of birt
h
20 INPUT "Name of first person
";n$
30 INPUT "Name of second perso
n ";s$
40 INPUT "Name of third person
";t$
50 INPUT "Date of birth of fir
st person ";d$
60 INPUT "Date of birth of sec
ond person ";a$
70 INPUT "Date of birth of thi
rd person ";y$
80 PRINT "NAME", "DATE OF BIRTH
"
90 PRINT n$,d$
100 PRINT s$,a$
110 PRINT t$,y$
120 STOP

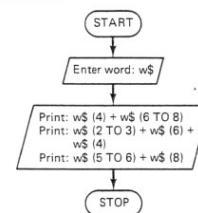
```

```

3. c) Variable:
w$ = word of at least 7 letters
(elephant)

```

Flowchart



```

10 REM Words
20 INPUT "Enter a word of at l
east seven letters ";w$
30 PRINT w$(4)+w$(6 TO 8)
40 PRINT w$(2 TO 3)+w$(6)+w$(4)
50 PRINT w$(5 TO 6)+w$(8)
60 STOP

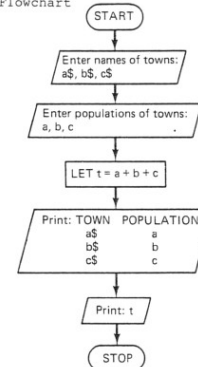
```

```

3. d) Variables:
a$ = name of first town
b$ = name of second town
c$ = name of third town
a = population of first town
b = population of second town
c = population of third town
t = total population

```

Flowchart



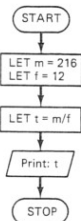
```

10 REM Towns and populations
20 INPUT "Name of first town "
; a$
30 INPUT "Name of second town "
; b$
40 INPUT "Name of third town "
; c$
50 INPUT "Population of first town "
; a
60 INPUT "Population of second town "
; b
70 INPUT "Population of third town "
; c
80 LET t=a+b+c
90 PRINT "TOWN", "POPULATION"
100 PRINT a$, a
110 PRINT b$, b
120 PRINT c$, c
130 PRINT
140 PRINT "The total population is "
; t
150 STOP

```

3. e) Variables:
 m = number of marbles
 f = number of friends
 t = total of marbles each friend receives

Flowchart



```

10 REM Sharing marbles
20 LET m=216
30 LET f=12
40 LET t=m/f
50 PRINT "The number of marble s that each friend receives is "
; t
60 STOP

```

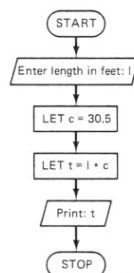
3. f) Variables:
 l = length in feet
 c = number of centimetres in a foot
 t = length in centimetres

```

10 REM Feet into centimetres
20 INPUT "Enter length in feet "
; l
30 LET c=30.5
40 LET t=l*c
50 PRINT l;" feet is ";t;" cm"
60 STOP

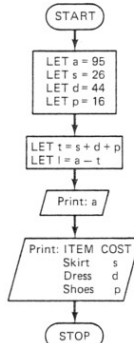
```

Flowchart



3. g) Variables:
 a = amount to spend
 s = price of skirt
 d = price of dress
 p = price of shoes
 t = amount spent
 l = amount left

Flowchart



```

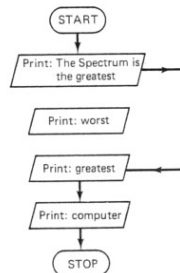
10 REM Girl's shopping
20 LET a=95
30 LET s=26
40 LET d=44
50 LET p=16
60 LET t=s+d+p
70 LET l=a-t
80 PRINT "Amount of money to s pend="
; a
90 PRINT
100 PRINT "ITEM", "COST"
110 PRINT "skirt", s
120 PRINT "dress", d
130 PRINT "shoes", p
140 PRINT
150 PRINT "Total cost of items="
; t
160 PRINT
170 PRINT
180 PRINT "Amount of money left "
; l
190 STOP

```

Unit 16

1.

Flowchart



```

2.
10 REM Adding numbers
20 INPUT "Enter a number "
; n
30 INPUT "Enter another number "
; n2
40 LET t=n+n2
50 PRINT n;"+";n2;"=";t
60 GO TO 20

3.
10 REM Savings account
20 LET s1=381.20

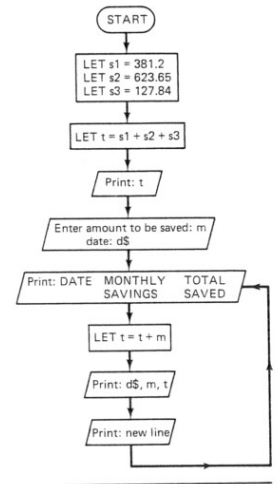
```

```

30 LET s2=623.65
40 LET s3=127.84
50 LET t=s1+s2+s3
60 PRINT "Total saved and tran sferred to one account is £"
; t
70 INPUT "Enter amount to be s aved this month "
; m
80 INPUT "Enter date "
; d$
90 PRINT "DATE";TAB 12;"MONTHLY Y";TAB 23;"TOTAL"
100 PRINT TAB 12;"SAVINGS";TAB 23;"SAVED"
110 LET t=t+m
120 PRINT d$;TAB 12;m;TAB 23;t
130 PRINT
140 GO TO 70

```

Flowchart



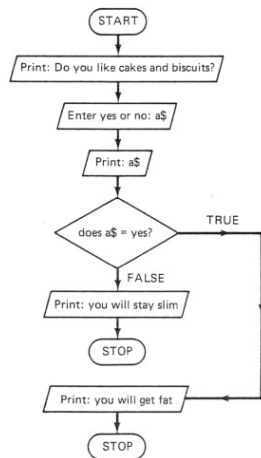
Unit 17

Question.
 Why was it possible to use the command PRINT in the middle of lines 40 & 50?
 Why did not the letter 'p' appear?

Answer.
After the word THEN the cursor automatically changes from an L to a K, so the computer automatically goes back into keyword mode - which means that another keyword, such as PRINT, can be entered.

1.

Flowchart

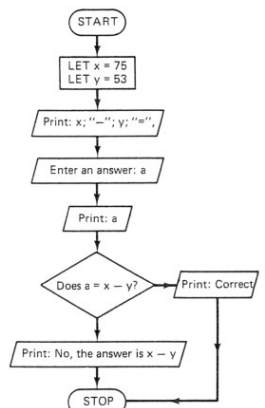


2.

```

10 REM Guessing my age
20 PRINT "What is my age?"
30 INPUT "Enter a number ";a
40 PRINT a
50 IF a=21 THEN GO TO 80
60 PRINT "Wrong"
70 STOP
80 PRINT "Correct"
90 STOP
  
```

3.



```

10 REM Subtracting numbers
20 LET x=75
30 LET y=53
40 PRINT x;"-";y;"=";
50 INPUT "Enter an answer ";a
60 PRINT a
70 IF a=x-y THEN PRINT "Correct"
80 PRINT "No, the answer is ";x-y
90 STOP
  
```

4.

