PROGRAMMING THE AMSTRAD CPC 464

Richard Meadows

Cassell
COMPUTING

PROGRAMMING THE

AMSTRAD CPC464

PROGRAMMING THE AMSTRAD CPC464

Richard Meadows

Cassell

Cassell Ltd: 1 St Anne's Road,
Eastbourne, East Sussex, BN21 3UN

British Library Cataloguing in Publication Data

Meadows, R.G.
Programming the Amstrad CPC464.
1. Amstrad CPC464 (Computer)—Programming
I. Title
001.64'2 QA76.8.A4

ISBN: 0-304-31250-9

Typeset by Phoenix Photosetting, Chatham
Printed and bound in Great Britain by Mackays of Chatham Ltd

Last digit is print no: 9 8 7 6 5 4 3 2 1

CONTENTS

Preface **vii**

1 Getting started **1**

2 Beginning to write and run programs **18**

3 Interacting with your program **34**

4 Decision making, repetition, jumping and subroutines **43**

5 Standard functions and applications **62**

6 Basic graphics, drawing and plotting **79**

7 Applications of arrays and files in programs **102**

8 Some practical programs **119**

Answers to exercises **129**

Appendix I: BASIC keywords **142**

Appendix II: Standard functions **147**

Index **150**

PREFACE

This book has been written to act as a practical guide to using and programming the Amstrad CPC464 microcomputer. No prior knowledge of computing is assumed, nor is it needed to follow the text.

The aim of the book is to give both newcomers, as well as the more experienced users, enough knowledge and understanding of the machine to enable them to write their own programs. At all stages, new concepts are thoroughly explained and backed by practical program examples. Coverage extends from the basics of the CPC464 computer and its operation to easy-to-understand, but nevertheless powerful, programming techniques.

Richard Meadows

Totteridge, London
January 1985

Richard Meadows is Head of the Department of Electronic and Communications Engineering at the Polytechnic of North London. He is an experienced lecturer and has written on a wide range of subjects, from popular home computing to advanced electronics. He is married with three children, and lives in Totteridge, North London.

Dedication

Alexis, Piers and James

1

GETTING STARTED

1.1 INTRODUCTION

This introductory chapter is designed to familiarise the newcomer with using the Amstrad CPC464 computer to gain experience in using the keyboard, correcting mistakes if they are made, and seeing how to use some BASIC commands.

We start straightaway to learn how to communicate directly with the CPC464 by typing in one of the most important commands in BASIC, the PRINT instruction. Using PRINT we can see how to use the computer as a calculator, how to display characters, words, symbols; and how to provide different formats for the display of results on monitor or tv screen.

At the end of the chapter we take an introductory look at the screen display modes and the colour facilities available on the CPC464.

1.2 SETTING UP AND MEANING OF SPECIAL KEYS

If you are not already familiar with setting up the CPC464 computer, fig. 1.1 summarises how to do this.

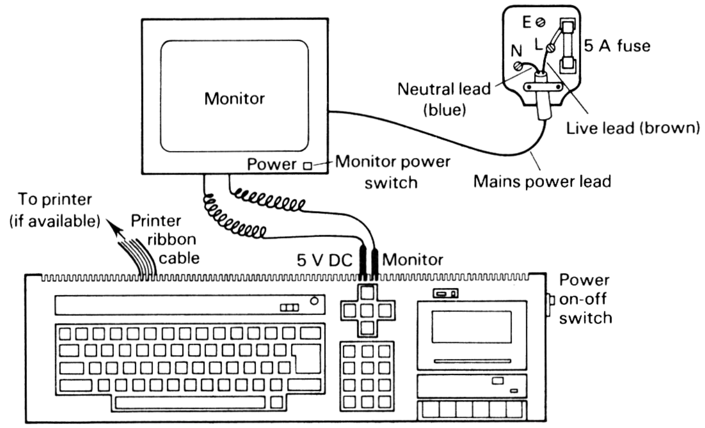
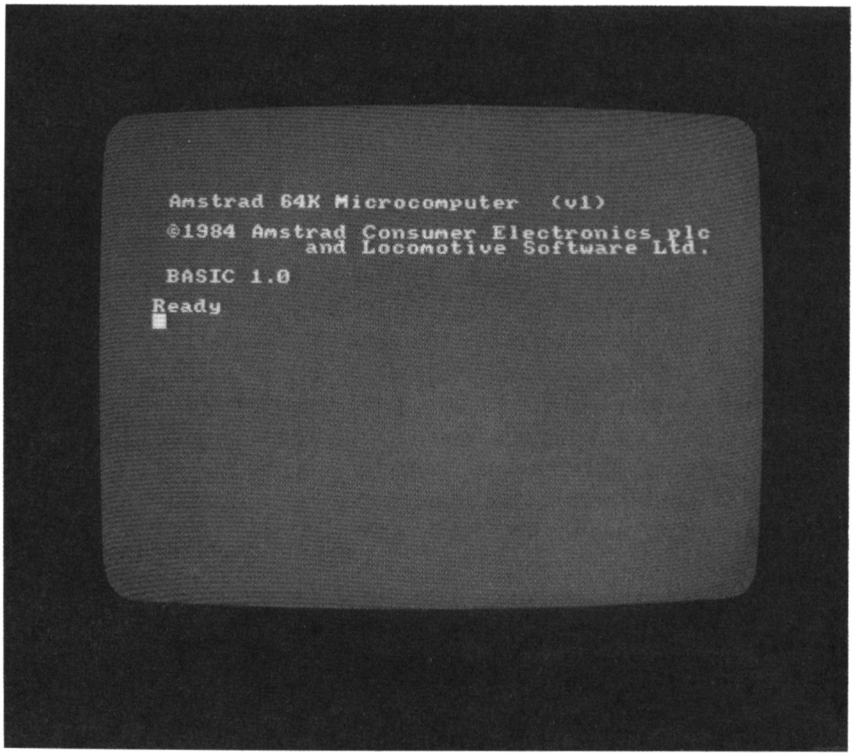


Fig. 1.1 Setting up the Amstrad CPC464 computer.

Simply set up as shown; switch on the monitor (power switch at front); switch on the computer (slide switch on right hand side)—the red light on the computer top panel shows you are on. Virtually immediately the caption shown in fig. 1.2 appears on the screen. Adjust, if necessary, the brightness control (right hand side of monitor) for a clear display.



```
Amstrad 64K Microcomputer  (v1)

©1984 Amstrad Consumer Electronics plc
        and Locomotive Software Ltd.

BASIC 1.0

Ready
█
```

Fig. 1.2 Display obtained on monitor when first switching on. Presence of cursor (the small square under Ready) shows the computer is 'ready' for immediate use.

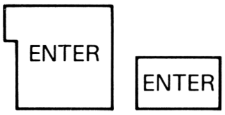
You are now 'Ready' to start. The 'Ready' condition is always indicated by the presence of the *screen cursor*—the small square directly under the R of Ready on the display.

The CPC464 computer has a full-size QWERTY keyboard that we will be using immediately to communicate with the computer. However, some keys have special functions and it is important to remember these:

The ENTER key, see fig. 1.3(a): This key is used to ENTER direct commands and program line statements into the computer for processing.

The SHIFT keys, see fig. 1.3(b): The keyboard has two shift keys; pressing SHIFT and an alphabetic key generates the upper case (capital) characters; in the case of other keys, the character engraved on the top part of the key will be generated, e.g. pressing SHIFT and the ; key generates the + symbol.

The CAPS LOCK key, see fig. 1.3(c): The CAPS LOCK key is switched on by pressing it down. When on it will generate upper case characters for the alphabetic keys, but continues to generate lower case characters (the character on the lower part of the key) for all other keys unless the SHIFT key is also pressed. CAPS LOCK is switched off by pressing the key again.



(a) the ENTER keys:
used to ENTER commands, statements, etc.

(b) the SHIFT key:
when pressed upper case characters are generated

(c) the CAPS LOCK key:
when down (or 'on') upper case alphabetic characters are generated but lower case for all other keys

(d) to DELETE:
deletes previous character (to left of cursor) and may be used repeatedly

(e) 'CLeaRs', i.e. deletes the
character under the cursor and like DELete may be used repeatedly.

(f) To RESET the computer (i.e. clear the computer memory for a fresh start):
press CTRL, SHIFT and ESC keys in order, holding each down until ESC key is pressed.

(g) ESCape key: (1) When pressed *once* will cause computer to pause when executing a program; the program can be restarted by pressing any other key.
(2) When pressed twice, program execution is halted; 'Break' message is displayed on screen. The cursor is also displayed and the computer is 'ready' for further commands.

Fig. 1.3 Function of special keys on the CPC464 computer.

The DEL and CLR keys (see fig. 1.3(d) and (e)) are used for deleting or clearing characters when errors have been made.

To RESET the computer (see fig. 1.3(f)): press CTRL, SHIFT and ESC keys in order, holding each key down until the ESC key is pressed. This action RESETs the computer completely, clearing the computer's memory and any characters on the screen, and returns the Amstrad caption and cursor to the display of fig. 1.2.

The ESC key (see fig. 1.3(g)); when pressed once, it will cause a pause in program execution; when pressed twice, it will halt or 'Break' program execution and return the cursor to the screen.

1.3 USING THE COMPUTER AS A CALCULATOR

Here we explain how to use the computer as a calculator in its direct mode—'direct' because the computer acts on a typed-in command immediately. We will be using the BASIC PRINT command and the arithmetic symbols

+ , the + key, for addition
– , the – key, for subtraction
* , the * key , for multiplication
/ , the / key, for division

All we need to do to perform a calculation is to type in PRINT, press the space bar to leave a space, and then the calculation using the arithmetic symbols and number keys (see fig. 1.4). The calculation is performed immediately the ENTER key is pressed and the result is displayed on the screen. Never type in ENTER, just press the ENTER key. Always remember to leave a space between PRINT and the calculation. If you do not the computer will respond with the message "syntax error". If you make any mistakes use the DEL key (see next section) or simply press ENTER and start again.

One further point, you can use the 'print' command in either lower or upper case, i.e. print and PRINT are identical. The computer makes no distinction between commands written in lower or upper case.

Examples

(1) To find 87 + 593
Action: type in
print 87 + 593 (and then press ENTER)

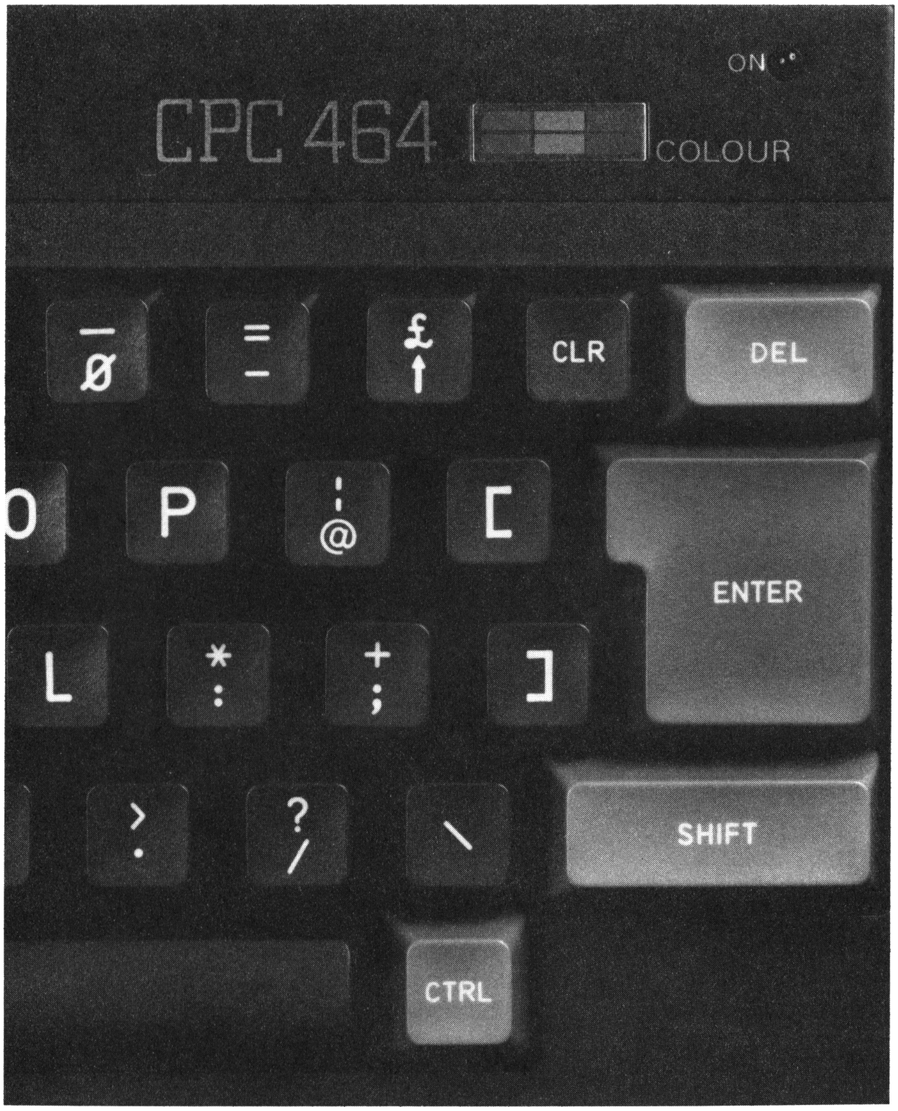
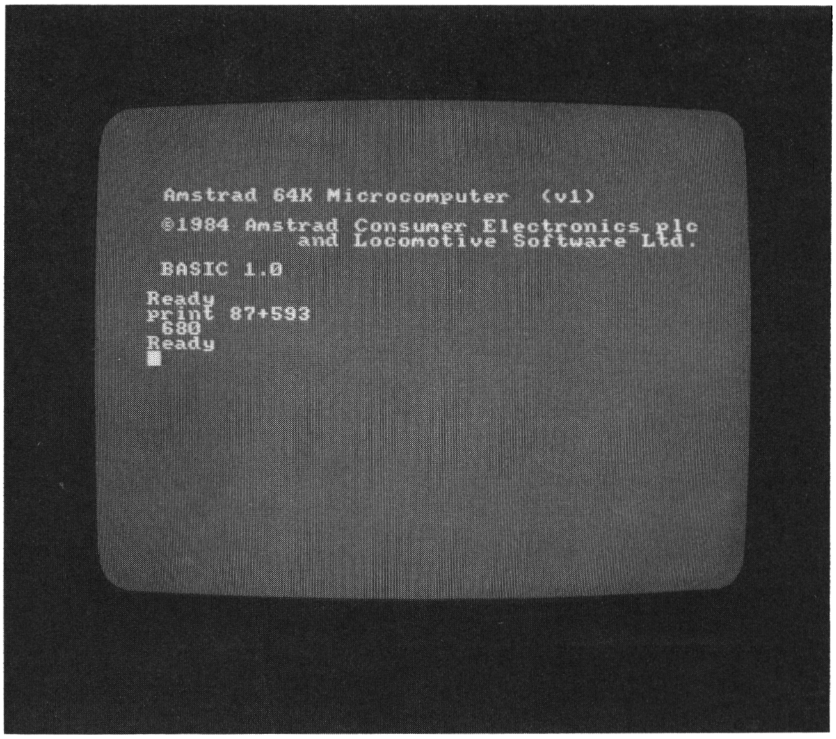


Fig. 1.4 Positions of arithmetic +, −, *, / keys.

The display obtained is shown in fig. 1.5.

Note the calculation is performed and displayed immediately the ENTER key is pressed. The result is displayed followed by 'Ready' and the cursor. The computer is now ready for further calculations.



```
Amstrad 64K Microcomputer  (v1)

©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.0

Ready
print 87+593
 680
Ready
```

Fig. 1.5 Screen display after initial switch on and on entering the print command: print 87 + 593.

(2) To find 68.72 − 29.35
Action: type in
print 68.72 − 29.35 [ENTER]
39.37 . . .answer displayed.

Note: use the full stop key for the decimal point and always remember to complete your print command by pressing ENTER; [ENTER] denotes this action.

(3) To find 46.7 × 51.23
Action: type in
print 46.7 * 51.23 [ENTER]
2392.441 . . .answer displayed.

(4) To find 83.4 ÷ 27.9
Action: type in
print 83.4 / 27.9
2.989247 . . .answer displayed.