```

10 REM Colour of my eyes
20 LET b$="blue"
30 INPUT "Guess the colour of my eyes ";e$
40 IF b$=e$ THEN PRINT "Yes, you are right": GO TO 60
50 PRINT "No, you are wrong"
60 STOP
  
```

5.

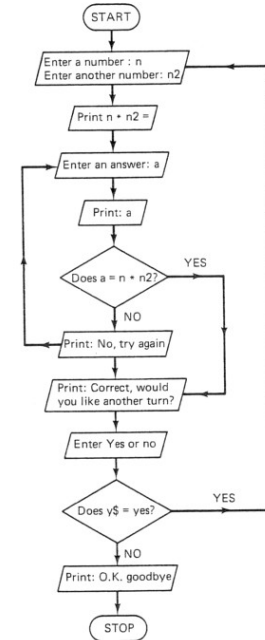
```

10 REM Names of towns
20 PRINT "TOWNS IN GT.BRITAIN"
30 INPUT "Enter the name of a town in Gt.Britain ";t$
40 IF t$="Timbuktu" THEN GO TO 70
50 PRINT t$
60 GO TO 30
70 STOP
  
```

6.

```

10 REM Testing multiplication tables
20 INPUT "Enter a number between 2 and 12 ";n
30 INPUT "Enter a second number between 2 and 12 ";n2
40 PRINT n;"*";n2;"=";
50 INPUT "Enter an answer ";a
60 PRINT a
70 IF a=n*n2 THEN GO TO 100
80 PRINT "No, try again"
90 GO TO 50
100 PRINT "Correct, would you like another turn?"
110 INPUT "Enter yes or no ";y$
120 IF y$="yes" THEN GO TO 20
130 PRINT "O.K.-goodbye"
140 STOP
  
```



Unit 18

1.
a) no f) yes k) yes p) yes
b) yes g) no l) no c) no
c) yes h) yes m) yes r) no
d) yes i) no n) yes s) no
e) yes j) yes o) no t) yes

2.

a) 16 < 23
b) 14 > 10
c) 10 <= 10 or 10 >= 10
d) 5 + 6 > 11 - 3
e) 3 * 7 >= 13 + 8
or
3 * 7 <= 13 + 8
f) x < y
g) x / 2 < y + 4
h) x * x > y / 2
i) (a+b) * 3 > a + b * 3
j) a + b - c > a + c - 10

3.

```

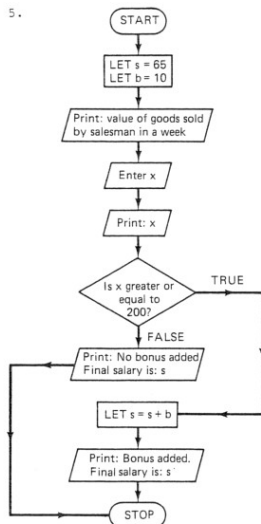
10 REM Which number is bigger
20 INPUT "Enter a number ";x
30 INPUT "Enter a second number ";y
40 IF x>y THEN GO TO 70
50 PRINT y;" is bigger than ";x
60 STOP
70 PRINT x;" is bigger than ";y
80 STOP
  
```

4.

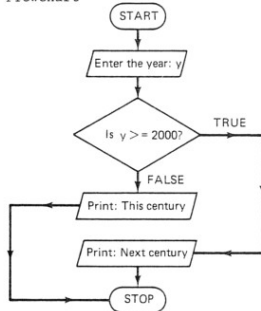
```

10 REM if x is greater or equal to 10
20 INPUT "Enter a number ";x
30 IF x>=10 THEN GO TO 70
40 LET y=x+10
50 PRINT y
60 STOP
70 LET w=x-10
80 PRINT w
90 STOP
  
```

5.



6. Variable:
y = year to be entered
Flowchart

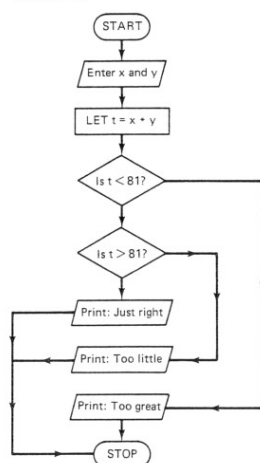


```

10 REM Which century
20 INPUT "Enter the number of
the year ";y
30 IF y>=2000 THEN GO TO 60
40 PRINT "This century"
50 GO TO 70
60 PRINT "Next century"
70 STOP
  
```

7. Variables:
x = one number
y = second number
t = x * y

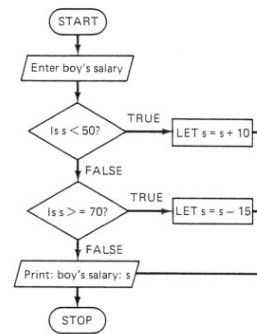
Flowchart



```

10 REM Greater or less than 81
20 INPUT "Enter a number ";x
30 INPUT "Enter a second numbe
r ";y
40 LET t=x*y
50 IF t>81 THEN GO TO 110
60 IF t<81 THEN GO TO 90
70 PRINT "Just right"
80 GO TO 120
90 PRINT "Too little"
100 GO TO 120
110 PRINT "Too great"
120 STOP
  
```

8. Variable:
s = boy's salary
Flowchart

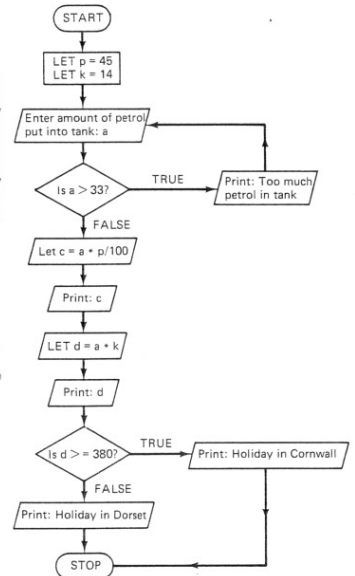


```

10 REM Boy's salary
20 INPUT "Enter boy's salary "
;s
30 IF s<50 THEN LET s=s+10: GO
TO 50
40 IF s>=70 THEN LET s=s-15
50 PRINT "Boy's weekly salary
is £";s
60 STOP
  
```

9. Variables:
p = price of petrol
k = distance travelled on
1 litre of petrol
d = distance travelled
a = amount of petrol
c = cost of petrol

Flowchart



```

10 REM Distance travelled
20 LET p=45
30 LET k=14
40 INPUT "Enter amount of petr
ol put into tank ";a
50 IF a>33 THEN PRINT "Too muc
h petrol in tank": GO TO 40
60 LET c=a*p
70 PRINT "The cost of petrol b
ought is £";c/100
80 LET d=a*k
  
```

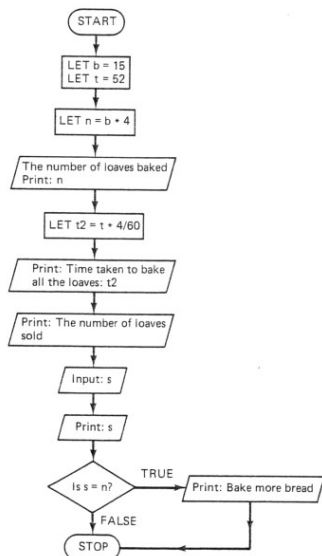
```