(5) To find 52.6 + 3.4 × 5.21 − 842 ÷ 63
Action: type in

```
print 52.6 + 3.4 * 5.21 - 842 / 63
56.9489207
```

By now if you have tried all the calculations you will be almost at the bottom of the monitor screen. Any further work will still be accommodated by the display automatically scrolling upwards.

Use of ? for print

We can use the ? key as shorthand for the print command, e.g.
? 47.3 + 42.9 and PRINT 47.3 + 42.9
are identical. A further advantage in using ? for print is that you do not have to leave a space between ? and the calculation. In programs ? can also be used for print and will be listed as such in the program listing.

Exercise problems 1.1

To gain some more practice try these:
(1) 456 + 876.
(2) 75 × 67 × 32.
(3) 452.78 ÷ 45.3.
(4) 22.4 × 56.2 − 13.7 × 28.9.
(5) 0.01672 ÷ 0.00034.

1.4 CLEARING THE SCREEN AND CORRECTING MISTAKES

Clearing the screen

Simply type in cls, followed as always by pressing ENTER to action the command, i.e.
cls [ENTER]
This will clear the whole of the screen and Ready followed by the cursor will appear at the top of the screen.

Correcting mistakes: use of arrow, CLR and DEL keys

The forward and backward arrow keys can be used to position the cursor anywhere in a line. You can then use the CLR key to delete the character under the cursor or the DEL key to delete a character immediately to the left of the cursor. Holding down CLR or DEL keys will cause successive deletions; CLR effectively deletes characters to right of cursor continuously, DEL to the left.

Insertions can be made by first positioning the cursor to the wanted position using the arrow keys and then simply typing in. The rest of the line to the right of the cursor automatically scrolls sideways to make room for the insertions.

For example, find 877.663 + 998.456

Action: print877.663;998.446■

We have in fact (deliberately) made 3 mistakes:

(1) Second decimal place figure in 998.446 should be 5.

This can be corrected by pressing backward arrow once to bring cursor over the 6, then DEL key (deletes the 4), then the 5 key (inserts a 5).

(2) The ; should be +. Correct this by positioning cursor over ; , press CLR (to clear ;) and + key for + insertion.

(3) Space between print and calculation missing : position cursor to just before the 't' in print and simply press space bar to enter space. Now press ENTER and the calculation will be executed.

1.5 PRINT ". . .string. . ." commands, separators, tab and print using

In this section we explain further uses of the print command including :

(1) The PRINT ". . ." command which is used to instruct the computer to display characters, words, symbols, etc. typed within the double quotation marks. (Note: the characters enclosed within the quotation marks are known as *strings* in computer terminology.)

(2) The use of the semi-colon (;) in a PRINT instruction. A semi-colon acts as a separator which provides an immediate display with no interleaving spaces between successive string items or a single space between numeric items.

(3) The use of the comma (,) in a PRINT instruction. A comma instructs the computer to print out successive items spaced 13 columns apart, i.e. the first item begins at column 1, the second item at column 14, the third at column 27 and so on; if an item exceeds 13 characters then a multiple of 13 columns is used.

(4) The TAB command.

PRINT TAB(n) "......"

will cause *n* character spaces to be left before 'printing' a string or result.

(5) PRINT using to format the results of calculations.

Print using takes the form,

PRINT USING "###.##"; (number, calculation, etc)

and allows you to specify the form of the result. The "###.##" sets the form of the result. In our example it is set to 3 places in front of the decimal point and two behind. If the result to be printed exceeds the allocated number of #s in front of the decimal point, the result is still displayed but with a % symbol to indicate that the format is insufficient.

Examples

(1) The command,
PRINT "Happy Birthday" [ENTER]
instructs the computer to display the words (the string) within the quotation marks. On pressing the ENTER key, the string is immediately displayed on the screen.

(2) Examples showing use of semi-colon and comma:
PRINT "a";"b";"c" [ENTER]
abc . . .displayed
PRINT "a","b","c" [ENTER]

```
a               b               c        . . .displayed.
↑               ↑               ↑
column 1        column 14       column 27
```

(3) Next let us see how we can combine the display of both strings and the results of a calculation in a single PRINT statement.
Suppose we wish to display:
58.6 × 3.29 = (answer here)
on the screen.

Action: type in,

$$\text{print } \underbrace{\text{"58.6 × 3.29 ="}}_{\substack{\uparrow \\ \text{this instructs the} \\ \text{computer to 'print'} \\ \text{the string within the} \\ \text{quotation marks}}} \; \underset{\substack{\uparrow \\ \text{typing in ; means answer} \\ \text{will be displayed one space only} \\ \text{to right of = sign in string}}}{;} \; \underbrace{58.6 * 3.29}_{\substack{\uparrow \\ \text{this instructs computer} \\ \text{to work out and display} \\ \text{result of calculation}}} \text{ [ENTER]}$$

Thus after pressing ENTER, the following is displayed on the screen:
58.6 × 3.29 = 192.794

(4) Examples of use of TAB:
PRINT TAB(5) "a" [ENTER]

```
     a      . . .i.e. 'a' displayed 5 spaces from left-hand side of the screen.
```

PRINT TAB(5) "a"; TAB(10) "b"; TAB(15) "c" [ENTER]

```
     a    b    c      . . .display obtained.
```

(5) Print using example:
The computer will give results to 8 or 9 significant figures and this, more than often, is far too many. For example,

```
PRINT 104.79*52.6/33.7 [ENTER]
163.559466          . . .answer displayed.
```

We can format our result to a given form to a defined number of decimal places using:

```
PRINT USING "###.###"; 104.79*52.6/33.7
```

This command will display the result to 3 decimal places, so on pressing (as always!) the [ENTER] key we obtain the display,

```
163.559
```

Exercise problems 1.2

Try these. For the calculation try also to display the calculation 'string' as well as the result.
(1) Calculate 56.7 ÷ 3.3 and display on the screen
(a) just the result,
(b) the string "56.7/3.3 = " as well as the result,
(c) the result to two decimal places.

(2) Try typing in the following:

```
mode 0 [ENTER]
print "BASIC = Beginners All-purpose Symbolic Instruction Code"
```

followed by, as always, ENTER. Try the same with mode 2 and also mode 1 for the first command.

(3) Calculate the following and display the answers of a single screen line:
4.62×3.8 , $57.33 \div 6.3$, $42378.6 - 33499.7$.

(4) Type in the command,

```
print PI [ENTER]
```

This will display π to 9 significant figures.
What commands will display π to 1, 2, 3 and 4 decimal places?

1.6 FURTHER CALCULATIONS

Here we consider some further calculations involving the use of:
↑ key for finding powers (↑ is the lower case character on the £ key)
SQR (X) for finding square roots
Brackets ( ) keys for more involved problems.

To find powers: use of ↑ key

Examples

(1) Find 86^3
Action: type in
print 86 ↑ 3 [ENTER]
      ↑ key pressed (see £ key); note display of the 'upward arrow' key on the screen is ^
636056 . . .answer displayed.

(2) Find 1.085^{20}
Action:
print 1.085 ↑ 20 [ENTER]
5.112046 . . .answer displayed.

(3) Find $5.4^{1.4}$
Action:
print 5.4 ↑ 1.4 [ENTER]
10.60111 . . .answer displayed.

To find square roots: use of SQR(X)

CPC464 BASIC supports a number of standard functions, which we consider in some detail in chapter 5. SQR(X) is one of them. It computes the square root of the number, or numeric expression, *X*, enclosed by the brackets.

Examples

(1) Find $\sqrt{2079.36}$
Action: type in
print SQR(2079.36) [ENTER]
45.6 . . .answer displayed.
Note that the ( ) brackets—uppercase on keys 8 and 9 of main keyboard—must be used.
We could, of course, use the ↑ key, i.e.
print 2079.36 ↑ 0.5 [ENTER]
45.6 . . .answer displayed.

(2) Find $\sqrt{0.00987}$
Action:
print SQR(0.00987) [ENTER]
9.934787E−2 . . .answer displayed.
Note 9.935E− 2 = $9.935 \times 10^{-2} = 0.09935$

Using brackets

When working out calculations involving several different operations, e.g. addition, and/or multiplication, division, finding powers, etc., we must instruct the computer with the correct order. The computer acts according to the usual rules of precedence:
(1) It evaluates terms contained in brackets first.
(2) It carries out multiplication and division (from left to right) before addition and subtraction.

Examples

(1) Evaluate

$$\frac{5.6 \times 2.3}{10.2 - 9.4 \times 0.7}$$

We must enter the expression into the computer using brackets to group the numerator terms together and likewise the denominator, i.e.
Action:

```
print (5.6*2.3)/(10.2 – 9.4*0.7) [ENTER]
3.5580111          . . .answer displayed.
```

(2) Evaluate

$$\frac{61.4 - 29.5}{4.8(56.2 + 33.6)}$$

Action:

```
print (61.4 – 29.5)/(4.8*(56.2 + 33.6) ) [ENTER]
                   ↑   ↑               ↑
                   |   |               |
                   |   * must be       |
                   |     inserted      |
                   |                   |
                   outer brackets to hold together
                   the total denominator
7.400705E–2        . . .answer displayed.
```

(3) Evaluate $\left(\dfrac{21.7}{3.8 + 6.5/3.5}\right)^5$ to 2 decimal places.

Action:

```
print USING "###.##" ; (21.7/(3.8 + 6.5/3.5) ) ↑ 5 [ENTER]
830.45        . . .answer displayed.
```

Exercise problems 1.3

Try working out these:
(1) $\sqrt{}(5.62 \times 3.98)$
(2) $1500\ (12.6 + 9.4)^2$

(3) 39.6^{-2}

(4) $\dfrac{5.6(41.6 + 33.4)}{(144.5 - 96.2)}$

(5) $\left(\dfrac{4.7 + 3.6 \times 2.9}{36.2 + 45.8}\right)^3$

1.7 SCREEN DISPLAY: MODES 0, 1 AND 2

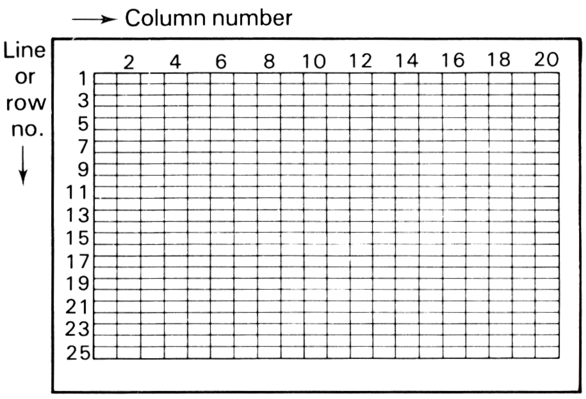
The CPC464 computer can be operated in any one of three modes which dictate the size of character display on the screen, and if using a colour monitor or tv, the number of colours you can select.

When you first switch on, the computer automatically defaults to mode 1, known as the normal mode. In mode 1 the screen can be considered as far as character unit size is concerned as consisting of 40 columns and 25 lines or rows, see fig. 1.6(b). Try typing 12345 etc. and completely fill a line of screen—note that you can display up to a maximum of 40 characters per line. In mode 1, four colours are available at any one time for screen background (paper) and character display (pen). The default colours—as you have obviously seen already—are blue for paper and bright yellow for characters.

To change mode simple type in
```
mode n [ENTER]
```
where $n = 0$ for the large character and multi-colour mode ; $n = 1$ for the normal mode; and $n = 2$ for the high resolution mode.

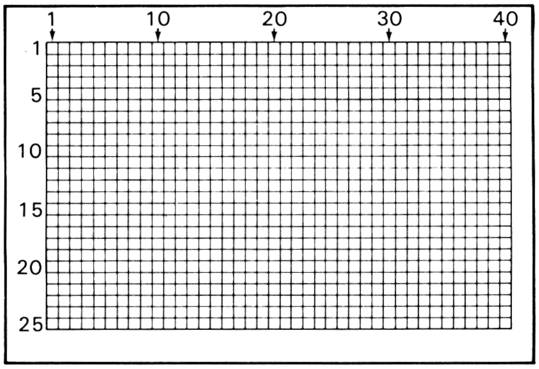
Thus if you type in,
```
mode 0 [ENTER]
```
the screen clears and you enter the 'large' character mode—you see this immediately by the larger Ready and cursor sizes on the display. In mode 0 the screen has 20 columns but still 25 lines, see fig. 1.6(a). Thus compared with mode 1 the character width is effectively doubled with up to a maximum of 20 characters per line compared with 40 for the normal mode. Sixteen colours may be used for paper and pen at any one time in mode 0.

Now type in,
```
mode 2 [ENTER]
```
The screen immediately clears and the display changes to mode 2, the high resolution mode. In the high resolution mode there are 80 columns—so it is possible to have up to 80 characters per line—but still as for modes 0 and 1, 25 lines. Mode 2 can be useful in programming where line statements have more than 40 characters since such longer statements can still be contained in a single screen line. Only two colours are available at any one time in mode 2.



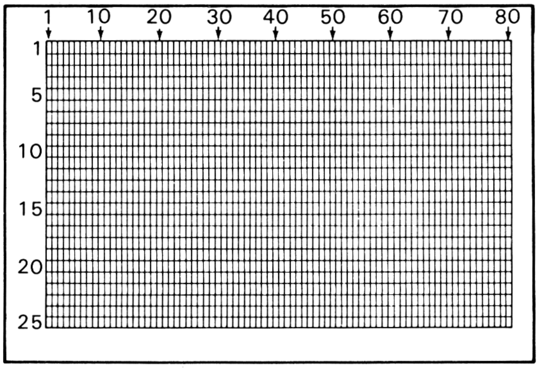
(a) *mode 0*

20 columns by 25 lines,
16 colours available,
160 x 200 pixels
'multi-colour, large character size mode'



(b) *mode 1*

40 columns by 25 lines,
4 colours available,
320 x 200 pixels
'normal mode'



(c) *mode 2*

80 columns by 25 lines,
2 colours available,
640 x 200 pixels,
'high resolution mode'

Fig. 1.6 The three screen modes available for the CPC464 computer for screen display.

1.8 INTRODUCTION TO COLOUR: BORDER, PAPER, PEN AND INK COMMANDS

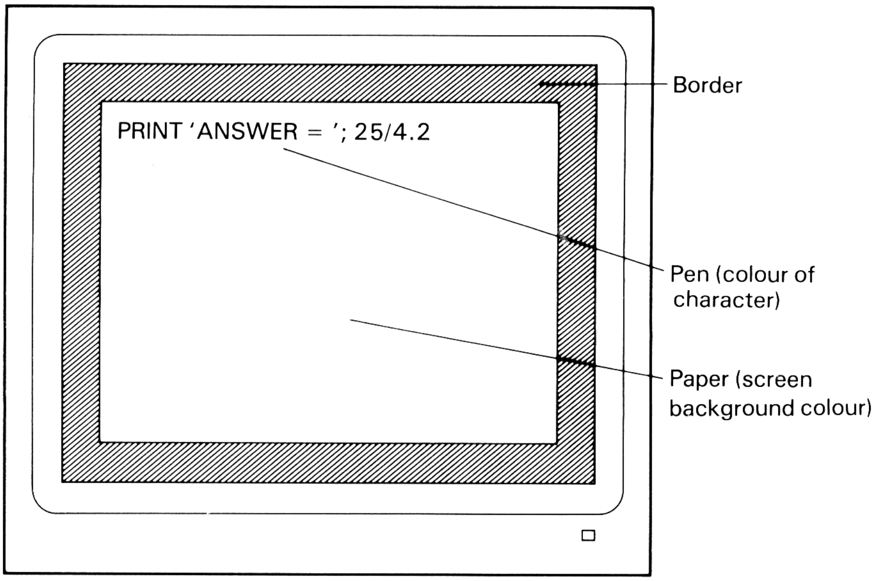

PRINT 'ANSWER = '; 25/4.2
Border
Pen (colour of character)
Paper (screen background colour)

Fig. 1.7 Screen areas: border, paper (main screen background) and pen (character colours).

If you are using a colour monitor or a colour tv (in conjunction with the Amstrad MP1 power supply and modulator unit) you can set the colour of screen border, the central screen background (the 'paper') and that of the character display (the 'pen') using simple commands.

You have a choice of 27 colours although only a selection of these are available for display at a given time:
in mode 0 : up to 16 colours are available at any one time,
in mode 1 : up to 4 are available,
in mode 2 : only 2 are available.
For the 'green' only GT64 Amstrad monitor, the 'colours' are displayed as various shades of green.

When you first switch on, the display is automatically in mode 1 (the default mode) with a blue screen (paper plus border) and bright yellow characters (pen).