90 PRINT "The distance that ca
n be travelled on ";a;" litres o
f petrol is ";d;" kilometres"
100 IF d>=380 THEN PRINT "The m
an takes his holiday in Cornwall
"; GO TO 120
110 PRINT "The man takes his ho
liday in Dorset"
120 STOP

```

10. Variables:
b = number of loaves held
by oven
t = time taken to bake 15 loaves
n = number of loaves baked in
a day
t2 = time taken to bake all
loaves
s = number of loaves sold

Flowchart



```

10 REM Baking bread
20 LET b=15
30 LET t=52
40 LET n=b*4
50 PRINT "The number of loaves
baked is ";n
60 LET t2=t*4/60
70 PRINT "Time taken to bake a
ll the loaves is ";t2;" hours"
80 PRINT "The number of loaves
sold is ";
90 INPUT "Enter number ";s
100 PRINT s
110 IF s=n THEN PRINT "Bake mor
e bread"
120 STOP

```

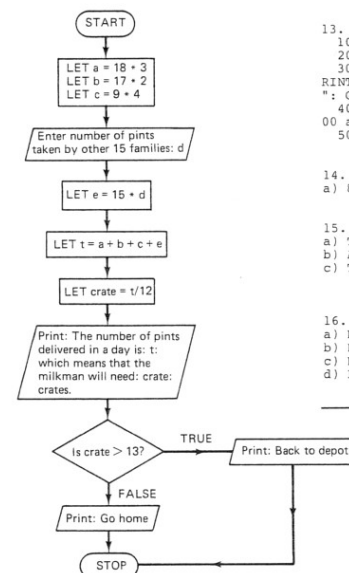
11. Variables:
a = number of pints to the 18
families which take 3 pints a
day
b = number of pints to the 17
families which take 2 pints a
day
c = number of pints to the 9
families which take 4 pints a
day
d = number of pints taken by the
other 15 families
e = number of pints taken by the
other 15 families at 'd' pints a
day
crate = no. of crates required

```

10 REM Milkman's round
20 LET a=18*3
30 LET b=17*2
40 LET c=9*4
50 INPUT "Enter number of pint
s taken by other 15 families ";d
60 LET e=15*d
70 LET t=a+b+c+e
80 LET crate=t/12
90 PRINT "The number of pints
delivered in a day is ";t;" whic
h means that the milkman will ne
ed ";crate;" crates"
100 IF crate>13 THEN PRINT "Bac
k to depot"; GO TO 120
110 PRINT "Go home"
120 STOP

```

Flowchart



12. a) 30 b) 40 c) 140 d) 30
50 400 100

13. 10 REM Number compared to 100
20 INPUT "Enter a number ";n
30 IF n>=100 AND n<=200 THEN P
RINT "Number between 100 and 200
"; GO TO 50
40 PRINT "Number not between 1
00 and 200"
50 STOP

14. a) 80 b) 180 c) 160

15. a) Teenager d) Child
b) Adult e) Teenager
c) Teenager

16. a) No walk today
b) No walk today
c) No walk today
d) Let's go for a walk

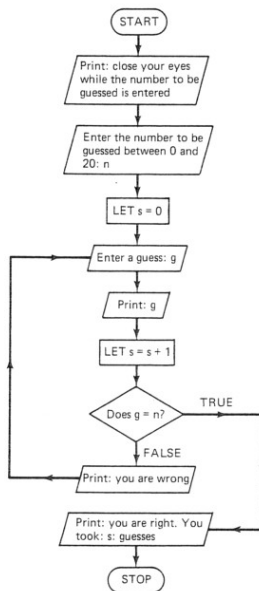
Unit 19

```

1. 10 REM Flowers
    20 PRINT "FLOWERS"
    30 LET c=0
    40 INPUT "Enter the name of a
flower ";f$
    50 PRINT f$
    60 LET c=c+1
    70 IF c=6 THEN GO TO 90
    80 GO TO 40
    90 STOP

```

3. Flowchart



```

2. 10 REM Flowers
    20 PRINT "FLOWERS"
    30 LET c=0
    35 INPUT "Enter the number of
flowers to be listed ";n
    40 INPUT "Enter the name of a
flower ";f$
    50 PRINT f$
    60 LET c=c+1
    70 IF c=n THEN GO TO 90
    80 GO TO 40
    90 STOP

```

```

10 REM Guessing a number
20 PRINT "CLOSE YOUR EYE WHILE
THE NUMBER TO BE GUESSED IS ENTERED"
30 INPUT "Enter a number between
0 and 20 ";n
40 LET s=0
50 INPUT "Enter a guess between
0 and 20 ";g
60 LET s=s+1
70 PRINT g
80 IF g=n THEN GO TO 110
90 PRINT "You are wrong"
100 GO TO 50
110 PRINT "You are right. You took
ok ";s;" guesses"
120 STOP

```

Unit 20

```

1. a) 1 b) 4 c) Goodbye
    2 5 Goodbye
    3 6 Goodbye
    4 7 Goodbye
    5 8
    6 9
    7 10 d) 1 This is a loop
    8 11 2 This is a loop
    9 12 3 This is a loop
    10 13
    11 14
    12 15 This is a loop
    16

2. a) 2 b) 15 c) 0 d) 50 e) -2
    4 19 20 25 -5
    6 23 40 0 -8
    8 60 -25 -11
    10 80
    12 100

```

```

f)-10
Stepping up
-5
Stepping up
0
Stepping up
5
Stepping up
10
Stepping up
15
Stepping up
20
Stepping up
25
Stepping up

```

```

3. a) 10 FOR x = 1 TO 10
    20 PRINT x
    30 NEXT x

```

```

b) 10 FOR x = 10 TO 1 STEP -1
    20 PRINT x
    30 NEXT x

```

```

c) 10 FOR x = 1 TO 6 STEP 2
    20 PRINT x
    30 NEXT x

```

```

d) 10 FOR x = n TO 10
    20 PRINT x
    30 NEXT x

```

```

e) 10 FOR x = 1 TO n
    20 PRINT x
    30 NEXT x

```

```

f) 10 FOR x = 1 TO 10
    20 PRINT x
    30 NEXT x

```

```

4. a) 2 6 b) 10 12
    3 7 12 14
    4 8 14 16
    16

```

```

c) 12 10
    9 10
    6 10
    3 10

```

```

d) 100 90 110
    125 115 135
    150 140 160
    175 165 185
    200 190 210

```

```

5. 10 FOR x=13 TO 23
    20 PRINT x
    30 NEXT x

```

```

6. 10 FOR x=3 TO 37 STEP 2
    20 PRINT x
    30 NEXT x

```

```

7. 10 FOR x=5 TO 35 STEP 5
    20 PRINT x
    30 NEXT x

```

```

8. 10 FOR x=10 TO -6 STEP -2
    20 PRINT x
    30 NEXT x

```

```

9. 10 REM 7 TIMES TABLE
    20 FOR x=1 TO 12
    30 PRINT x;"*";7;"=";x*7
    40 NEXT x

```

```

10. 10 PRINT "I'm on the top line"
    20 FOR x=1 TO 5
    30 PRINT
    40 NEXT x
    50 PRINT "I'm five lines further down"
    60 FOR x=1 TO 5
    70 PRINT
    80 NEXT x
    90 PRINT "I'm another five lines further down"
    100 STOP

```

```

11. 10 FOR x=1 TO 8
    20 PRINT "Spectrum"
    30 NEXT x

```

```

12. 10 REM Leap years
    20 FOR x=1904 TO 1984 STEP 4
    30 PRINT x
    40 NEXT x
    50 STOP

```

```

13. 10 FOR x=10 TO 20
    20 PRINT x,x+3
    30 NEXT x

```

```

14.
10 REM Names and ages
20 PRINT "NAME", "AGE"
30 FOR x=1 TO 6
40 INPUT "Enter a name "; n$
50 INPUT "Enter that person's
age "; a
60 PRINT n$, a
70 NEXT x
80 STOP

```

```

15.
10 REM Names and prices of car
s
20 PRINT "NAME OF CAR", "PRICE
IN £ "s"
30 LET t=0
40 FOR x=1 TO 4
50 INPUT "Enter name of car ";
c$
60 INPUT "Enter the price of t
hat car "; p
70 PRINT c$, p
80 LET t=t+p
90 NEXT x
100 PRINT "Total price of the c
ars is £"; t
110 PRINT "Average price of car
s is £"; t/4
120 STOP

```

```

16.
10 REM Natural population incr
ease
20 PRINT "COUNTRY"; TAB 9; "BIRT
H RATE"; TAB 21; "DEATH RATE"
30 FOR x=1 TO 7
40 INPUT "Enter name of countr
y "; c$
50 INPUT "Enter birth rate "; b
60 INPUT "Enter death rate "; d
70 PRINT c$; TAB 13; b; TAB 25; d
80 LET n=b-d
90 PRINT "The natural increase
per 1000 people is "; n
100 PRINT
110 NEXT x
120 STOP

```

Unit 21

```

1.
10 READ a,b,c
20 PRINT a,b,c
30 DATA 100,200,300

```

```

2.
10 READ x$,y$,z$
20 PRINT x$,y$,z$
30 DATA "cat","dog","fish"

```

```

3.
10 READ c,a$,b$
20 PRINT c;a$,b$
30 DATA 3,"blind","mice"

```

```

4.
10 READ x$,y,b$
20 PRINT y;b$;x$
30 DATA "bottles",10,"green"

```

```

5.
10 FOR n=1 TO 4
20 READ x
30 PRINT x
40 DATA 1,2,3,4
50 NEXT n

```

```

6.
10 FOR n=1 TO 3
20 READ x$
30 PRINT x$
40 DATA "man","woman","child"
50 NEXT n

```

```

7.
10 FOR n=1 TO 3
20 READ b,a$
30 PRINT b,a$
40 DATA 3,"ships",2,"cars",1,"
train"
50 NEXT n

```

```

8.
a$ = Tom b$ = Dick c$ = Harry

```

```

9.
a = 3 b$ = Norway

```

```

10.
x = 10 y = 3 z = 17

```

```

11.
10 REM Days of the week
20 FOR n=1 TO 7
30 READ d$
40 PRINT d$
50 DATA "Monday","Tuesday","We
dnesday","Thursday","Friday","Sa
turday","Sunday"
60 NEXT n
70 STOP

```

```

12.
10 REM List of numbers
20 FOR x=1 TO 10
30 READ n
40 PRINT n
50 DATA 35,42,57,39,48,64,44,5
0,61,54
60 NEXT x
70 STOP

```

```

13.
10 REM Wicked witch
20 READ a$,b$,c$,d$
30 READ e$,f$,g$
40 DATA "The","wicked","witch"
,"cast","many","magic","spells"
50 PRINT a$;" ";b$;" ";c$;" ";
d$;" ";e$;" ";f$;" ";g$
60 PRINT a$;" ";c$;" ";d$;" ";
b$;" ";g$
70 PRINT a$;" ";f$;" ";c$;" ";
d$;" ";e$;" ";g$
80 STOP

```

```

14.
10 REM Food and prices
20 READ l$,l$,s$,j$,j
30 READ b$,b$,e$,e$,r$
40 PRINT "FOOD","PRICE"
50 PRINT l$,l
60 PRINT s$,s
70 PRINT j$,j
80 PRINT b$,b
90 PRINT e$,e
100 PRINT r$,r
110 DATA "bread",36,"soup",19,"
jam",58,"beans",17,"eggs",42,"ri
ce",88
120 LET t=l+s+j+b+e+r
130 PRINT "The total cost of th
e food is £";t/100
140 STOP

```

Unit 22

```

1.
10 REM Stars
20 FOR n=1 TO 10
30 FOR x=1 TO 10
40 PRINT " ";
50 NEXT x
60 PRINT
70 NEXT n
80 STOP

```

```

2.
10 REM Saving accounts
20 PRINT "Name","Amount saved"
30 FOR n=1 TO 5
40 INPUT "Enter person's name
"; n$
50 INPUT "Enter amount saved "
"; s$
60 PRINT n$,s$
70 FOR r=6 TO 10 STEP 2
80 LET i=s*r/100
90 PRINT "The interest gained
at a rate of ";r;"% is £";i
100 PRINT "Total saved is £";s+
i
110 NEXT r
120 PRINT
130 NEXT n
140 STOP

```

```

3.
10 REM Rectangles of stars
20 FOR n=1 TO 5
30 INPUT "Enter length of rect
angle "; l
40 INPUT "Enter width of recta
ngle "; w
50 FOR x=1 TO l
60 FOR y=1 TO w
70 PRINT " ";
80 NEXT y
90 PRINT
100 NEXT x
110 PRINT
120 PRINT
130 NEXT n
140 STOP

```

Unit 23

```

2. a
10 REM Moving car
20 BORDER 6
30 PAPER 1
40 LET a=10
50 FOR b=0 TO 24
60 INK 7
70 PRINT AT a-2,b+1;" "
80 PRINT AT a-1,b;" "
90 PRINT AT a,b+1;" "
100 PRINT AT a,b+5;" "
110 PAUSE 5
120 NEXT b
130 STOP

```

```

2. b
10 REM
20 BORDER 6
40 LET a=10
45 FOR i=1 TO 6
47 INK i

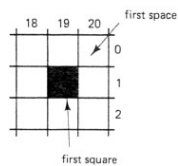
```

```

50 FOR b=0 TO 24
70 PRINT AT a-2,b+1;"■■■■"
80 PRINT AT a-1,b;"■■■■■■■■"
90 PRINT AT a,b+1;" O"
100 PRINT AT a,b+5;" O"
110 PAUSE 5
120 NEXT b
125 CLS - this is to remove car
127 NEXT i from right-hand of
130 STOP the screen

```

3.