To set the border colour use the command,
border (ink number) [ENTER]
where the *ink number* is taken from the 'ink number colour reference chart' of fig. 1.8. For example,
border 26 [ENTER]
will change the border colour from blue (default) to ink colour no. 26, which you will see from fig. 1.8 is bright white. Any one of the 27 colours can be used

Ink no.	Colour	Ink no.	Colour	Ink no.	Colour
0	black	10	cyan	20	bright cyan
1	blue	11	sky blue	21	lime green
2	bright blue	12	yellow	22	pastel green
3	red	13	white	23	pastel cyan
4	magenta	14	pastel blue	24	bright yellow
5	mauve	15	orange	25	pastel yellow
6	bright red	16	pink	26	bright white
7	purple	17	pastel magenta		
8	bright magenta	18	bright green		
9	green	19	sea green		

Fig. 1.8 Ink number colour reference chart.

for border. To see each one of these colours try entering in the following short program. Type in each line and at the end of each line always press [ENTER]. To run the program, type in
RUN [ENTER]
The program uses the border command at line 50 and runs consecutively through the available ink colours from ink number 0 to 26.

```
10 REM *** PROGRAM TO SHOW 27  COLOURS ***
20 REM  ** AVAILABLE ON THE AMSTRAD CPC 464 **
30 CLS
40 FOR ink.no = 0 TO 26
50 BORDER ink.no
60 PRINT "ink number of border =";ink.no
70 FOR delay = 1 TO 4000 : NEXT delay
80 CLS
90 NEXT ink.no
```

Paper **and** pen **commands**

The colour of the screen background—the paper—can be set using the paper command and referencing the paper/pen number table of fig. 1.9 as follows:
paper (paper number) [ENTER]
cls [ENTER]
The paper command sets the screen colour as dictated by the paper number and mode being used (i.e. according to the reference table of fig. 1.9). The cls command clears the screen to this colour. For example, for mode 1
paper 3 [ENTER]
cls [ENTER]
changes the screen to bright red.

Paper/pen no. 'Ink-well no.'	*Mode 0* 16 colours, 20 columns	*Mode 1* 4 colours, 40 columns	*Mode 2* 2 colours, 80 columns
0	blue	blue	blue
1	bright yellow	bright yellow	bright yellow
2	bright cyan	bright cyan	blue
3	bright red	bright red	bright yellow
4	bright white	blue	blue
5	black	bright yellow	bright yellow
6	bright blue	bright cyan	blue
7	bright magenta	bright red	bright yellow
8	cyan	blue	blue
9	yellow	bright yellow	bright yellow
10	pastel blue	bright cyan	blue
11	pink	bright red	bright yellow
12	bright green	blue	blue
13	pastel green	bright yellow	bright yellow
14	flashing blue↔yellow	bright cyan	blue
15	flashing pink↔sky blue	bright red	bright yellow

Fig. 1.9 Paper (screen) and pen (characters) colour reference chart. Screen and character colours may be changed using the paper, pen and ink commands.

The character colour may be changed using the pen command and again referencing the table of fig. 1.9 for the pen number colour, i.e.
pen (pen number) [ENTER]
so pen 2 [ENTER]
will change, for modes 0 and 1, the character colour to bright cyan. Note paper and pen numbers are identical. They can be thought of as 'ink-well' numbers containing various ink colours.

Use of ink command

The paper and pen colours can be changed directly using the ink command which takes the form:
ink (present paper/pen no. of colour) , (ink no. of new colour) [ENTER]
Thus, for example, if we first reset by pressing CTRL, SHIFT and ESC keys (holding each down until ESC is pressed), then we return to the default position: mode 1, blue screen (border plus paper) and bright yellow characters (pen), i.e. paper no. = 0 (blue), pen no. = 1 (bright yellow). Now if we wish to change the paper to, say, bright green (ink no. = 18) and pen to orange (ink no. = 15) we use,
for the paper colour change: ink 0,18 [ENTER]
for the pen colour change: ink 1,15 [ENTER]

2

BEGINNING TO WRITE AND RUN PROGRAMS

2.1 INTRODUCTION AND SUMMARY

In this chapter we begin to write simple but complete programs on the CPC464 computer.

We introduce the chapter with a brief discussion of what a program is, the form a BASIC program takes, and some important points to be noted in program construction.

We then consider some of the important BASIC commands and instructions that will be used to form our program statements, to execute the program, and to display a listing of the program contents.

We consider also the meaning of variables and allowable identifiers that may be used within programs for handling data; how a program may be *edited*; and finally how a program may be *saved* and how a program stored on a cassette may be *loaded* and '*run*'.

2.2 SOME INTRODUCTORY COMMENTS ON PROGRAMMING

So far we have, with one exception, been entering direct commands into the computer. These commands have been directly executed by the computer with an immediate action such as display of results, change of colour, etc. Clearly we are very limited in using only direct commands to solve our problems or execute all but the simplest of tasks. The idea of a program is to tie our set of commands together in a complete self-contained package which we can 'run' at one go.

What is a program? A program consists of a complete list of consecutive instructions designed to solve a particular problem or execute a given number of tasks.

To provide an immediate idea of the form taken by a simple BASIC program, consider the following example:

```
10 CLS
20 PRINT "Bank and public holidays 1986"
30 PRINT "*****************************"
35 PRINT
40 PRINT "Bank holiday,Wednesday 1 January"
50 PRINT "Good Friday, Friday 28 March  "
60 PRINT "Easter Monday,31 March "
70 PRINT "May day holiday, Monday 5 May "
80 PRINT "Spring bank holiday, Monday 26 May"
90 PRINT "Summer bank holiday, Monday 25 August"
100 PRINT "Christmas day, Thursday 25 December"
110 PRINT "Boxing day, Friday 26 December"
120 PRINT
130 PRINT "---------------------------------"
```

This simple program consists of a list of 14 instructions, each instruction typed in as a separate line statement. A line statement is an instruction to the computer to perform a specific operation.

Each line statement must always be preceded by a line number, the value of the numbers dictating the order in which the program will be executed, i.e. line statement 10 will be executed first, followed by 20, 30, 35, 40 . . . 130 in the above example. Any ascending order sequence can be used for the line numbers. We chose 10, 20 . . . 130 but we could have used 1, 2, 3, 4, 5, etc. or 15, 26, 48, 115, 207 . . . in fact any sequence in the range 1 to 65535 (65535 is the maximum line number for the CPC464 computer).

In typing in a program from the keyboard, each line statement is entered into the computer by pressing the ENTER key. The action of ENTER is to store the line in the computer. No further action is taken until the program is commanded to RUN. This is different from using the computer in the direct mode, where, for example

```
PRINT "Monday, Tuesday, Wednesday" [ENTER]
```

would produce immediate (direct) execution and cause

```
Monday, Tuesday, Wednesday
```

to be displayed on the screen.

Thus if a line is not preceded by a line number then the line statement is interpreted as a direct command and the computer processes the statement immediately. When the statement is preceded by a number then the statement becomes part of a program.

Now try typing in the program. Remember to press the ENTER key at the end of each line. The program can be run by typing in the direct command

```
RUN [ENTER]
```

which causes execution of lines 10, 20, 30, 35, 40, 50 . . . 130 ascending order and produces immediately the following display on the screen:

```
Bank and public holidays 1986
*****************************

Bank holiday,Wednesday 1 January
Good Friday, Friday 28 March
Easter Monday,31 March
May day holiday, Monday 5 May
Spring bank holiday, Monday 26 May
Summer bank holiday, Monday 25 August
Christmas day, Thursday 25 December
Boxing day, Friday 26 December

---------------------------------------
```

Before proceeding further, let us briefly review some basic points to be considered in writing a program.

(1) *Ensure that you have an exact understanding of what you want your program to do. Define your problem as clearly as possible.*

This is perhaps obvious, but absolutely essential as your first step for any serious problem solving using a computer. Note what information you are given—this will constitute the data input to your program (we see how to input information using INPUT and READ-DATA statements in the next chapter). Define clearly what information output you require from your program.

(2) *Develop general ideas for your program solution. Think out a suitable sequence for your instructions.*

Draft a logical sequence of the actions required to solve your problem. From these form the basis of what type of line statements (or, as we shall see in chapter 4, what subroutines) are needed to effect your solution.

(3) *Type in the program line statements.*

Remember that each statement must be preceded by a number and that the computer acts on these statements in ascending number order. It is a useful tip to separate consecutive lines by 10, e.g. use 10, 20, 30 . . . etc., so that if you subsequently need to insert lines in your program you have plenty of space to do so.

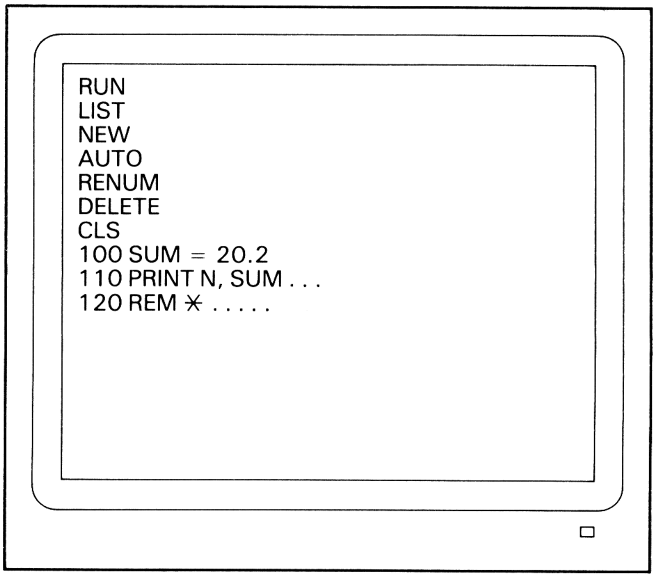
All line statements must, of course, be written according to the syntax (the 'grammar' rules) dictated by the version of BASIC language being used on the CPC464. If these are not followed exactly the program will not run. In practice the program is continuously monitored as each line is typed in. After each line is entered, the computer scans it for errors and will display an error message, if any should be present, giving guidance as to the type of error detected.

(4) *Feed in any data (when and where necessary) to the program.*

This may be done interactively using INPUT statements or by storing data within the program itself using READ . . . DATA statements.

2.3 SOME FUNDAMENTAL BASIC COMMANDS AND STATEMENTS

Let us now review some of the important BASIC commands and statements that we will be using to run and write our programs.



```
RUN
LIST
NEW
AUTO
RENUM
DELETE
CLS
100 SUM = 20.2
110 PRINT N, SUM . . .
120 REM * . . . . .
```

Fig. 2.1 Fundamental BASIC commands and statements.

1 First some fundamental direct mode commands:

RUN tells the computer to execute or 'RUN' our program; the processing starts, in the absence of any number typed in after RUN at the lowest number line statement in our program.

Although RUN can be used within a program, its normal use is to start program execution by typing RUN as a direct mode command.

RUN 40 would start execution at line statement number 40.

LIST displays the program currently stored in the computer on the screen.

LIST 10 – 100 will list the group of program statements starting at 10 and ending at 100.

LIST 50 – will list from line 50 to the end of the program.

LIST – 70 will list from start to line 70.

LIST 90 will list specified line 90.

NEW clears all programs stored in the computer—it allows us to start afresh or 'a-new'.

AUTO automatically assigns for us line numbers in steps of 10, i.e. 10, 20, 30, 40 . . . during the writing of our program; pressing the ESC (ESCAPE) key cancels the automatic line numbering.

RENUM renumbers program line statement numbers starting at 10 with increments of 10, i.e. line numbers would be renumbered 10, 20, 30, . . .

RENUM 100, 10, 200 renumbers program lines from 'old' line number 10, to give it a new number 100 and increments subsequent lines by 200; the general form of the command is:

RENUM (new line number) , (old line number) , (line increment)

DELETE 50 deletes line 50.

DELETE −100 deletes all lines up to and including line 100.

DELETE 200− deletes all lines from 200 to end of program.

DELETE 20 − 80 deletes range of lines, in this case lines 20 to 80 inclusive.

CLS clears the screen; useful to include this as first line statement in a program to ensure screen is cleared and results can be clearly displayed starting from top of the screen.

To BREAK *or abort program execution:*

If you wish to force a halt to program execution:
press ESCape key twice.

If you just wish program execution to pause:
press ESCape key once and then to restart, press any other key.

2 Some fundamental program statements

(1) *Assignment statements: the LET statement*

100 LET X = 98.2 or equivalently

100 X = 98.2

This statement assigns to the variable *X* the value 98.2. The use of the keyword LET is optional in BASIC and therefore may be omitted. Thus the two assignment statements given above are identical.

The use of the = sign in a LET statement does not have quite the same significance of equality as in normal algebra or arithmetic. For example, the program

```
100 SUM = 527
110 X = 243
120 SUM = SUM + X
130 PRINT SUM
RUN (ENTER)
770        . . .result obtained
```

i.e. the result yielded on running the program is SUM = 527 + 243 = 770. Statement 120 is nonsense in normal algebraic terms (unless $X = 0$). In computing, however, the assignment statement = sign means '*LET left hand variable take the value of the right hand expression*' or more briefly '*is given the value of*'. Thus statement 120 is read as: let the variable SUM take the value of the original SUM value (i.e. 527) plus the value of *X* (i.e. 243).

(2) *Output statements: the* PRINT statements
We have already used PRINT in direct mode usage. PRINT is used in an identical way in program statements. For example,
120 PRINT X instructs the computer to display the value currently assigned to *X* on the screen
140 PRINT "SOLUTION=";X displays the words within the quotation marks, i.e. the string, immediately followed by the value of *X*
160 PRINT 'prints' a blank line and is very useful in separating results for clear display.

(3) *The REMark statement*
10 REM cash-flow program is used to include a REMark or comment, e.g. to put in program title or any helpful explanatory note; it is for our convenience only—it is passed over during the execution of the program.

(4) *Use of the colon* :
More than one statement may be written on a single line by separating the individual statements by a colon, i.e. :
For example, the program

```
10 X = 242
20 Y = 251
30 Z = 363
40 PRINT X + Y + Z
```

could be condensed to

```
10 X = 242: Y = 251: Z = 363
20 PRINT X + Y + Z
```

or even into a single line,

```
10 X = 242: Y = 251: Z = 363: PRINT X + Y + Z
```

(5) *Use of* CLS *as a program statement:*
10 CLS will clear the display on the screen (just as the direct command would) and is very useful in a program to ensure clear presentation of results.

2.4 EXAMPLE OF WRITING AND RUNNING SOME SIMPLE PROGRAMS

(1) Try writing a program to display the names and phone numbers of your friends.
Here is the form the program could take:

```
10 REM *** Friends Phone Nos. ***
15 CLS
20 PRINT "Jane" TAB(15)"44 0138 98"
30 PRINT "Harry"TAB(15)"78 54 8769"
```

```
40 PRINT "Mary" TAB(15)"887 98 2385"
50 PRINT "Fred" TAB(15)"101 77390"
60 PRINT "The Bell" TAB(15)"33 43432 78"
70 PRINT "Rev. Greene"TAB(15)"2740 1092"
```

Remember each line statement has its own line number and must always be 'entered' by pressing the ENTER key. Try using AUTO for automatic line numbering.

To execute the program type in RUN followed by pressing the ENTER key, i.e.

RUN [ENTER]

You will, of course, obtain the following display

```
Jane           44 0138 98
Harry          78 54 8769
Mary           887 98 2385
Fred           101 77390
The Bell       33 43432 78
Rev. Greene    2740 1092
```

The program is easy to understand:

Line statement 10 contains REM (REMark), a remark statement to 'remind' us what our program does.

Statement 15 uses CLS to clear the screen.

Statements 20 to 70 instruct the computer to 'print' the names and phone numbers; we have also used the tab instruction to line up the start of the numbers, i.e. tab(15) causes the numbers to be displayed commencing at column 15.

If you now type in,

LIST [ENTER]

you will obtain a re-display of the program listing.

If you follow this by,

NEW [ENTER]

you will clear the program from the computer memory, although not the display on the screen. Check this by typing in,

CLS [ENTER]

LIST [ENTER]

The CLS command clears the screen. The LIST command lists the program currently stored in the computer. Since no 'list' appears the program was in fact cleared by the NEW command.

Note the BASIC keywords run, list, print, cls, etc. can be entered in either lower or upper case letters. The computer makes no distinction. Your program listing, however, will always display keywords as upper case characters.

(2) Try displaying your initials in the three screen modes: mode 0 (large character), mode 1 (normal), mode 2 (high-resolution).

Here is a program which displays each mode presentation in turn. Again the program is virtually self-explanatory, except for lines 40, 70 and 100. These are all identical and are an example of the use of a FOR loop (see chapter 4). They are included to produce a delay of 2 to 3 seconds so you can see the display for each mode. Note also the use of the colon, which enables us to include more than one instruction in a single line statement.

```
10 REM Initials in different modes
20 MODE 0 :PRINT
30 PRINT "R G M":PRINT "******":PRINT
40 FOR n=1 TO 2000:NEXT n:REM delay
50 MODE 1
60 PRINT "R G M":PRINT "******":PRINT
70 FOR n=1 TO 2000:NEXT n:REM delay
80 MODE 2
90 PRINT "R G M ":PRINT"******":PRINT
100 FOR n=1 TO 2000:NEXT n:REM delay
110 MODE 1
```

(3) Here is an example of a simple program to calculate the area of a triangle:

```
10 REM Simple area calculation
20 CLS
30 PRINT "area of triangle":PRINT
40 PRINT "A =(b x h)/2"
50 b=45.6 :h=33.8
60 PRINT "A =";b*h/2
70 PRINT
80 PRINT "for b=";b,"h=";h
```

Again the program should be reasonably easy to understand. Line 50 assigns *b* the value 45.6 and the second statement (separated by the colon) assigns *h* the value 33.8. Line 60 displays the result for area, *A*.

On running the program, you will obtain the following display:

```
area of triangle
A =(b x h)/2
A = 770.64

for b= 45.6   h= 33.8
```

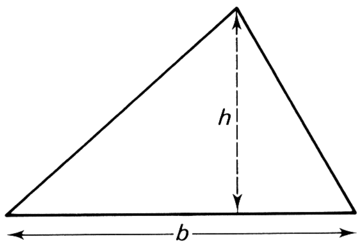


Fig 2.2 Area of triangle, $A = \frac{1}{2}b \times h$.

2.5 VARIABLES AND IDENTIFIERS

When we assign values to quantities to be used in our programs we need a 'box' or storage location to hold these values. In computer terminology we talk about a *variable* (the word used for the quantities we wish to process etc. in our programs) and an *identifier* (the name we use to 'identify' or label our 'variable' box).

The term *variable* refers to any element or quantity in a program whose value may (or may not) be changed in the execution of the program. Variables are used to provide storage locations in the computer to hold the data the variables represent.

Each variable we use in a program must be given a unique *identifier*. The variable identifier may be thought as the name of the 'container' used to store the data value currently assigned to the variable. Once a value is assigned to a given variable, the value remains fixed at this value and stored in its 'container', but may be changed when or if any subsequent program statement assigns the variable a new value.

In the version of BASIC used on the CPC464 computer there are essentially three main classes of variables.

1 Real or numeric variables

These are used to store numbers (with or without decimals) in floating point form within the computer to 8 to 9 significant figure accuracy within the range for CPC464 computer : $-\ 10^{-38}$ to $+\ 10^{38}$.

Identifiers for real variables must always begin with an alphabetic letter and can be followed by any sequence of letters or number digits. For the CPC464 a full stop symbol can be included. The total character length for an indentifier must not exceed 40.

Obviously no BASIC keywords, e.g. print, list, cls, etc., can be used as variable identifiers. All keywords, system commands, etc. are regarded as *reserved* words and must never be used except in their correct context. Note also that spaces, punctuation marks (other than the full stop) and arithmetic symbols cannot be used in identifiers. For example,

a	maxvalue	Z237	GREATEST
area	min. point	interest	least
x11	last. value	POPULAR	no. of. terms

are all valid identifiers for real (number) variables; the following, however, are not:

max value	. . . invalid, spaces cannot be used in identifiers
x + y	. . . invalid, arithmetic symbols cannot be used
input	. . . invalid, BASIC reserved word
first. value?	. . . invalid, ? cannot be used

2 Integer variables

These are used to store whole number values, i.e. integers. The range for the CPC464 computer is −32768 to +32767. Integer variable values occupy less space in the computer memory and so can be used in preference to real variables in programs that do not require decimal information.

Integer variables are distinguished from real variables by including the % symbol as the last character of the identifier. Integer variable identifiers follow exactly the same syntax rules as for real variables but must always end with %, e.g.

index% represents an integer variable identifier
index represents a real (floating-point) variable identifier

3 String variables (non-numeric variables)

These, as their name implies, are used to store 'strings' of characters, i.e. letters, numbers, symbols, etc. String variable identifiers are distinguished from numeric variables by always ending in the $ (dollar) symbol. It is the $ symbol that 'tells' the computer that it is handling string variable data. Apart from the $ symbol, string variable identifiers are formed using the same general rules: they must start with a letter, may then be followed by any sequence of letters, numbers or full stops (up to a maximum including the $ symbol of 40), e.g.

name$	nameandaddress$
x11$	SUBJECT3$

are valid string variable identifiers.

We use assignment statements to assign values to string variables in the same way as for numeric variables but we must enclose the actual value being assigned (the string) within double quotation marks, as shown for example in the following short program:

```
10 REM ** String variable example 1 **
20 CLS
30 name$="Richard"
40 blank$="  "
```

```
50 address$="4260 Brasilia Cresent, Clifton"
60 PRINT name$
70 PRINT address$
80 PRINT
90 PRINT name$;blank$;address$
```

On running the program, we obtain the display

```
Richard
4260 Brasilia Cresent, Clifton

Richard 4260 Brasilia Cresent, CliftonRichard
```

String variables can be combined to form larger strings using the + symbol. For example, in the following program at line 60:
total$ = name$ + blank$ + address$
assigns the string variable total$ the value:
"Richard 4260 Brasilia Cresent, Clifton"
Try running the program to verify this.

```
10 REM ** String variable example 2 **
20 CLS
30 name$="Richard"
40 blank$=" "
50 address$="4260 Brasilia Cresent, Clifton"
60 total$=name$+blank$+address$
70 PRINT total$
```

Note for all variable classes: upper and lower case usage

Identifiers written in either lower or upper case letters are treated as identical in BASIC. Thus, for example, the following pairs of identifiers are treated by the computer as the same:
X , x ; N% , n% ; name$, NAME$
Do not fall into the trap of thinking they can be used for two different variables—they are indistinguishable as far as program execution is concerned.

Identifiers are used to identify not only variables but, as we shall soon see, for arrays, user-defined functions, loops, file-names, etc.

One final point—always try to choose meaningful identifiers for the variables used in your programs. This adds extra clarity to the program listing. It helps in understanding what your program is about, is especially useful in tracing errors and for future reference, and invaluable for others who may use your programs.

2.6 USING A PRINTER TO OBTAIN HARD COPY

If you are lucky enough to have a printer—the CPC464 may be used directly with a parallel printer with Centronics interface, the Epson MX, RX and FX series of printers being ideal—you can easily obtain a hardcopy print-out of your programs listings and also direct your program output to the printer as well as or instead of the monitor screen.

The listing command is simply,

```
LIST #8 [ENTER]
```

The # symbol (uppercase symbol on key 3) denotes the 'channel' and 8 the channel number used by the computer to transmit the program listing data to the printer.

The same channel is also used to output the PRINT statement data from within the program to the printer.

PRINT statements for printer output take the form

```
100 PRINT #8,"Results follow:"
110 PRINT #8, 56.2*39.7
```

i.e. #8, is added to the PRINT instruction to direct the output to the printer.

2.7 EDITING PROGRAMS: THE EDIT COMMAND AND USE OF COPY

The CPC464 has excellent editing facilities for correcting mistakes and making changes in a program listing.

The mistakes in any given line statement can always be rectified by typing the line in again with, of course, the errors removed. On pressing ENTER the new line automatically replaces the original. This method, however, is rather laborious especially as the CPC464 provides two very easy-to-use methods.

Use of the EDIT command

Suppose we wish make a correction or modification to, say, line 100 in a program. Type in the command,

```
EDIT 100 [ENTER]
```

Line 100 will be then displayed on the screen with the cursor at the beginning of the line. The cursor is then positioned using the forward (or backward) cursor arrow keys to where required in the line and deletions made using either CLR or DEL keys. If text is to be inserted space is made automatically as you key in by the portion of the line to the right of the cursor scrolling to the right. After completing the changes to the line, enter the line into the program by pressing the ENTER key. If no errors are detected the line replaces the old line in the program list. If errors are still present these will be detected by the computer and it is not possible to move off the line until these are removed.

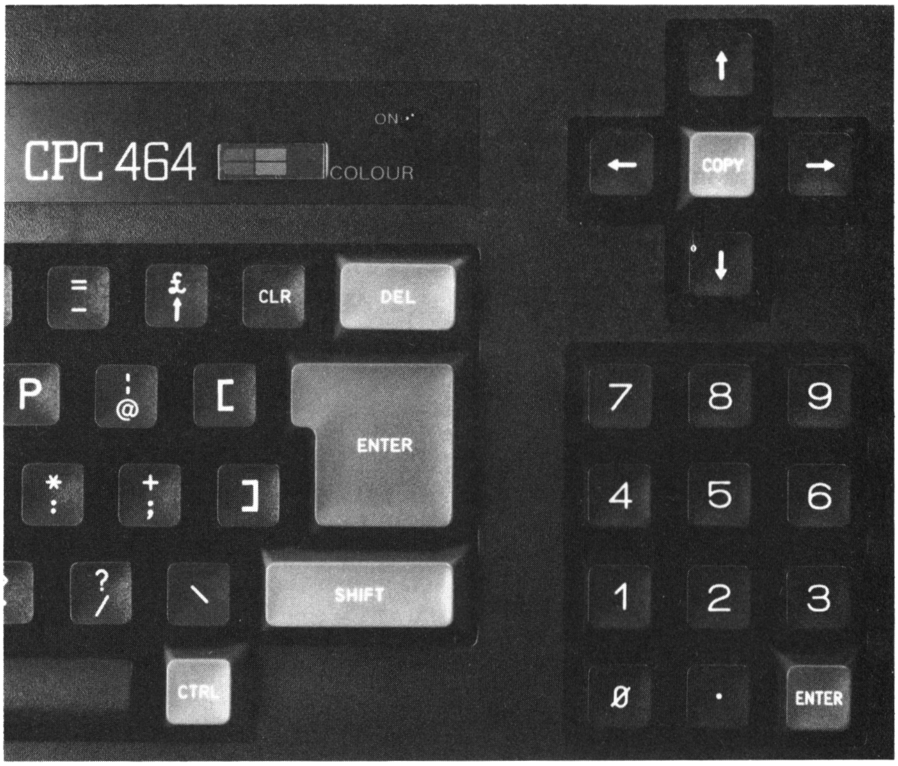


Fig. 2.3 Keys involved in EDITing programs: cursor position, COPY, CLEAR and DELete keys.

Use of COPY cursor

First list your program or, if the program exceeds 20 or so lines, list the sections you wish to edit. Hold down the SHIFT key and press the upward arrow (↑) cursor key until the 'copy' cursor is positioned at the beginning of the line to be edited. Note that the 'main' cursor remains in position at the end of the listing, so we now have two cursors on the screen.

Now press the green COPY key until the copy cursor is at the position where you wish to make a change. Note as you press the COPY key the main cursor also moves, copying the line as it goes. Make your changes and note that these are included in the main line but not at the copy cursor position, which remains stationary. After completing your changes move on again by pressing COPY and finally press ENTER. The 'main' line then replaces the original line in the program, provided the latter contains no errors. As with the EDIT command process you cannot move the COPY cursor off the line until all errors in the main line are removed. If no errors are detected the 'main' line enters the program and the copy cursor disappears.

2.8 SAVEing AND LOADing PROGRAMS

1 The SAVE command (cassette tape)

To save the program currently stored in the computer simply use the command,
SAVE "program name" [ENTER]
Note that the program name is the name you give to the program and it will be stored and accessed under this name. The name must be included within the quotation marks and can be any combination of characters including spaces up to 16. Once the SAVE command is entered the computer responds with the following message display on the screen,
Press REC and PLAY then any key
Press the RECord and PLAY buttons of the cassette firmly down until both are locked in position. As soon as you press any other key, the cassette tape starts and copying commences as indicated by a further message on the screen:
Saving program name block 1
When copying is complete the cassette tape stops. Ready is displayed and the cursor returns to the screen. Finally press the STOP/EJECT button.

2 The LOAD command

To load a program stored on cassette tape, place the cassette containing the program in the cassette unit of the computer, rewind to the beginning of the tape or, if you know the 'count' number on the cassette revolution counter where the program starts, rewind to this, and then enter the command:
LOAD "program name" [ENTER]
The computer immediately responds with the screen message:
Press PLAY then any key
As soon as the PLAY button is pressed and locked down and then any other key pressed the cassette tape will start moving and a search for the program initiated. In this search the computer will display all other program titles it finds before reaching and then loading the requested one. Thus messages of the form
Found Statistics 1
Found Graphics Demo
might be displayed, but as soon as the required program is reached, we are notified so by the message
Loading program name block 1
and when the loading is fully complete, Ready is displayed and the cursor returns to the screen.

To check the fact that the program has been copied from tape to computer, type in the LIST command. The program can, of course, be run once we know it has been 'loaded' by the usual RUN command.

To load and automatically run a program stored on tape you can use the single command:

RUN "program name" [ENTER]
The computer responds with (as before):
Press PLAY then any key
and the loading processes commence with the search, the load and then an automatic RUN without the need of any further commands.

Exercise problems 2

(1) Write programs to display the following:
(a) names and dates of birth of family and friends
(b) a list of common items and their prices
(c) a football league table.

(2) Write a program to calculate the volume of
(a) a rectangular block (cuboid) (see fig. 2.(a))
$v = b \times l \times d$, for $b = 2.27$, $l = 5.74$, $d = 1.86$
(b) a cylinder (see fig. 2.4(b))
$v = \pi r^2 h$, for $r = 8.63$, $h = 12.4$ and where $\pi = 3.142$ (or use PI, which holds an accurate value for π in the CPC464).

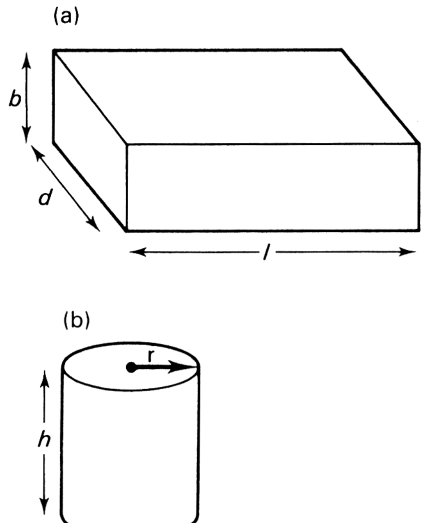


Fig. 2.4 (a) Cuboid: $v = b \times l \times d$; (b) Cylinder: $v = \pi r^2 h$.

(3) Write a program which displays the result of the following calculations:
(a) $3.4 \times 5.67 - 6.21 \times 4.98$
(b) $5.37^2 + 4.46^2$
(c) pv^c , where $p = 525$, $v = 72$, $c = 1.4$

(4) Write a program to find the average value of the following weights:
20.7 kg, 14.5 kg, 12.9 kg, 13.6 kg

(5) Write a program to calculate the compound interest on
(a) £1000 invested for 5 years at 11%
(b) £25 invested for 32 years at 7.5%
The compound interest formula is
$I = P(1+R/100)^N - P$
where I = interest, P = sum invested, R=rate %, N=number of years invested.

3

INTERACTING WITH YOUR PROGRAM

3.1 INTRODUCTION AND SUMMARY

So far the programs we have considered have contained all the necessary information within the actual program with no need to reference any outside source to complete their relatively simple tasks. To enable us to interact with a program during its execution and, for example, be able to enter values from the keyboard, BASIC provides the INPUT statement. We consider the application of INPUT statements in this chapter.

To enable us to store a number of values such as a list of data, BASIC provides the READ . . . DATA statements. The use of these statements is normally much more convenient than writing individual assignment statements for each variable especially for a long list of items. READ . . . DATA statements are also considered in this chapter.

Frequently we may wish to STOP program execution and make, for example, a check before allowing it to continue. The use of STOP statements and the CONTinue command is considered. In using READ . . . DATA statements we may also wish to go back to the beginning of the DATA list at some point in the sequence of program execution; to do this BASIC provides the RESTORE statement, the use of which we finally consider.