```

4. 10 REM Square moves from botto
m right to top left
20 FOR n=20 TO 1 STEP -1
30 PRINT AT n,n;" "
40 PRINT AT n-1,n-1;"■"
50 PAUSE 10
60 NEXT n
70 STOP

```

```

5. 10 REM Moving a square
20 LET a=11
30 LET b=16
40 PRINT AT a-1,b;" "
50 PRINT AT a+1,b;" "
60 PRINT AT a,b-1;" "
70 PRINT AT a,b+1;" "
80 PRINT AT a,b;"■"
90 IF INKEY$="5" THEN LET b=b-
1
100 IF b<1 THEN LET b=1
110 IF INKEY$="6" THEN LET a=a+
1
120 IF a>20 THEN LET a=20
130 IF INKEY$="7" THEN LET a=a-
1
140 IF a<1 THEN LET a=1
150 IF INKEY$="8" THEN LET b=b+
1
160 IF b>30 THEN LET b=30
170 GO TO 40

```

Unit 24

```

1. a & b
10 REM Mountains
20 DIM a$(6,11)
30 DIM x(6)
40 DIM c$(6,12)
50 FOR n=1 TO 6
60 READ a$(n)
70 DATA "Everest","Kilimanjaro",
"Tablet","Blanc","Matterhorn",
Ben Nevis"
80 NEXT n
90 FOR m=1 TO 6
100 READ x(m)
110 DATA 8848,5895,1087,4807,44
78,1347
120 NEXT m
130 PRINT "MOUNTAIN","HEIGHT IN
METRES"
140 PRINT
150 FOR y=1 TO 6
160 PRINT a$(y),x(y)
170 NEXT y
180 FOR c=1 TO 6
190 READ c$(c)
200 DATA "Nepal-Tibet","Tanzani
a","South Africa","France","Swit
zerland","Scotland"
210 NEXT c
220 PRINT "MOUNTAIN","COUNTRY"
230 FOR w=1 TO 6
240 PRINT a$(w),c$(w)
250 NEXT w
260 STOP

```

```

2. 10 REM Reversing numbers
20 DIM x(8)
30 FOR n=1 TO 8
40 READ x(n)
50 DATA 10,20,30,40,50,60,70,8
0
60 NEXT n
70 FOR y=8 TO 1 STEP -1
80 PRINT x(y)
90 NEXT y
100 STOP

```

```

3. 10 REM Every other number
20 DIM x(12)
30 FOR n=1 TO 12
40 READ x(n)
50 DATA 5,10,15,20,25,30,35,40
,45,50,55,60
60 NEXT n
70 FOR y=1 TO 12 STEP 2
80 PRINT x(y)
90 NEXT y
100 STOP

```

```

4. a
10 REM Days of the week
20 DIM d$(7,9)
30 FOR n=1 TO 7
40 READ d$(n)
50 DATA "Monday","Tuesday","We
dnesday","Thursday","Friday","Sa
turday","Sunday"
60 NEXT n
70 INPUT "Enter a number betwe
en 1 and 7 ";n
80 PRINT "Day ";n;" is ";d$(n)
90 STOP

```

```

4. b - modified
10 REM Days of the week
20 DIM d$(7,9)
30 FOR n=1 TO 7
40 READ d$(n)
50 DATA "Monday","Tuesday","We
dnesday","Thursday","Friday","Sa
turday","Sunday"
60 NEXT n
70 INPUT "Enter a number betwe
en 1 and 7 ";n
80 IF n<1 THEN PRINT "Number t
oo low": GO TO 70
90 IF n>7 THEN PRINT "Number t
oo high": GO TO 70
100 PRINT "Day ";n;" is ";d$(n)
110 STOP

```

```

4. c
Add to above program:
65 FOR y=1 TO 4
105 NEXT y

```

```

5. 10 REM Names of months
20 DIM m$(12,9)
30 DIM w$(1,9)
40 FOR x=1 TO 12
50 READ m$(x)
60 DATA "January","February","
March","April","May","June","Jul
y","August","September","October",
"November","December"
70 NEXT x
80 INPUT "Enter a word ";w$(1)
90 FOR y=1 TO 12
100 IF m$(y)=w$(1) THEN PRINT m
$(y); " is month ";y: GO TO 130
110 NEXT y
120 PRINT w$(1); " is not a mont
h"
130 STOP

```

```

6. 10 REM How electricity is prod
uced
20 DIM e$(4,7)
30 DIM p(4)
40 FOR x=1 TO 4

```

```

50 READ e$(x),p(x)
60 DATA "coal",79,"oil",8,"nuc
lear",11,"water",2
70 NEXT x
80 PRINT "HOW ELECTRICITY IS P
RODUCED IN"
90 PRINT TAB 13;"THE U.K."
100 PRINT "SOURCE OF";TAB 16;"%
OF TOTAL"
110 PRINT TAB 3;"ELECTRICITY";T
AB 22;"PRODUCED"
120 PRINT "-----"

```

```

130 FOR y=1 TO 4
140 PRINT e$(y);TAB 21;p(y)
150 PRINT
160 NEXT y
170 STOP

```

```

7. a$(2,1) = Paul
a$(1,3) = Sarah
a$(2,4) = Mark

```

```

8. 10 REM Sportsmen
20 DIM s$(5,2,12)
30 PRINT "NAME","SPORT"
40 FOR x=1 TO 5
50 FOR y=1 TO 2
60 READ s$(x,y)
70 DATA "J.Connors","Tennis","
S.Cram","Athletics","S.Davis","S
nooker","L.Piggott","Horse Racin
g","E.Bruno","Boxing"
80 PRINT s$(x,y)
90 NEXT y
100 PRINT
110 NEXT x
120 STOP

```

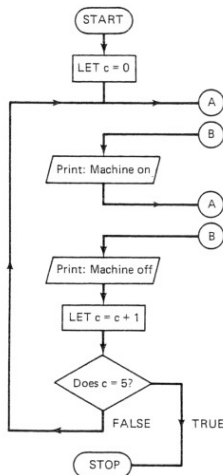
```

9. 10 REM World trade
20 DIM t(3,3)
30 PRINT "YEAR SHARE OF WORLD
TRADE IN $"
40 PRINT TAB 7;"DEVELOPED";TAB
19;"DEVELOPING"
50 PRINT TAB 7;"COUNTRIES";TAB
19;"COUNTRIES"
60 PRINT "-----"
70 FOR x=1 TO 3
80 FOR y=1 TO 3
90 READ t(y,x)
100 DATA 1950,68,32,1960,79,21,
1970,82,18
110 PRINT t(y,x); " "
120 NEXT y
130 PRINT
140 NEXT x
150 STOP