3.2 INPUT STATEMENTS

INPUT statements are used to input values to the computer directly from the keyboard. They have the basic format:

20 INPUT (variable identifier)
20 INPUT ". . .string message. . ." ; (variable identifier)

The use and action of INPUT statements is illustrated in the following program examples.

(1) This program finds the square and cube of any number you input from the keyboard.

```
10 REM *Square and cube of any no. n input from
   keyboard *
20 INPUT n
30 PRINT n*n,n^3
```

When we execute the program by typing
RUN [ENTER]
the computer steps over the REMark statement 10 and executes 20 the input statement. At this point a ? is displayed and the cursor reappears on the screen indicating that it is waiting for us to enter in the value we wish to assign to *n*. Enter a number, e.g. 34.8:

```
? ■ 34.8 [ENTER]
  ↑
cursor
```

Program execution continues immediately and line 30 computes and displays the results for 34.8^2 and 34.8^3:
1211.04 and 42144.192.

The following version makes the program much more informative to the user by including a string message in the input statement:

```
10 REM * To calculate square and cube of a
   number *
20 INPUT "Enter number, n=";n
30 PRINT n*n,n^3
```

On running the program, line 20 now instructs the computer to display the words within the quotation marks (the string message) followed by the cursor, i.e.
Enter number, n = ? ■
Now enter in a number, followed as always by pressing the ENTER key. The results for n^2 and n^3 will be displayed immediately on the next line of the screen.

(2) This program acts as a conversion calculator for converting the weight in kilograms input from the keyboard to pounds.

```
10 REM *** To convert kilos to lbs ***
20 INPUT "Enter weight in kilos";kilos
30 PRINT kilos;"kg =";2.2046*kilos;"lbs"
```

On running the program:

```
RUN [ENTER]
Enter weight in kilos? 8 [ENTER]        . . . (8 entered)
8 kilos = 17.6368 lbs        . . . (result displayed)
```

(3) We can input more than one value by either using a series of input statements, e.g.

```
10 INPUT "ENTER VALUE FOR A";A
20 INPUT "ENTER VALUE FOR B";B
30 INPUT "ENTER VALUE FOR C";C
40 PRINT "A+B+C =";A+B+C
```

or more concisely using a single input line statement:

```
10 INPUT "VALUES FOR A,B,C=";A,B,C
20 PRINT "A+B+C =";A+B+C
```

On running the first version, the program execution pauses at each of the line statements, 10, 20 and 30 awaiting for us to enter values for *A*, *B*, *C* respectively, i.e.

```
RUN [ENTER]
ENTER VALUE FOR A? 47 [ENTER]          . . .(47 entered for A)
ENTER VALUE FOR B? 42 [ENTER]          . . .(42 entered for B)
ENTER VALUE FOR C? 18 [ENTER]          . . .(18 entered for C)
A + B + C = 107        . . .result displayed
```

On running the single input statement version, the computer pauses at line 10 and we enter *the values for A, B, C in order and separating each value by a comma,* i.e.

```
RUN [ENTER]
VALUES FOR A, B, C, = ? 47, 42, 18 [ENTER]
A + B + C = 107
```

(4) To find the volume of (for example) a cylinder, using INPUT statements to input the dimensions.

```
10 REM *** Volume of cylinder ***
20 CLS
30 INPUT "Enter radius, r=";r
40 INPUT "Enter height, h=";h
50 PRINT "Volume =";PI*r^2*h
60 PRINT "for r=";r,"h=";h
```

```
RUN [ENTER]
Enter radius, r = ? 2.4 [ENTER]
Enter height, h = ? 6.8 [ENTER]
Volume = 123.049901
For r = 2.4, h = 6.8        . . .display of results obtained
```

3.3 READ . . . DATA STATEMENTS

If we wish to use data from within the program rather than input values from the keyboard, which for a long list may tend to be very tedious, we can use READ . . . DATA statements. For example, suppose we wish to incorporate a list of prices of several items in a program, we can utilise READ . . . DATA statements as follows:

```
100 READ itemname$,number,cost,ref.no$
110 DATA desk,32,68.75,GH473-01
```

The values of the variables itemname$, number, cost, ref.no$ in the READ statement 100 are assigned the values in the *exact corresponding order* as given in the DATA statement of 110. So after execution of line 100, data would have been taken from the DATA statement and assigned as follows:
itemname$ = "desk" , number = 32 , cost = 68.75
ref.no$ = "GH473 – 01". [Note values for string variables need not be enclosed within quotation marks in a DATA statement.]

Each READ statement in a program consists of READ followed by a list of variable identifiers, each identifier being separated from the next by a comma.

Each DATA statement is a list of values and/or expressions, each value separated being separated by a comma. DATA statements may be put anywhere you wish in a program—normally at the beginning or at the end if this is convenient—since the computer ignores any DATA statements until it meets a READ statement.

The first time the computer in executing the program meets a READ statement it takes for the first variable value the first data value from the DATA list, for the second variable the second data value, and so on, working its way through the whole of the DATA statement lists.

If the number of items in the READ and DATA statements do not match, e.g. if there are more variables in the READ statements than values in the DATA statements, then the computer will display an error message immediately it tries to READ a value which is not there.

Program examples for READ-DATA statements

(1) This simple program illustrates the basic function of READ . . . DATA statements

```
5 CLS
10 READ A,B,C
20 PRINT "FIRST TERM A=";A
30 PRINT "SECOND TERM B=";B
40 PRINT "THIRD TERM C=";C
50 DATA 111,222,333
```

RUN [ENTER]
FIRST TERM A = 111
SECOND TERM B = 222
THIRD TERM C = 333 . . .display obtained on screen, showing that *A* is assigned the first value in the DATA statement list, *B* the second and *C* the third.

(2) This program works out the average value of a number of items listed in the DATA statements.

```
10 REM*** To find average of list of values ***
20 CLS
30 READ a,b,c,d,e,f,g
40 READ h,i,j,k,l,m,n
50 sum=a+b+c+d+e+f+g
60 sum=sum+h+i+j+k+l+m+n
70 PRINT "average =";
80 PRINT USING"##.##";sum/14
90 DATA 23,87,90,65,43,6,82
100 DATA 55,87,52,68,8,29,80
```

RUN [ENTER]

average = 55.36 . . .display of results obtained
(3) This program uses READ . . . DATA statements to assign data to a list of items. It then works out sub-total and total cost of all items and displays the list and results on the screen.

```
10 REM ** List cost program **
20 CLS
30 PRINT "ITEM";TAB(12)"NUMBER";
40 PRINT TAB(20)"UNIT COST";TAB(30)"SUB-TOTAL"
50 PRINT
60 READ item1$,no1,unitcost1
70 cost1=no1*unitcost1
80 PRINT item1$;TAB(13)no1;
90 PRINT TAB(20)unitcost1;TAB(30)cost1
100 READ item2$,no2,unitcost2
110 cost2=no2*unitcost2
120 PRINT item2$;TAB(13)no2;
130 PRINT TAB(20)unitcost2;TAB(30)cost2
140 READ item3$,no3,unitcost3
150 cost3=no3*unitcost3
160 PRINT item3$;TAB(13)no3;
```

```
170 PRINT TAB(20)unitcost3;TAB(30)cost3
180 sum=cost1+cost2+cost3
190 PRINT
200 PRINT "total sum = £";sum
210 DATA desk units,126,52.19
220 DATA tables,550,39.66
230 DATA chairs, 1216,9.95
```

On running the program you will obtain the following display on the screen:

```
ITEM          NUMBER   UNIT COST SUB-TOTAL

desk units     126      52.19      6575.94
tables         550      39.66      21813
chairs         1216     9.95       12099.2

total sum = £ 40488.14
```

3.4 THE USE OF THE STOP STATEMENT AND THE CONTinue COMMAND

Frequently we may wish to stop program execution at a given line and, for example, make a check on the results so far obtained before allowing the execution to continue. This is especially useful in checking intermediate results in a lengthy program.

For stopping execution of a program at a given line, BASIC provides the STOP statement:

100 STOP

This statement stops execution of the program at line 100 and will return the computer to the command mode.

The CONTinue command allows a program which has been interrupted by a STOP statement (or by a BREAK action*) to be resumed at the next line after the STOP statement. For example, consider the

```
10 REM *** STOP and CONTinue demo ***
20 CLS
30 READ A,B
40 PRINT (A-32)*5/9,B*4.2
50 STOP
60 READ C,D,E
70 PRINT (C+D)*A-E
```

*To ‘BREAK’ program execution, press ESCape key twice. This action will return cursor to screen.

```
80 STOP
90 PRINT"excution CONTinues"
100 x= C-D+B/e
500 DATA 70,450,12.3,67,90,23.6
510 DATA 67.9,90.87,32,76,89.08
```

```
RUN [ENTER]
21.1111111 1890
Break in 50
CONT [ENTER]
5461
Break in 80
```

On running the program, execution starts from the beginning, READs in DATA for *A, B* at line 30, outputs results at line 40 and then is stopped by the STOP statement at line 50. Program execution can be restarted from line 60 by typing in the direct mode command CONT. A further READ is made at line 60, results output at line 70 and then execution stopped at line 80 by a second STOP statement.

3.5 USE OF THE RESTORE STATEMENT WITH READ . . . DATA

In the 'normal' use of READ . . . DATA statements the computer works from the beginning of the first DATA statement (the one with the lowest line number) when it meets the first READ statement and then subsequently works through all the DATA lists until it reaches the end.

This sequence can be 'restored' by using RESTORE statements which have the form:

100 RESTORE
100 RESTORE (line number)

The first RESTORE statement (with no line number after RESTORE) forces any subsequent READ statement to start reading data from the first value of the first DATA statement (lowest number) in the program. When a specified line number is given after RESTORE, then any subsequent READ statement takes its data from the first DATA statement at or immediately following the specified line number in the RESTORE statement.

For example, let us work through the following program:

```
10 REM *** RESTORE demo ***
20 CLS
30 READ A,B,C
40 PRINT A,B,C
50 RESTORE
60 READ X,Y,Z
```

```
70 PRINT X,Y,Z
80 READ A,B,C
90 PRINT A,B,C
100 DATA 1,2,3,4,5,6
```

```
RUN [ENTER]
1              2              3         . . .(display of A, B, C at line 40)
1              2              3         . . .(display of X, Y, Z at line 70)
4              5              6         . . .(display of A, B, C at line 90)
```

At line 30 the READ statement assigns $A = 1$, $B = 2$ and $C = 3$, i.e. the first three values in the DATA list of line 100. Thus at line 40 we obtain the display:

```
1              2              3
```

Line 50, the RESTORE statement, returns us to the beginning of the DATA list. Thus at line 60, the READ statement assigns $X = 1$, $Y = 2$, $Z = 3$ and line 70 produces the display (again):

```
1              2              3
```

At line 80, however, the READ statement 'pointer' is at the fourth item in the DATA list and hence this READ statement assigns $A = 4$, $B = 5$, $C = 6$ and this is confirmed by the action of display statement 90:

```
4              5              6
```

Finally, consider a second example:

```
10 REM *** SECOND RESTORE demo ***
20 CLS
30 READ A,B,C
40 PRINT A;B;C,(A+B+C)/3
50 RESTORE 2000
60 READ A,B,C,D
70 PRINT A;B;C;D,(A+B+C+D)/4
80 RESTORE 1000
90 READ A,B,C,D,E,F,G
100 READ H,I,J,K,L,M
110 PRINT A;B;C;D;E;F;G;
120 PRINT H;I;J;K;L;M
1000 DATA 1,2,3,4,5,6
2000 DATA 10,20,30,40
3000 DATA 100,200,300
```

```
RUN [ENTER]
1  2  3          2
10   20   30   40          25
1 2 3 4 5 6 10 20 30 40 100 200 300
```

is the display obtained.

At line 50 the RESTORE statement 'restores' the read pointer to the first value in DATA statement 2000. Thus the READ statement at line 60 assigns $A = 10$, $B = 20$, $C = 30$, $D = 40$. In the absence of 50 RESTORE 2000,DATA would have been taken from the next value in line 1000, i.e. $A = 4$, $B = 5$, $C = 6$ and then from line 2000, $D = 10$.

At line 80, the RESTORE 1000 statement returns the read pointer to the beginning of DATA list 1000, so the READ statement 100 starts reading in its data from the first value in 1000 and continuing with 2000 to work right through the complete DATA statements.

Exercise problems 3

(1) Write a program using READ DATA statements to find the average value of the following 20 numbers:
52, 14, 37, 67, 73, 11, 75, 89, 19, 24, 61, 49, 12, 33, 47, 55, 16, 71, 81, 92.

(2)The following program using the INPUT statement could also be used to find averages:

```
10 REM ** TO FIND AVERAGE OF VALUES INPUT FROM KEYBOARD**
20 INPUT "ENTER VALUE"; VALUE
30 COUNT = COUNT + 1
40 SUM = SUM + VALUE
50 PRINT "AVERAGE SO FAR ="; SUM/COUNT
60 GOTO 20
```

Try running the program.

This program has a GOTO statement at line 60 and in fact will never stop! This statement 'jumps' execution back to 20 so the whole sequence 20, 30, 40, 50 is repeated followed again by the GOTO 20 at line 60 which returns execution to 20, forming a closed loop; bad programming practice! When you finished entering values you can BREAK the loop and return to the command mode by pressing the ESCape key twice.

BASIC has better ways of exiting loops: to be considered in the next chapter.

(3) Using INPUT statements write programs to convert
(a) feet and inches to metres (1 inch = 2.54 cm.)
(b) pounds and ounces to kilograms (1lb = 0.4536 kg).

4

DECISION MAKING, REPETITION, JUMPING AND SUBROUTINES

4.1 INTRODUCTION AND SUMMARY

So far, in virtually all the programs we have considered, the computer executes the individual line statements in exact ascending sequence, e.g. line statement 10 before 20, 20 before 30, and so on.

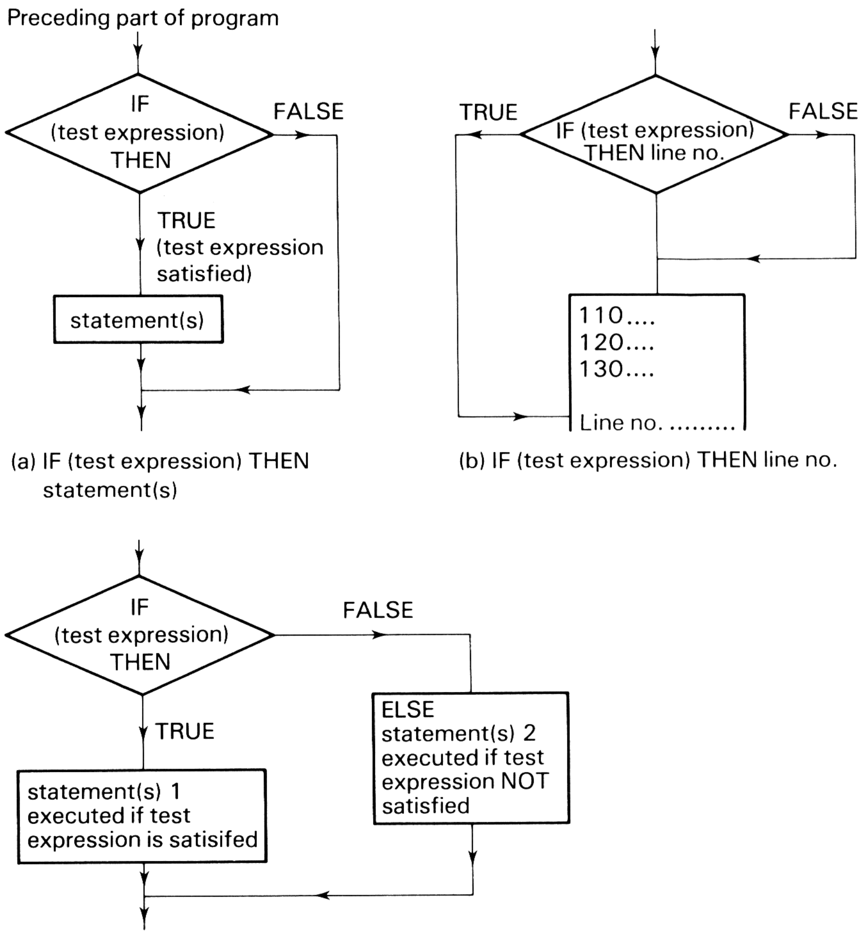
We now consider the BASIC statements available for the important tasks of
(1) making decisions as to alternative courses of action and 'jumping' within a program
(2) repeating a section of program statements a controlled number of times
(3) repeating a group of statements while some condition is satisfied
(4) selection of one of several different courses of action
(5) using subroutines

We commence the chapter by explaining the use of IF statements and test expressions used in making decisions. We then consider the FOR and WHILE constructs used for repetition and the use of Boolean logic expression for forming more comprehensive test condition expressions. Finally we consider the use of GOTO and GOSUB. In all cases the application of these statements is illustrated in practical program examples.