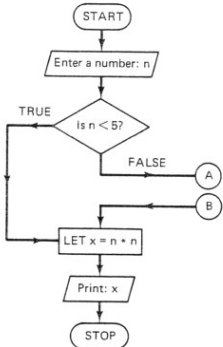
```


Unit 25

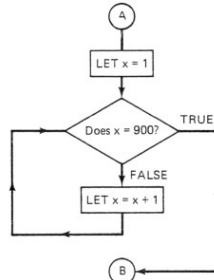
1. Flowchart



2. Flowchart



Subroutine



Subroutine



3. a) 49 b) 4
4. 10 REM Questions about your home
20 PRINT "How many rooms has your house? ";
30 INPUT "Enter a number ";n
40 PRINT n
50 GO SUB 200
60 PRINT "What is the number of your house? ";
70 INPUT "Enter a number ";x
80 PRINT x
90 GO SUB 200
100 PRINT "Does your house have a garden? ";
110 INPUT "Enter yes or no ";a\$
120 PRINT a\$
130 GO SUB 200
140 PRINT "Does your house have a garage? ";
150 INPUT "Enter yes or no ";b\$
160 PRINT b\$
170 GO SUB 200
180 STOP
200 REM Subroutine for stars
210 FOR n=0 TO 11
220 PRINT "*";
230 NEXT n
240 RETURN
5. 10 REM Ex25.5
20 INPUT "Enter a number between 1 and 100 ";n
30 IF n<50 THEN GO SUB 100
40 FOR x=0 TO n STEP 5
50 PRINT x
60 NEXT x
70 STOP
100 REM Subroutine
110 LET n=n+25
120 RETURN
6. 10 REM Guessing a number
20 PRINT "Player guessing number close your eyes"
30 INPUT "Enter a number to be guessed between 1 and 100 ";a
40 INPUT "Player enter a number between 1 and 100 ";n
50 PRINT n
60 IF n=a THEN PRINT "Correct"
70 GO TO 20
70 IF n>a THEN GO SUB 100
80 IF n<a THEN GO SUB 200
7. 10 REM OPENING BOTTLES
20 INPUT "Enter type of bottle top-cork or screw cap ";b\$
30 IF b\$="cork" THEN GO SUB 20
40 IF b\$="screw cap" THEN GO SUB 300
50 IF b\$<>"cork" AND b\$<>"screw cap" THEN PRINT "Wrong type of bottle top": GO TO 20
60 PRINT "Bottle opened. Shall I pour a glass?"
70 PRINT "Cheers!!"
80 PRINT
90 PRINT "Cheers!!"
100 STOP
200 REM Subroutine to open bottle with cork
210 PRINT "1. Place corkscrew on top of cork with left hand-hold firmly"
220 PRINT "2. With right hand turn handle of corkscrew clockwise pressing downwards"
230 PRINT "3. With right hand turn handle of corkscrew anti-clockwise pulling cork upwards"
240 RETURN
300 REM Subroutine to open bottle with screw cap
310 PRINT "1. Hold bottle with left hand"
320 PRINT "2. Grip bottle cap firmly with right hand"
330 PRINT "3. Turn cap anti-clockwise"
340 RETURN
8. 10 REM Painting walls
20 LET r=0.15
30 INPUT "Enter the number of walls to be painted ";n
40 LET t=0
50 FOR w=1 TO n
60 INPUT "Enter length of wall ";f
70 GO SUB 300
80 LET l=f
90 INPUT "Enter height of wall ";h
100 GO SUB 300
110 LET h=h
120 LET a=l*h

```

130 PRINT "Area of wall is ";a
140 LET t=t+a
150 PRINT "Total area =";t
160 PRINT
170 NEXT w
180 LET p=t*f
190 PRINT "Amount of paint requ
ired is ";p;" litres"
200 STOP
300 REM Subroutine for changing
feet into metres
310 LET m=f/3.208
320 RETURN

```

9. Distances between cities

Line 20: Sets up a one-dimensional string array called `t$` which has seven elements, the maximum length of any element being ten characters. The data is supplied by lines 50 to 80. This array stores the names of the cities.

Line 30: Sets up a two-dimensional numeric array called `d` which has 49 elements - 7 rows of 7 columns. The nested loops (Lines 90 to 140) put the distances between the cities into this array.

Line 40: Sets up a one-dimensional string array with one element of maximum length ten characters, which matches the length of the `t$` array. The information for this array is entered by the user in lines 150 and 190.

Line 150: City travelled to is entered into `c$(1)` and the computer then goes to the subroutine.

Lines 300-350: A subroutine compares the city entered into `c$(1)` with all the cities contained in `t$()`. If `c$(1)` matches up with one of the cities in `t$()` then the computer returns to the main program. If no match is found, then the computer sets the value of `x` to 0 and returns to the main program.

Line 170: Returning from the subroutine if `x = 0` then the city did not match, so "No information" is printed and the user is sent back to line 150 where another city can be entered.

Line 180: If a match was found, then `x` will have a value of between 1 and 7, depending on which element of `t$()` the match was found in. This value of `x` is given to `y` to be stored for later use, as `x` is used again and will therefore alter its value.

Line 190: The city travelled from is entered into `c$(1)`; the computer then goes to the subroutine where the process of checking the name of the city is done again. (For information, see lines 300-350.)

Line 210: This is almost the same as line 170; if `x = 0` there is no information about the city in `c$(1)`, so the user is sent back to line 190 for another city to be entered.

Line 220: The distance between the cities is now printed out. This is done by looking up the names in the `t$()` array. One city is stored in `t$(y)` - see line 180 - and the other in `t$(x)`. The numbers contained in `y` and `x` are then used to look up the distance in the `d` array. For example, if `y = 5` and `x = 3` [`t$(5)` is Liverpool and `t$(3)` is Plymouth] the element `d(5,3)` is looked up, giving a value of 278, as can be seen from the following table.

	1 L'DR	2 T'TN	3 P'MTH	4 N'HAM	5 L'PL	6 LEEDS	7 HULL
1 LONDON	0	144	211	122	197	190	168
2 TAUNTON	144	0	74	180	203	237	247
3 PLYMOUTH	211	74	0	254	278	312	321
4 NOTTINGHAM	122	180	254	0	97	67	73
5 LIVERPOOL	197	203	278	97	0	73	128
6 LEEDS	190	237	312	67	73	0	55
7 HULL	168	247	321	73	128	55	0

Unit 26

1. Top E = 16 Bottom B = -13

2. Add:
15 FOR n = 1 TO 3
25 NEXT n

3. Add:
22 PAUSE 3 * 50

4. a
10 REM Scale of C major all in
semiquavers
20 BEEP .25,0: BEEP .25,2: BEE
P .25,4: BEEP .25,5: BEEP .25,7:
BEEP .25,9: BEEP .25,11: BEEP .
25,12
30 STOP

b
10 REM Scale of C major with c
rotchets and quavers
20 BEEP .5,0: BEEP .5,2: BEEP
1,4: BEEP .5,5: BEEP .5,7: BEEP
1,9: BEEP .5,11: BEEP .5,12
30 STOP

c
10 REM Scale of C major with
a variety of different length
notes
20 BEEP 2,0: BEEP .25,2: BEEP
.25,4: BEEP .25,5: BEEP .25,7:
BEEP .5,9: BEEP .5,11: BEEP 4,12
30 STOP

5.
10 REM London's Burning
20 FOR x=1 TO 2
30 BEEP .5,2: BEEP .5,2: BEEP
1,7: BEEP 1,7
40 NEXT x
50 FOR y=1 TO 2
60 BEEP .5,9: BEEP .5,9: BEEP
1,11: BEEP 1,11
70 NEXT y
80 FOR w=1 TO 2
90 BEEP 1,14: BEEP 2,14
100 NEXT w
110 FOR z=1 TO 2
120 BEEP .5,14: BEEP .5,12: BEE
P 1,11: BEEP 1,11
130 NEXT z
140 STOP

6. a
10 REM Scale of G major
20 BEEP .5,-5: BEEP .5,-3: BEE
P .5,-1: BEEP .5,0: BEEP .5,2: B
EEP .5,4: BEEP .5,6: BEEP .5,7
30 STOP

b
G A B C D E F# G
7 9 11 12 14 16 18 19

c
10 REM Scale of G major
20 BEEP .5,-5: BEEP .5,-3: BEE
P .5,-1: BEEP .5,0: BEEP .5,2: B
EEP .5,4: BEEP .5,6: BEEP .5,7
30 BEEP .5,9: BEEP .5,11: BEEP
.5,12: BEEP .5,14: BEEP .5,16:
BEEP .5,18: BEEP .5,19
40 STOP

```

7.
10 REM Good King Wenceslas
20 FOR n=1 TO 2
30 BEEP 1,7: BEEP 1,7: BEEP 1,7
7: BEEP 1,9: BEEP 1,7: BEEP 1,7:
BEEP 2,2: BEEP 1,4: BEEP 1,2: B
EEP 1,4: BEEP 1,6: BEEP 2,7: BEE
P 2,7
40 NEXT n
50 GO SUB 200
60 BEEP 1,11: BEEP 1,2: BEEP 2
,7: BEEP 1,4: BEEP 1,2: BEEP 1,4
: BEEP 1,6: BEEP 2,7: BEEP 2,7
70 BEEP 1,2: BEEP 1,2: BEEP 1,
4: BEEP 1,6: BEEP 1,7: BEEP 1,7:
BEEP 2,9
80 GO SUB 200
90 BEEP 2,7: BEEP 2,12: BEEP 4
,7
100 STOP
200 REM Subroutine
210 BEEP 1,14: BEEP 1,12: BEEP
1,11: BEEP 1,9
220 RETURN

```

Unit 27

```

1. a 4 b 103 c 99
d -6 e -88 f -1
g 0

```

2.
Line 30: LET X = INT(21*RND) was not written as LET X = INT(20*RND) because INT rounds down; 21 will automatically give numbers only up to 20, which is what is required.

```

3. Add:
65 INPUT "Enter number you wish to be counted ";n
Alt:
70 IF d= n THEN LET c = c + 1
120 PRINT "The number of times ";n;" appeared was ";c

```

```

4.
10 REM List of 20 numbers
20 FOR n=1 TO 20
30 LET x=INT (101*RND)
40 PRINT x
50 NEXT n
60 STOP

```

```

5.
10 REM List of 15 numbers
20 FOR n=1 TO 15
30 LET x=INT (20*RND)+1
40 PRINT x
50 NEXT n
60 STOP

```

```

6.
10 REM Same sequence of random numbers
20 RANDOMIZE 600
30 FOR n=1 TO 10
40 LET x=INT (11*RND)
50 PRINT x
60 NEXT n
70 STOP

```

```

7.
10 REM How many times numbers between 1 and 5 occur in a 100 random numbers
20 DIM a(5)
30 FOR y=1 TO 5
40 READ a(y)
50 DATA 0,0,0,0,0
60 NEXT y
70 FOR n=1 TO 10
80 FOR m=1 TO 10
90 LET s=INT (5*RND)+1
100 PRINT s;" ";
110 FOR x=1 TO 5
120 IF s=x THEN LET a(x)=a(x)+1
130 NEXT x
140 NEXT m
150 PRINT

```

```

160 NEXT n
170 PRINT
180 PRINT "NUMBER","APPEARED"
190 FOR w=1 TO 5
200 PRINT w,a(w)
210 NEXT w
220 STOP

```

```

8.
10 REM Testing multiplication tables
20 LET x=INT (11*RND)+2
30 LET y=INT (11*RND)+2
40 PRINT x;"*";y;"=";
50 INPUT "Enter an answer ";a
60 PRINT a
70 IF a=x*y THEN PRINT "Correct": GO TO 90
80 PRINT "Incorrect,the answer is ";x*y
90 STOP

```

```

9. a
10 REM Throwing 2 dice
20 LET c=0
30 FOR n=1 TO 50
40 LET d1=INT (6*RND)+1
50 LET d2=INT (6*RND)+1
60 LET s=d1+d2
70 IF s=7 THEN LET c=c+1
80 NEXT n
90 PRINT "The number of times 7 occurred is ";c
100 STOP

```

```

b
10 REM Throwing 2 dice
20 LET t=0
30 FOR x=1 TO 5
40 LET c=0
50 FOR n=1 TO 50
60 LET d1=INT (6*RND)+1
70 LET d2=INT (6*RND)+1
80 LET s=d1+d2
90 IF s=7 THEN LET c=c+1
100 NEXT n
110 PRINT "The number of times 7 occurred is ";c
120 LET t=t+c
130 PRINT "The total number of times 7 has occurred is ";t
140 NEXT x
150 PRINT
160 PRINT "The average number o f times that 7 occurred is ";t/5
170 STOP

```

```

10.
10 REM Counting heads and tail
s
20 LET h=0
30 LET t=0
40 FOR n=1 TO 80
50 LET x=INT (2*RND)
60 IF x=0 THEN LET h=h+1: GO T
O 80
70 LET t=t+1
80 NEXT n
90 PRINT "The number of times heads occurred is ";h
100 PRINT
110 PRINT "The number of times tails occurred is ";t
120 STOP

```

Unit 28

```

1. A 7,7 B 16,8
C 0,15 D 31,11
E 0,0 F 21,0
G 11,9

```

```

2. a
10 REM Line A
20 PLOT 16,130
30 DRAW 80,0
40 REM Line B
50 PLOT 24,72
60 DRAW 28,24
70 REM Line C
80 PLOT 183,40
90 DRAW -127,0
100 REM Line D
110 PLOT 116,167
120 DRAW 0,-103
130 REM Line E
140 PLOT 136,119
150 DRAW 111,-103
160 REM Line F
170 PLOT 231,88
180 DRAW -79,71

```