4.2 IF STATEMENTS FOR CHOICE OF ACTION

The choice of one or two different courses of action in a program is made using statements of the form:
100 IF (test expression) THEN (statement(s) to be executed)
100 IF (test expression) THEN (statement line number)
100 IF (test expression) THEN (statement(s)) ELSE (statement(s))
100 IF (test expression) THEN (line no. N1) ELSE (line no. N2)

The IF . . . THEN and IF . . . THEN . . . ELSE statements are the fundamental 'decision-making' statements in BASIC. Flowcharts illustrating their action are shown in fig. 4.1. IF the test expression is satisfied, i.e. yields a TRUE



(a) IF (test expression) THEN statement(s)

(b) IF (test expression) THEN line no.

(c) IF (test expression) THEN (statement(s)1) ELSE (statement(s)2)

Fig. 4.1 Flowchart illustration of IF statement actions.

value, THEN the statement(s) following THEN are executed; if a line number follows THEN, execution is transferred to this line and any interleaving lines are skipped over. The action of the IF . . . THEN . . . ELSE statement is very similar. IF the test expression is satisfied THEN the statement(s) following THEN are executed or program execution is transferred to line number 'N1' if a line number follows THEN. IF the test expression is not satisfied, i.e. yields a FALSE value, THEN the statement(s) following ELSE are executed or program execution is transferred to line number 'N2'.

The *test expressions* used in making the decisions are formed using the comparison operators listed in table 4.1 below. Comparison expressions can also be used in conjunction with the Boolean operators (AND, OR, NOT—see section 4.5) for forming test expressions.

Table 4.1 Table of comparison operators, symbols and meaning

Symbol	Meaning
=	equality; e.g. the expression $A = B$ checks whether the value of the left-hand term A is equal to the value of the right-hand term B
<>	inequality; e.g. $A<>B$ checks whether A is unequal to B
<=	less than or equal to; e.g. $A<=B$ checks whether A is less than or equal to B
>=	greater than or equal to; e.g. $A>=B$ checks whether A is greater than or equal to B
<	less than; e.g. $A<B$ checks whether A is less than B
>	greater than; e.g. $A>B$ checks whether A is greater than B

Comparison expressions such as $A>B$, $A<B$, $X<>Y$ . . . etc. always provide one of two results, either TRUE or FALSE. In CPC464 BASIC a comparison expression which is satisfied and therefore yields a TRUE result is assigned the numerical value −1; an expression yielding a FALSE result is assigned the value of 0.

IF statements: examples of use

(1) This example shows the IF statements being used with a string variable in the test expression. Try running the program. IF you enter red or green or amber the appropriate IF statement will be actioned.

```
10 REM *** IF...THEN demo 1 ***
20 CLS
30 INPUT "Enter traffic light colour";light$
40 IF light$="red" THEN PRINT "STOP"
50 IF light$="green" THEN PRINT "GO"
60 IF light$="amber" THEN PRINT "CAREFUL  !"
```

(2) A simple example of the IF . . . THEN . . . ELSE statement.

```
10 REM *** IF...THEN...ELSE demo ***
20 CLS
30 INPUT "Enter your temperature";temp
40 IF temp>102 THEN 50 ELSE 70
50 PRINT "send for doctor"
60 END
70 PRINT "take 2 aspirins and stay in bed"
```

(3) A more comprehensive and useful example which you can adapt to write your own program for 'finding addresses'.

Note the use of END. This effectively 'ends' execution of the program after carrying out its required task, i.e. finding the address from the name you input from the keyboard. If you do not include END, then execution will proceed at the next line and continue until it meets an END statement or the end of the program.

```
10 REM*** IF...THEN demo 2 ***
15 CLS
20 PRINT "finding Addresses":PRINT
30 n1$="Fred":n2$="Joe"
40 n3$="Alice":n4$="Rose"
50 PRINT "Addresses are availabe for:":PRINT
60 PRINT n1$,n2$,n3$,n4$
70 PRINT"To find address enter Christian name"
80 INPUT name$
90 PRINT
100 IF name$="Fred" THEN 1000
200 IF name$="Joe" THEN 2000
300 IF name$="Alice" THEN 3000
400 IF name$="Rose" THEN 4000
500 PRINT "name not listed"
510 END
1000 PRINT "Fred Smith"
1010 PRINT "16 Orange Avenue"
1020 PRINT "Chelmsford, Essex"
1030 END
2000 PRINT "Joe White"
2010 PRINT "Christmas Hotel, Malta, MZ 29"
2020 END
3000 PRINT "Alice Springs"
3010 PRINT "Shaw-Hoo House"
3020 PRINT "West Road, Hastings"
3030 END
4000 PRINT "Rose Browne"
4010 PRINT " Block 20, Hawke's Bay, Kent"
4020 END
```

4.3 REPETITION A CONTROLLED NUMBER OF TIMES: THE FOR LOOP

FOR type statements, or really FOR loops, provide the means for repeating the actions performed by a group of line statements a controlled number of times.

The FOR loop takes the following form:

100 FOR (variable identifier) = (start value) TO (end value)
110 }
120 } group of statements to be repeatedly executed
130 }
170 NEXT (variable identifier)

On entering the FOR loop the *variable identifier* (control variable for the loop) is set to the *start value*; the group of statements terminated by NEXT *(variable identifier)* are executed with this value and when completed the value is incremented by 1 and the process repeated continuously until the control variable value reaches the *end value*. By including STEP in the first line, i.e.
100 FOR (identifier) = (start val.) TO (end val.) STEP (increment)
the control value may be incremented by any value: whole, decimal, positive or negative rather than by just 1.

A flowchart diagram illustrating the action of the FOR loop is shown in fig. 4.2.

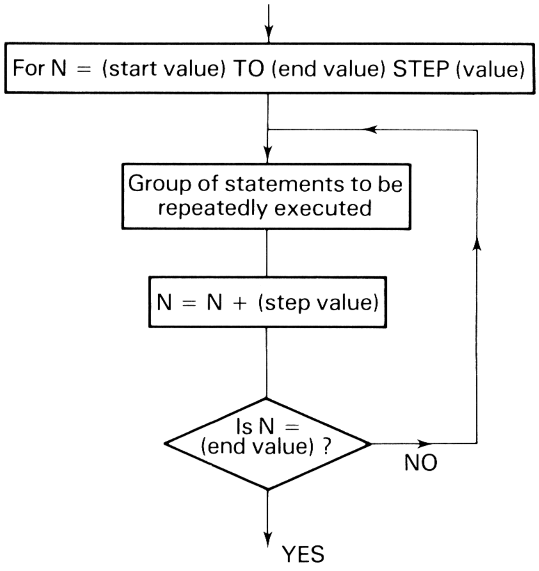

For N = (start value) TO (end value) STEP (value)
Group of statements to be repeatedly executed
N = N + (step value)
Is N = (end value) ?
NO
YES

Fig. 4.2 Flowchart illustration of FOR loop action.

FOR loops: examples of their use

(1) On running the following simple FOR loop program:

```
10 REM *** FOR loop demo 1 ***
20 CLS
30 FOR n=1 TO 6
40 PRINT n;"Coffee please !"
50 NEXT n
```

i.e. RUN [ENTER]
you will obtain the display

```
1 Coffee please !
2 Coffee please !
3 Coffee please !
4 Coffee please !
5 Coffee please !
6 Coffee please !
```

Statement 30 instructs that *n* should be changed consecutively from 1 to 6 in steps of 1. Thus statement 40 is executed 6 times with *n* incremented by 1 at the end of each cycle.

(2) This program utilises the FOR loop to display a table of squares, cubes and fourth powers of the numbers from *n* = 1 to 10.

```
10 REM *** FOR loop demo 2 ***
20 MODE 2
30 PRINT "n","n^2","n^3","n^4"
40 PRINT
50 FOR n=1 TO 10
60 PRINT n,n^2,n^3,n^4
70 NEXT n
80 FOR delay=1 TO 10000:NEXT delay
90 MODE 1
```

On typing,
RUN [ENTER]
we obtain the display:

```
n                n^2              n^3              n^4

  1                1                1                1
  2                4                8                16
  3                9                27               81
  4                16               64               256
  5                25               125              625
  6                36               216              1296
  7                49               343              2401
  8                64               512              4096
  9                81               729              6561
  10               100              1000             10000
```

Note, we have used mode 2 (line 20), the high resolution mode with 80 columns. This ensures that the output of statement 60 is contained on a single line of the screen. We have also used at line 80 the FOR loop as a delay so as to hold the display for 10 or so seconds before changing back to mode 1 with the subsequent clearance of the screen.

(3) This program illustrates the use of STEP. The FOR loop statement at line 40 uses a 0.1 step value; the second FOR loop at line 120 uses a negative step value of −0.5

```
10 REM *** FOR demo with STEP ***
20 PRINT "x","x^2","x^3"
30 PRINT
40 FOR x=0 TO 0.7 STEP 0.1
50 PRINT x,x^2,x^3
60 NEXT x
70 PRINT
80 PRINT "With negative step value"
90 PRINT
100 PRINT "n","1/n"
110 PRINT
120 FOR n= 5 TO 1 STEP -0.5
130 PRINT n,1/n
140 NEXT n
```

On running the program, you will obtain the following display:

```
x             x^2           x^3

 0             0             0
 0.1           0.01          0.001
 0.2           0.04          0.008
 0.3           0.09          0.027
 0.4           0.16          0.064
 0.5           0.25          0.125
 0.6           0.36          0.216

With negative step value

n             1/n

 5             0.2
 4.5           0.222222222
 4             0.25
```

```
3.5                0.285714286
3                  0.333333333
2.5                0.4
2                  0.5
1.5                0.666666667
1                  1
```

(4) This program utilises the FOR loop to draw up a table showing how a given sum invested at a given rate grows annually over a period of 10 years.

The program is set with
p = sum invested = £100
r = interest rate, % = 8.25
and uses the compound interest formula:
$a = p(1 + r/100)^n$
where a = amount = sum invested + interest
n = number of years

```
10 REM *** compound interest calc. ***
20 CLS
30 PRINT "year","interest","total"
40 PRINT
50 FOR n=1 TO 15
60 p=100:r=8.25:a=p*(1+r/100)^n
70 PRINT n,USING"####.##";a-p;
80 PRINT TAB(26);USING"###.##"; a
90 NEXT n
```

On running the program with the above values you will obtain the following display:

```
year            interest      total
 1                 8.25      108.25
 2                17.18      117.18
 3                26.85      126.85
 4                37.31      137.31
 5                48.64      148.64
 6                60.90      160.90
 7                74.18      174.18
 8                88.55      188.55
 9               104.10      204.10
 10              120.94      220.94
 11              139.17      239.17
 12              158.90      258.90
```

```
13                 180.26      280.26
14                 203.38      303.38
15                 228.41      328.41
```

Try modifying the program so you can input your own values for *p* and *r*. For example, include

```
24 INPUT "Enter sum invested" ; p
28 INPUT "Rate of interest" ; r
```

and change line 60 to

```
60 a = p*(1+r/100) ↑ n
```

4.4 REPETITION UNDER CONDITIONS: THE WHILE LOOP

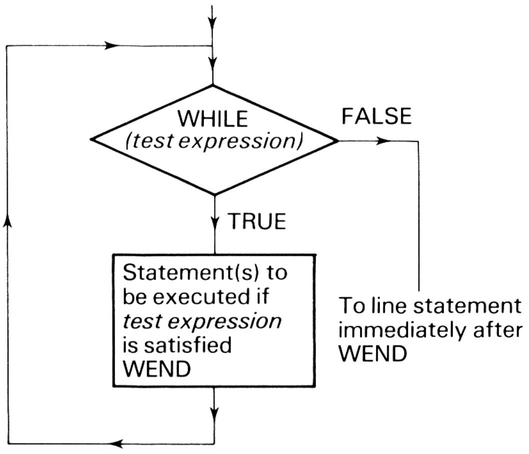

WHILE
(test expression)
FALSE
TRUE
Statement(s) to be executed if *test expression* is satisfied WEND
To line statement immediately after WEND

Fig. 4.3 Flowchart illustration of WHILE loop.

CPC464 BASIC provides a very useful form of repetitive control which allows a group of statements to be repeated WHILE a test condition is satisfied.

The WHILE loop takes the form:

```
100 WHILE (test condition expression)
110 )      group of statement(s) to be
120 }      executed WHILE test expression
130 )      is satisfied
 .
 .
 .
170 WEND
180 . . .
```

The test condition expression following WHILE determines whether or not the statement(s) grouped between WHILE and WEND are executed. This expression is evaluated at the beginning of each cycle. If it yields a TRUE value the following statements are executed, if not they are skipped and execution transfers to the statement immediately following WEND. Remember the statements between WHILE and WEND should eventually produce a result to alter the test condition expression to FALSE, otherwise your program will be trapped in the loop forever!

Fig. 4.3 shows a flowchart representation of the WHILE loop.

WHILE loops: examples of their use

(1) The following program uses the WHILE . . . WEND loop to READ fifty marks from the DATA statements. WHILE $n<50$ it READs and sums the marks. After READing the fiftieth data value it exits the loop and displays the mark average.

```
10 REM *** WHILE...WEND demo ***
20 CLS
30 n=0:sum=0
40 WHILE n<50
50 READ mark
60 sum=sum + mark
70 n=n+1
80 WEND
90 PRINT "Aveage mark =";USING"##.##";sum/50
100 DATA 67,83,23,78,45,12,88,43,39,10
110 DATA 23,41,65,9,76,22,18,67,90,76
120 DATA 55,98,0,43,12,87,44,31,74,11
130 DATA 81,54,78,25,54,89,67,50,23,14
140 DATA 66,99,41,36,12,54,82,45,36,89
```

(2) This program is a minor modification of the one above.

It contains two additional statements in the WHILE loop to count the number of 'passes' and 'failures' (see lines 52 and 56). Note for both programs that the test condition at line 40 must be 'evaluatable' before entry to the WHILE loop for the first time. For this reason we initialise *n* to zero (see line 30). Also ensure all other variables, i.e. sum, pass, fail, are initialised before entry to the loop.

```
10 REM *** WHILE...WEND demo ***
20 CLS
30 n=0:sum=0
35 pass=0:fail=0
40 WHILE n<50
50 READ mark
52 IF mark>=40 THEN pass=pass+1
```

```
56 IF mark<40  THEN fail=fail+1
60 sum=sum + mark
70 n=n+1
80 WEND
90 PRINT "Average mark =";USING"##.##";sum/50
92 PRINT "no.of passes =";pass
96 PRINT "no. of failures =";fail
100 DATA 67,83,23,78,45,12,88,43,39,10
110 DATA 23,41,65,9,76,22,18,67,90,76
120 DATA 55,98,0,43,12,87,44,31,74,11
130 DATA 81,54,78,25,54,89,67,50,23,14
140 DATA 66,99,41,36,12,54,82,45,36,89
```

On running the program you will obtain (for the above data):

```
Average mark = 50.50
no. of passes = 32
no. of failures = 18
```

(3) This simple program utilises the WHILE loop to keep a running total of values entered from the keyboard. Exit from the WHILE loop is obtained by entering 0.

```
10 REM *** Simple WHILE loop demo ***
20 CLS
30 sum=0
35 PRINT TAB(28)"sum so far"
40 INPUT "First vaue";x
50 WHILE x<>0
60 sum=sum+x
70 PRINT TAB(28)sum
80 INPUT "next value";x
90 WEND
```

Try running the program. You will find that the 'running' total is displayed from column 28 (for mode 1, approximately three-quarters across the screen to the right).