```

b
190 PRINT AT 4,7;"A"
200 PRINT AT 10,4;"B"
210 PRINT AT 15,11;"C"
220 PRINT AT 8,13;"D"
230 PRINT AT 11,23;"E"
240 PRINT AT 6,26;"F"
250 STOP

```

```

3.
10 REM Shape A
20 PLOT 32,32

```

```

30 DRAW 0,39
40 DRAW 39,0
50 DRAW 0,-39
60 DRAW -39,0
70 PRINT AT 15,6;"A"
100 REM Shape B
110 PLOT 48,88
120 DRAW 23,71
130 DRAW 24,-63
140 DRAW -47,-8
150 PRINT AT 7,9;"B"
200 REM Shape C
210 PLOT 120,152
220 DRAW 15,15
230 DRAW 80,-72
240 DRAW -15,-15
250 DRAW -80,72
260 PRINT AT 6,21;"C"
300 REM Shape D
310 PLOT 96,24
320 DRAW 0,47
330 DRAW 31,32
340 DRAW 48,-32
350 DRAW 0,-47
360 DRAW -79,0
370 PRINT AT 14,16;"D"
400 REM Shape E
410 PLOT 192,39
420 DRAW 0,17
430 DRAW 15,15
440 DRAW 17,0
450 DRAW 15,-15
460 DRAW 0,-17
470 DRAW -15,-15
480 DRAW -17,0
490 DRAW -15,15
500 PRINT AT 15,27;"E"
510 STOP

```

```

4. 10 REM Drawing triangles
20 INPUT "Enter x coordinate "
: x
30 IF x<0 OR x>255 THEN PRINT
"Not acceptable": GO TO 20
40 INPUT "Enter y coordinate "
: y
50 IF y<0 OR y>175 THEN PRINT
"Not acceptable": GO TO 40
60 INPUT "Enter width ";w
70 IF x>w>255 THEN PRINT "Too
wide": GO TO 60
80 INPUT "Enter height ";h
90 IF y>h>175 THEN PRINT "Too
high": GO TO 80
100 CLS
110 PLOT x,y
120 DRAW w/2,h
130 DRAW w/2,-h
140 DRAW -w,0
150 STOP

```

```

5. add to previous program:
92 INPUT "What do you wish to
draw? Enter t for triangle or p
for pentagon ";a$
94 IF a$="t" THEN GO TO 100
96 IF a$="p" THEN GO TO 160
160 CLS
170 PLOT x,y
180 DRAW 0,h/2
190 DRAW w/2,h/2
200 DRAW w/2,-h/2
210 DRAW 0,-h/2
220 DRAW -w,0
230 STOP

```

```

6. 10 REM picture
20 REM trees
30 FOR n=1 TO 20
40 READ x,y
50 DATA 20,96,24,52,40,40,54,
40,68,100,80,124,92,112,104,132,
114,112,124,132,132,112,100,48,
112,64,140,32,160,32,180,104,196
,96,216,128,236,124,244,52
60 PLOT x,y
70 DRAW 0,8
80 DRAW -7,0
90 DRAW 7,32
100 DRAW 7,-32
110 DRAW -7,0
120 NEXT n
130 REM houses
140 FOR h=1 TO 4
150 READ x,y
160 DATA 32,96,64,40,144,112,20
8,72
170 PLOT x,y
180 DRAW 0,24
190 DRAW 12,12
200 DRAW 12,-12
210 DRAW 0,-24
220 DRAW -16,0
230 DRAW 0,16
240 DRAW 8,0
250 DRAW 0,-16
260 DRAW -16,0
270 NEXT h
280 REM roads
290 PLOT 0,40
300 DRAW 103,0
310 DRAW 37,56
320 DRAW -84,0
330 DRAW 0,8
340 DRAW 91,0
350 DRAW 5,8
360 DRAW 8,0
370 DRAW -20,-32
380 DRAW 68,0
390 DRAW 0,-8
400 DRAW -74,0

```

```

410 DRAW -24,-40
420 DRAW -109,0
430 STOP

```

```

8. Keep lines 10 - 110
Add:
120 FOR n = 1 to h-1
130 PLOT x,y+n
140 DRAW w-1,0
150 NEXT n
160 STOP

```

```

9. 10 REM VIKING SHIP
20 REM Hull of ship
30 PLOT 72,72
40 DRAW 104,24,2.5
50 DRAW 8,-4
60 DRAW -8,-4
70 REM Shields
80 LET y=68
90 LET r=8
100 FOR s=1 TO 5
110 READ x
120 DATA 153,136,120,103,86
130 CIRCLE x,y,r
140 NEXT s
150 REM Mast
160 PLOT 120,76
170 DRAW 0,88
180 REM Flag
190 DRAW -20,-5
200 DRAW 20,-5
210 REM Sails
220 DRAW 0,-4
230 DRAW 54,8
240 DRAW -8,-68
250 DRAW -100,-12
260 DRAW 8,64
270 DRAW 46,8
280 REM oars
290 LET y=32
300 FOR o=1 TO 5
310 READ x
320 DATA 64,82,100,118,136
330 PLOT x,y
340 DRAW 19,28
350 NEXT o
360 PRINT AT 20,9;"VIKING SHIP"
370 STOP

```

Unit 29

```

1. a
10 REM Fat little man
20 POKE USR "M",BIN 00011100
30 POKE USR "M"+1,BIN 00011100
40 POKE USR "M"+2,BIN 00001000
50 POKE USR "M"+3,BIN 00111110
60 POKE USR "M"+4,BIN 01011101
70 POKE USR "M"+5,BIN 00011100
80 POKE USR "M"+6,BIN 00100010
90 POKE USR "M"+7,BIN 01000001
100 PRINT "M" (Entered in
graphics mode)
110 STOP

b
10 REM House
20 POKE USR "H",BIN 00111000
30 POKE USR "H"+1,BIN 01111100
40 POKE USR "H"+2,BIN 11111110
50 POKE USR "H"+3,BIN 10101010
60 POKE USR "H"+4,BIN 11111110
70 POKE USR "H"+5,BIN 10101010
80 POKE USR "H"+6,BIN 11101110
90 POKE USR "H"+7,BIN 00000000
100 PRINT "H" (Entered in
graphics mode)
110 STOP

2. 10 REM Moving car
20 POKE USR "C",BIN 00000000
30 POKE USR "C"+1,BIN 00000000
40 POKE USR "C"+2,BIN 00111100
50 POKE USR "C"+3,BIN 01010100
60 POKE USR "C"+4,BIN 11111111
70 POKE USR "C"+5,BIN 11111111
80 POKE USR "C"+6,BIN 01100110
90 POKE USR "C"+7,BIN 00000000
100 LET a=11
110 FOR b=0 TO 30
120 PRINT AT a,b;" "
130 PRINT AT a,b+1;"C" (entered
in graphics mode)
140 NEXT b
150 STOP

```

Unit 30

```

1. 10 REM Length of words
20 PRINT "WORD","LENGTH"
30 FOR n=1 TO 8
40 INPUT "Enter a word ";a$
50 PRINT a$,LEN a$
60 NEXT n
70 STOP

```