The following program is a modification of the above. It provides running total output displays to the window on the screen (the window being defined in line 20, from column 28 to 40 and from top row 1 to row 25 of the screen) and also a hardcopy output to the printer. These outputs are channelled via #5 for the window (see lines 30 and 70) and via the printer channel #8 for the printer (see lines 35 and 75).

```
10 CLS
20 WINDOW #5,28,40,1,26
30 PRINT #5,"sum so far"
35 PRINT #8,"sum so far"
40 INPUT "first value";x
50 WHILE x<>0
60 sum=sum+x
70 PRINT #5,sum
75 PRINT #8,sum
80 INPUT "next value";x
90 WEND
```

4.5 BOOLEAN OPERATORS AND LOGIC EXPRESSIONS FOR FORMING TEST CONDITIONS

In the IF . . . THEN statement it is often very useful to use what are called *Boolean expressions* to instruct the computer to make decisions. 'Boolean' refers to a very simple form of logic and is a method of joining together two or more conditions to form a decision-making statement.

Boolean-logic expressions are used essentially to form the *test expressions* for decision making in IF . . . type statements.

A Boolean expression, just like the comparison expressions considered in section 4.2, can take only one of two values: TRUE or FALSE. In CPC464 BASIC the TRUE value is numerically equal to 1 (or −1 for comparison expressions) and the FALSE value is 0.

In addition to the comparison operators, four Boolean or logic operators:
NOT, AND, OR, XOR
are used to create Boolean expressions, which are invaluable in forming test condition expressions. Their meaning is as follows:

NOT is the logical NOT or logical negation, e.g. NOT *A* is TRUE if *A* is FALSE and is FALSE if *A* is TRUE.

AND is the logical AND, e.g. *A* AND *B* is TRUE if and only if *A* and *B* are both TRUE; if *A* and/or *B* are FALSE, *A* AND *B* is assigned a FALSE value.

OR is the logical OR, e.g. *A* OR *B* is TRUE if either or both *A* and *B* are TRUE; if *A* and *B* are both false *A* OR *B* is assigned a FALSE value.

XOR is the logical EXCLUSIVE OR, e.g. *A* XOR *B* is TRUE if either *A* or *B* is TRUE; if *A* and *B* are both TRUE or both FALSE *A* XOR *B* is FALSE.

Precedence

The order of precedence of these operators in evaluating Boolean expressions is as follows:

highest NOT
AND
OR, XOR
=,<>,<=,>=,<,>

A simple Boolean expression consists of a series of Boolean values separated by AND, OR, XOR or preceded by NOT. For example, suppose *A, B, C, D* are variables which are assigned either TRUE or FALSE (i.e. either 1 or 0) values using comparison type expressions, then,
A AND *B* AND *C* AND *D* is TRUE if and only if *A, B, C, D* are all assigned TRUE values
A AND NOT *D* is TRUE if *A* is TRUE and *D* is FALSE
A OR *C* OR *D* is TRUE if one or more of the variables is assigned TRUE
C XOR *D* is TRUE if either *C* or *D* is TRUE, otherwise it is FALSE.

Examples

(1) The following program illustrates the meaning of the fundamental AND, OR and XOR operators. Enter various combinations of 0 and 1s for the variables *A* and *B*, the results of
F1 = A AND B
F2 = A OR B
F3 = A XOR B
are displayed.

```
10 REM *** Boolean expressions demo ***
20 CLS
30 INPUT "A and B, 0 or 1";A,B
40 PRINT:PRINT
50 PRINT "For A ="A;" and B =";B
60 PRINT:PRINT
70 PRINT "AND example:"
80 F1= A AND B
90 PRINT "F1 = A AND B=";F1
100 PRINT:PRINT
110 PRINT "OR example:"
120 F2= A OR B
130 PRINT "F2 = A OR B =";F2
140 PRINT:PRINT
150 PRINT "XOR example:"
160 F3= A XOR B
170 PRINT "F3 = A XOR B =";F3
```

(2) This example illustrates the formation of test conditions using a combination of Boolean and comparison operators (see lines 30 and 80). These expressions are used respectively in lines 50 and 90 as the test condition expression in IF statements. Try running the program to see how it works.

```
10 REM *** Boolean expression demo ***
20 INPUT "a,b";a,b
30 state1= a>10 AND b<20
40 PRINT "state1 =";state1
50 IF state1 THEN PRINT "OK" ELSE PRINT "Not
   satisfied"
60 PRINT:PRINT
70 INPUT "c,d";c,d
80 state2= c=100 OR d<=0
90 IF state2 THEN PRINT "state2 is satisfied"
```

(3) The following program illustrates the use of a logic expression in the control of a WHILE loop. The program could be regarded as a control simulation exercise for a machine or process, i.e. WHILE A = 1 AND B = 1 AND C = 1 keep the process running, but make checks on the *A*, *B*, *C* control values. If a fault occurs exit from the WHILE loop and display which fault(s) have occurred.

Once again try running the program.

```
10 CLS
20 PRINT "IF A,B,C TESTS O.K. ENTER 1"
30 PRINT "for each test state,IF NOT ENTER 0"
40 PRINT
50 A=1:B=1:C=1:REM Initial conditions set as OK
60 WHILE A=1 AND B=1 AND C=1
70 PRINT "ALL CONDITIONS O.K."
80 PRINT "CONTINUE RUNNING"
90 INPUT "A,B,C VALUES";A,B,C
100 CLS
110 WEND
120 IF A=0 THEN PRINT "FAULT A: CHECK OIL"
130 IF B=0 THEN PRINT "FAULT B: INSUFFICIENT
    COOLANT"
140 IF C=0 THEN PRINT "FAULT C:PUMP NOT WORKING"
```

4.6 GOTO COMMANDS AND STATEMENTS

In many program solutions we frequently wish to 'jump' the normal ascending order of line statement execution and transfer to a higher or lower line rather

than proceed to statement immediately following. We have already seen how this is done in IF and WHILE type statements. For example,
100 IF x>0 THEN 200
would transfer execution to line 200 if *x* is greater than 0. This statement can also be written including GOTO, i.e.
100 IF x>0 THEN GOTO 200
which has the identical effect. The inclusion of GOTO, however, is not necessary in IF statements.

GOTO is one of the two BASIC 'jump' instructions. It can be used as a direct mode command in a similar manner as RUN (line number) to start the execution of a program at a given line. For example,
GOTO 200 [ENTER]
would start the execution of a program at line 200.

Examples: use of GOTO

(1) This short program illustrates the use of GOTO for holding a display, without the 'Ready' caption reappearing. At the end of the program line 70,
70 GOTO 70
forms a continuous closed loop. When you wish the program to be halted, press the ESCape key twice. This will break program execution and return the cursor to the screen.

```
10 REM *** GOTO demo ***
20 MODE 0
30 LOCATE 4,12
40 PRINT "BUY THIS BOOK"
50 PRINT:PRINT
60 PRINT "***** R G M *****"
70 GOTO 70
```

(2) This program allows you to continually convert feet and inches to metres. At line 60 you are asked to INPUT the number of feet and inches. Remember to separate the 'ft' and 'in' values by a comma and as always press ENTER. The computer will then effect the calculation and display the result, whilst line 120:
120 GOTO 40
transfers you back for another calculation. To exit from the program ENTER −999 for 'ft' and any value for 'in'. The IF statement at line 70 will THEN END the program.

```
10 REM *** GOTO example ***
20 CLS
30 PRINT "Feet and inches to meters conversion"
40 PRINT
```

```
50 PRINT "enter no. of feet and inches"
60 INPUT ft,in
70 IF ft=-999 THEN END
80 m=(12*ft+in)*0.0254
90 PRINT ft;"feet";in;"inches =";
100 PRINT m;"meters"
110 PRINT "-----------------------------------"
120 GOTO 40
```

The ON . . . GOTO statement

We can also use the ON . . . GOTO statement:
ON (select variable) GOTO (line number 1) , (line number 2), . . .
This type of statement causes program execution to jump to one of the specified line numbers according to the value of the select variable. Transfer to the first line number specified after GOTO will be made if the value of the select variable value is 1; transfer to the second specified line number if the value is 2, and so on. Any number of line numbers may be specified provided they can be placed on the same logical line.
For example,
100 ON n GOTO 200, 300, 400
If the value of *n* is 1 then program execution will jump to line 200; if *n* = 2 execution jumps to 300; if *n* = 3 then execution jumps to 400; if we input a higher value for *n*, e.g. 4, program execution would proceed to the line immediately following the ON . . . GOTO statement.

The following program illustrates a practical example of the use of the ON . . . GOTO statement.

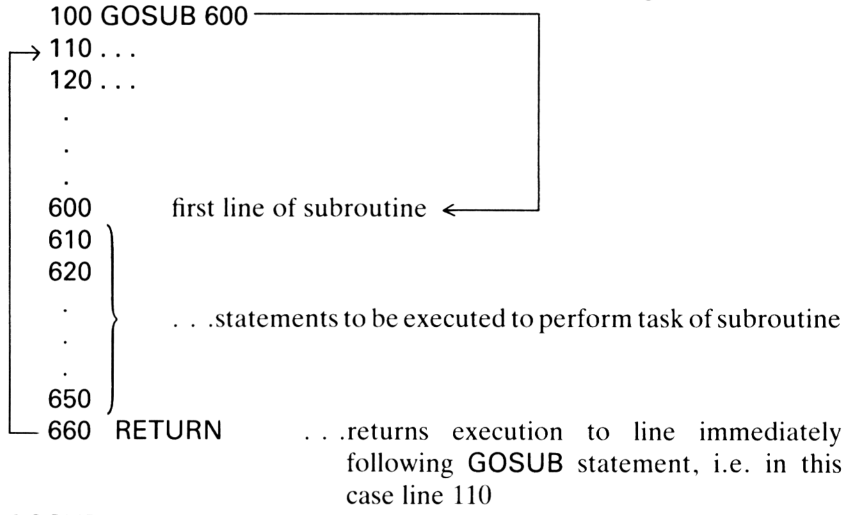
```
10 REM ***  ON... GOTO example  ***
20 CLS
30 PRINT " Ski Holiday Tarriffs "
40 PRINT "Resorts on file:"
50 PRINT "La Plagne...Enter 1"
60 PRINT "Verbier.....Enter 2"
70 PRINT "Soll........Enter 3"
80 PRINT "Les Arcs....Enter 4"
90 INPUT "your selection =";n
100 PRINT
110 IF n>4 THEN PRINT "Incorrect entry":GOTO 40
120 PRINT
130 ON n GOTO 140,170,200,220
140 PRINT "High season FF 2500 per week"
150 PRINT "Low season  FF 1700 per week"
160 END
170 PRINT "High season SF 1000 per week"
```

```
180 PRINT "Low season  SF 700 per week"
190 END
200 PRINT "All seasons AS 6600"
210 END
220 PRINT "December FF 2200 per week  "
230 PRINT "Christmas and New Year"
240 PRINT "special 10 day package FF 3200"
250 PRINT "Jan, Feb  FF 2400 per week"
260 END
```

4.7 GOSUB STATEMENTS AND SUBROUTINES

The action of GOSUB statements are similar to GOTO statements but contain in addition a RETURN. They are used to transfer program execution to a given section of the program which contains a subroutine designed to execute a certain task. After executing the subroutine execution is returned, by a RETURN statement which must form the last line of the subroutine, to the line statement immediately following the GOSUB statement. The use of subroutines have important advantages in programming. The subroutine need only be written once and can be 'called' into action as many times as is required in the main program.

The general form of a GOSUB . . . RETURN construct is given below


100 GOSUB 600
110 . . .
120 . . .
600 first line of subroutine
610
620
650
. . .statements to be executed to perform task of subroutine
660 RETURN
. . .returns execution to line immediately following GOSUB statement, i.e. in this case line 110

The GOSUB line number statement transfers processing of a program to the line number specified and is used, for example, to go to a subroutine which follows from that line number. A RETURN statement acts to terminate the subroutine statements and return processing back to the line statement immediately following the GOSUB statement.

We can also use the ON . . . GOSUB statement:

ON (select variable) GOSUB (line number 1) , (line number 2) , . . .

which is fundamentally the same as the ON . . . GOTO statement considered i
the last section, but differs in that program execution always returns to the firs
statement immediately following the ON . . . GOSUB statement afte
execution of the subroutine.
For example,

```
100 ON X GOSUB 500, 600
110 . . .
 .
 .
 .
500 REM *        subroutine 1 *
 .
 .
 .
590 RETURN: REM*        End of subroutine 1*
600 REM *        Subroutine 2 *
 .
 .
 .
750 RETURN : REM *        End of subroutine 2 *
```

When $X = 1$ at line 100, program execution jumps to the subroutine at line 50
and after execution of the subroutine returns to line 110, the line immediatel
following the ON . . . GOSUB statement.

If $X = 2$ at line 100, execution would then jump to execute the subroutin
commencing at line 600 and then return at the end of the subroutine (line 750
to line 110.

The following example illustrates a practical example of using GOSUB. Th
program plots a horizontal bar chart for the sales figures contained in DAT
statement, line 5000. The subroutine used to plot bars is contained in th
section lines 1000 to 1060.

```
10 REM *** Use of GOSUB: Bar Chart ***
20 CLS
30 PRINT "    5    10    15    20    25"
35 PRINT "------------------------------"
40 PRINT "U.K. sales":GOSUB 1000
50 PRINT "French sales":GOSUB 1000
60 PRINT "U.S.A. sales":GOSUB 1000
70 PRINT "German sales":GOSUB 1000
80 PRINT "African sales":GOSUB 1000
90 PRINT "Italian sales":GOSUB 1000
100 PRINT "Swedish sales":GOSUB 1000
110 PRINT "Australian sales":GOSUB 1000
120 END
```

```
1000 REM ** Subroutine for plotting bar"
1010 READ sales
1020 FOR n=1 TO sales
1030 PRINT "*";
1040 NEXT n
1050 PRINT "   $";sales;"x 100000"
1060 RETURN
5000 DATA 14,19,12,24,5,17,13,7
```

Here is the bar chart display obtained on running the program:

```
    5    10   15   20   25
---------------------------
U.K. sales
**************   $ 14 x 100000
French sales
*******************   $ 19 x 100000
U.S.A. sales
************   $ 12 x 100000
German sales
************************   $ 24 x 100000
African sales
*****   $ 5 x 100000
Italian sales
*****************   $ 17 x 100000
Swedish sales
*************   $ 13 x 100000
Australian sales
*******   $ 7 x 100000
```

EXERCISE PROBLEMS 4

(1) Write a program that will display a message only if the correct password 4401326019 is keyed in.

(2) Write a program that will print out a standard memo a given number of times. This number is to be input from the keyboard.

(3) Write a program that READs up to 50 values contained in data statements and determines their average.

(4) Write a program that will select five different courses of action depending on whether 1, 2, 3, 4 or 5 is keyed in.

(5) Write a subroutine that will determine the maximum and minimum values in a list of data contained in DATA statements. Incorporate this subroutine in a program which displays these values for the first 10, the first 20, and the total list.

5

STANDARD FUNCTIONS AND APPLICATIONS

5.1 INTRODUCTION AND SUMMARY

A wide variety of standard functions are provided in CPC464 BASIC. In thi: chapter we describe the form and explain the meaning and use of thos(functions used in handling numbers, for providing values for mathematica functions and also those used to process characters and strings. We als(consider how we can define and use our own 'user-defined' functions.

5.2 STANDARD FUNCTION FOR SQUARE ROOTS: SQR(X)

SQR(X), where *X* is any positive number or expression, will compute th(square root of the value *X* within the brackets. For example,

(1)

```
PRINT SQR (1197.16) [ENTER]
34.6          . . . square root of 1197.16 is displayed on screen
```

(2)

```
10 INPUT "Value of x";x
20 PRINT SQR(x)
```

```
RUN [ENTER]
Value of X = 57.76 [ENTER]          . . . note we enter 57.76
7.6          . . . square root of 57.76 displayed
```

(3)

```
10 INPUT "Values of a,b,c";a,b,c
20 x=a+b+c
30 y=b*c
40 PRINT x,y,x/y,SQR(x/y)
```

RUN [ENTER]
Values of a, b, c? 23.6, 8.96, 4.28 [ENTER]

```
36.84      38.3488      0.960656      0.980131
```

. . . results displayed, i.e. $x = a + b + c$, $y = b \times c$, x/y, $\sqrt{(x/y)}$

We can, of course, use the ↑ key to find any power or root (see also chapter 1, section 1.6):

PRINT 2.25 ↑ 0.5 . . . gives square root of 2.25
PRINT 46 ↑ 2 . . . gives square of 46
PRINT 112.6 ↑ 4 . . . gives 112.6 raised to the power of 4
PRINT 59.3 ↑ 0.68 . . . gives 59.3 raised to power 0.68
PRINT 48 ↑ − 1.8 . . . gives $48^{-1.8} = \frac{1}{48^{1.8}} = 9.41382E{-04}$

5.3 STANDARD FUNCTIONS FOR ABSolute, INTeger, ROUNDing AND SiGN OF NUMBERS

ABS(X) provides the absolute value (i.e. magnitude) of the value of the number or arithmetic expression *X* specified within the brackets.

For example,
PRINT ABS (44.67) . . . displays 44.67
PRINT ABS (−44.67) . . . displays 44.67, i.e. the magnitude of −44.67
PRINT ABS (32 + 49) . . . displays 81
PRINT ABS (32 − 49) . . . displays 17 , i.e. $|32-49| = |-17| = 17$

CPC464 BASIC provides four functions for 'rounding' numbers:

ROUND (x,r) provides the value of *x* to *r* decimal places.
For example,
PRINT ROUND (5.759345,2) gives
5.76 . . . i.e. *x* specified to 2 decimal places
PRINT ROUND (8.934527E−3,4) gives
0.0089 . . . i.e. *x* to 4 decimal places
PRINT ROUND (897.56*34.62/29.42,1)
1056.2 . . . i.e. *x* to 1 decimal place

INT(X) provides the value of *X* to the nearest *smallest* whole number.
For example,
PRINT INT (11.8) . . . displays 11
PRINT INT (−11.4) . . . displays −12
PRINT INT (8.4 * 6.3) . . . displays 52, as $8.4 \times 6.3 = 52.92$
PRINT INT (−8.4 * 6.3) . . . displays −53

FIX(X) removes any decimal part of *X* and returns the whole number part.
For example,
PRINT FIX (11.9) . . . returns 11
PRINT FIX (−11.9) . . . returns −11
Note INT (−11.9) returns −12, i.e. rounds to *smallest* whole number.

CINT(X) converts the value of *X* to a rounded whole number in the INTeger range −32768 to +32767 of the CPC464.

For example,
PRINT CINT (119.72) . . . returns 120
PRINT CINT (−119.72) . . . returns −120
PRINT CINT (98890.87) . . . returns "Overflow" message, i.e. 9889(exceeds the CPC464 INTeger range

Sign of numbers, SGN(X)
The SiGN function is used to determine the sign of the value of the variable oı numeric expression *X*. SGN(X) returns −1 if *X* is less than 0, returns 0 if *X* = 0, and returns +1 if *X* is greater than 0, e.g.
SGN (52.6) . . . returns +1
SGN (0) . . . returns 0
SGN (−10.7) . . . returns −1

5.4 GENERATION OF RANDOM NUMBERS

RND(N) generates a pseudo random number in the range 0 to 0.999 999 999. For example, each time the following program is run it generates 10 random numbers

```
10 FOR n = 1 TO 10
20 PRINT n,RND(22)
30 NEXT n
```

Here is a typical print-out:

```
1              0.182127052
2              0.729726442
3              0.526303542
4              0.671128625
5              5.51433E-02
6              0.598122045
7              1.17327E-02
8              0.177665838
9              0.596751153
10             0.814979649
```

Using the FIX or INT standard functions you can generate random whole numbers in any range as illustrated by the following program.

```
10 REM *** Generation of random nos ***
20 PRINT " 0 to 1";TAB(15)"0-99";
30 PRINT TAB(22)"0-499";TAB(32)"0-999"
```

```
40 PRINT
50 FOR n=1 TO 10
60 x=RND(n):x100=FIX(100*x)
70 x500=FIX(500*x):x1000=FIX(1000*x)
80 PRINT x;TAB(15);x100;
90 PRINT TAB(22);x500;TAB(32);x1000
100 NEXT n
```

Here is a typical display:

```
0 to 1          0-99    0-499        0-999

0.879326445      87      439          879
0.516902614      51      258          516
7.14947E-02      7       35           71
9.12086E-02      9       45           91
0.68626697       68      343          686
0.122103377      12      61           122
0.862517284      86      431          862
0.264316374      26      132          264
0.790767248      79      395          790
0.441823238      44      220          441
```

This program can be used to check the 'randomness' of the CPC464 random generator. It counts the number of 1s, 2s, 3s, 4s, 5s and 6s generated for a total of 600 random numbers. Your counts should be approximately 100 each. Try running the program, it only takes about 12 seconds.

```
10 REM *** Random number check ***
20 CLS
30 FOR n=1 TO 600
40 x=INT(6*RND(n)+1)
50 IF x=1 THEN c1=c1+1
60 IF x=2 THEN c2=c2+1
70 IF x=3 THEN c3=c3+1
80 IF x=4 THEN c4=c4+1
90 IF x=5 THEN c5=c5+1
100 IF x=6 THEN c6=c6+1
110 NEXT n
120 PRINT
130 PRINT "no. of 1's =";c1
140 PRINT "no. of 2's =";c2
150 PRINT "no. of 3's =";c3
160 PRINT "no. of 4's =";c4
170 PRINT "no. of 5's =";c5
180 PRINT "no. of 6's =";c6
```

Random numbers are extremely useful in making tests, e.g. sorting and statistics, which we consider later. They are also widely employed in games. Here is a simple example:

```
10 REM *** Simple game ***
20 CLS
30 PRINT"This is a simple game, you versus the CPC"
35 PRINT
40 PRINT"It involves tossing an imaginary die!"
45 PRINT
50 PRINT "Input what number between 1 AND 6"
60 PRINT"you think the CPC will generate"
65 PRINT
70 INPUT "My guess is";n
80 CPC.no =INT(RND(4)*6)+1
90 IF n = CPC.no THEN 120
95 PRINT
100 PRINT"Hard luck!";" the CPC no. was ";CPC.no
110 PRINT: GOTO 130
120 PRINT"Well done, your guess was right"
130 PRINT"If you want another go, type yes"
140 PRINT"if not, type no"
150 INPUT answer$
160 IF answer$="yes" THEN 70
170 END
```

5.5 TRIGONOMETRIC FUNCTIONS

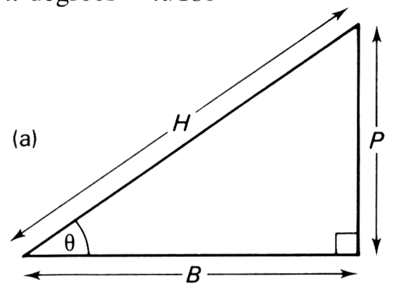
Standard functions are available for the following trigonometric functions:
SIN (x) . . . gives sin x
COS (x) . . . gives cos x
TAN (x) . . . gives tan x
ATN (x) . . . gives arc tan x or $\tan^{-1} x$, with x, with x given in radians within the range $-\pi/2$ to $+\pi/2$. The value of x (the default value) is in radians unless set to the degrees mode using DEG. If a DEG statement or command is used then x is set to degrees, e.g.

```
10 DEG
20 x=60
30 PRINT SIN(x),COS(x),TAN(x)
40 PRINT ATN(1):REM tan 45 degrees = 1
```

Statement 10 sets the degree mode, so the output from line 30 will give sine, cosine and tangent of 60°; line 40 will give arctan (1), i.e. 45°.

Once called into operation DEG sets the degree mode until instructed by CLEAR which clears all variables, or by using RAD, or more drastically by using NEW which deletes the entire program.

The RAD command or statement sets the radian mode. Remember the relationship between angles expressed in radians and degrees is
x radians = x degrees $\times \pi/180$



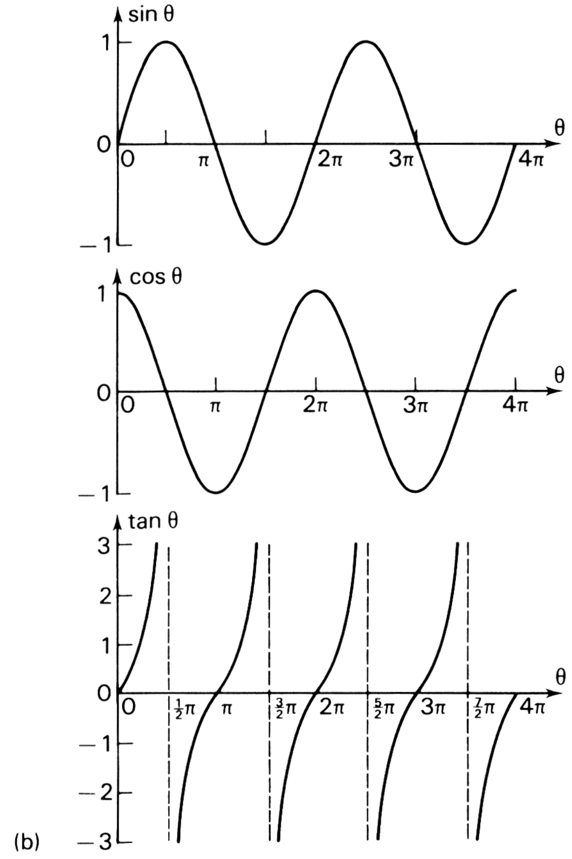


Fig. 5.1 The three basic trigonometric functions. (a) Definitions of sine, cosine and tangent of an angle:

$\sin\theta = \dfrac{P}{H}$, $\cos\theta = \dfrac{B}{H}$, $\tan\theta = \dfrac{P}{B}$

(b) Waveforms for sine, cosine and tangent.

The CPC464 stores the value of π to 8 decimal places as pi or PI, e.g.
print pi . . . returns the value 3.14159265

Fig. 5.1 gives the basic definitions and the waveforms for sin, cos and tan. Using the CPC464 graphics facilities we can easily display waveforms (see example 4 following in this section and chapter 6).

Trigonometric functions: program examples

(1) This program produces a table of sin *x* values from 0° to 360° in steps of 20°.

```
10 REM *** Table of sin x   ***
20 CLS
30 PRINT "x deg.";TAB(15)"sin x"
40 PRINT
50 DEG
60 FOR x=0 TO 360 STEP 20
70 PRINT x,SIN(x)
80 NEXT x
```

RUN [ENTER]

```
x deg.          sin x

 0              0
 20             0.342020143
 40             0.64278761
 60             0.866025404
 80             0.984807753
 100            0.984807753
 120            0.866025404
 140            0.64278761
 160            0.342020143
 180            0
 200           -0.342020143
 220           -0.642787609
 240           -0.866025404
 260           -0.984807753
 280           -0.984807753
 300           -0.866025404
 320           -0.642787609
 340           -0.342020143
 360            0
```

(2) This program applies the Sine Rule (see fig. 5.2) to calculate the sides BC and AB given the angles *A* and *C* and the third length AC.

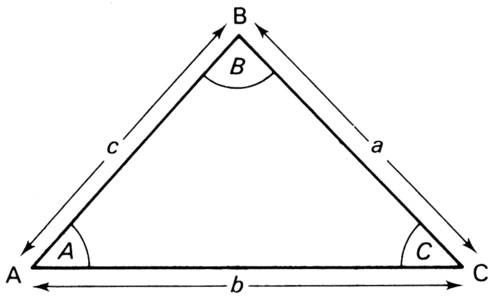


Fig 5.2 Sine Rule: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$

```
10 REM *** Sine Rule example ***
20 CLS
30 INPUT "enter two known angles";A,C
40 INPUT "known side length";AC
50 B=180-A-C
60 DEG
70 BC=SIN(A)*AC/SIN(B)
80 AB=SIN(C)*AC/SIN(B)
90 PRINT "BC =";ROUND(BC,2)
100 PRINT "AB =";ROUND(AB,2)
```

Try running the program, e.g. by inputting $A = 32°$, $C = 75°$ and AC = 64.4. The results obtained for the two sides are: BC = 35.69, AB = 65.05

(3) This program applies the second very useful rule, the Cosine Rule (see fig. 5.3).

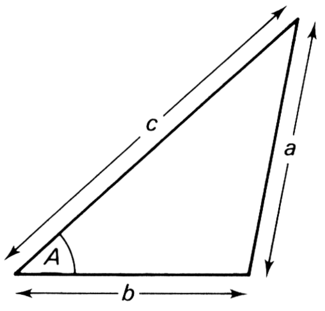


Fig. 5.3 Cosine Rule: $a^2 = b^2 + c^2 - 2bc \cos A$.

```
10 REM *** Application of Cosine Rule ***
20 CLS
30 INPUT "Two sides and included angle";b,c,A
```

```
40 DEG
50 BC=SQR(b^2+c^2-2*b*c*COS(A))
60 PRINT "a =";ROUND(BC,4)
```

On running the program with the following data: $b = 112$, $c = 58.6$, $A = 47°$, you will obtain $a = 83.8199$.

(4) This program pre-empts the next chapter on graphics but is fairly easy to understand. The origin is set by the **ORIGIN** statement to be in the centre of the screen. **PLOT** instructs the computer to plot points and **DRAW** to draw lines. This program plots a sinewave $y = \sin x$ from $-320°$ to $+320°$. The display obtained is shown in fig. 5.4.

```
10 REM ** drawing a sinewave **
20 CLS
30 PRINT "sinewave y=sin x from x=-320 to 320 degrees"
40 ORIGIN 320,200
50 PLOT -320,0
60 DRAW 320,0
70 PLOT 0,-200
80 DRAW 0,200
90 FOR x=-320 TO 320
100 DEG:y=SIN(x)
110 PLOT x,y*100
120 NEXT x
```

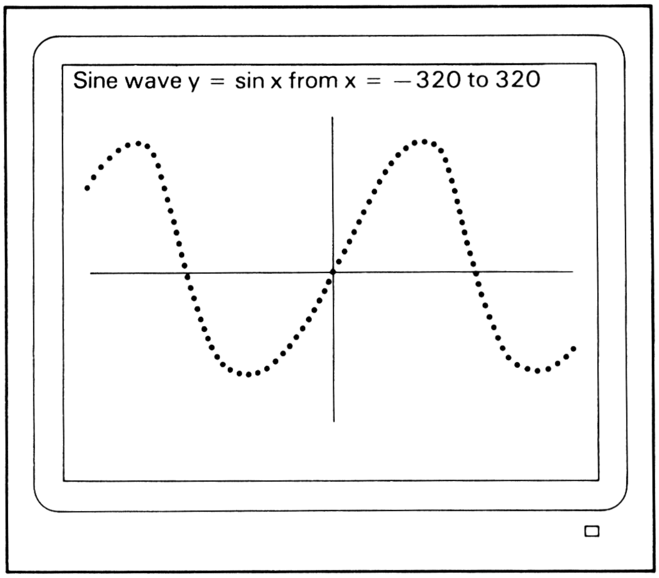

Sine wave y = sin x from x = −320 to 320

Fig. 5.4 Display obtained on running sinewave program of example 4.

5.6 EXPONENTIAL AND LOGARITHMIC FUNCTIONS

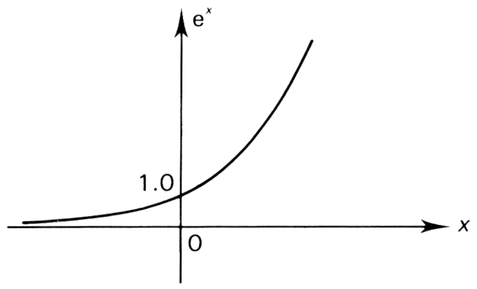


Fig. 5.5 Plot of exponential function e^x.

EXP(X) gives e raised to the power X, i.e. gives values of the exponential function, e^x where e = 2.7182818 . . .

LOG(x) gives the natural logarithm of x, where x must be greater than zero.

LOG10(x) gives the common logarithm (log to base 10) of x ; x must be greater than zero.

Exponential and log functions: examples

(1)

```
PRINT EXP (2.4) [ENTER]
11.0231764
```
. . . value of $e^{2.4}$ obtained

(2)

```
PRINT EXP(1.6) + EXP(−1.6) [ENTER]
5.15492894
```
. . . value of $e^{1.6}+e^{1.6}$ displayed

(3) The following program produces a table of $\log_e$ and $\log_{10}$ for $x = 1$ to 10. The log values have been rounded to 5 decimal places.

```
10 PRINT " x"," loge"," log10"
20 PRINT
30 FOR x=1 TO 10
40 PRINT x,ROUND( LOG(x),5),ROUND(LOG10(x),5)
50 NEXT x
```

```
x             loge          log10

1             0             0
2             0.69315       0.30103
3             1.09861       0.47712
4             1.38629       0.60206
```

```
5                  1.60944         0.69897
6                  1.79176         0.77815
7                  1.94591         0.8451
8                  2.07944         0.90309
9                  2.19722         0.95424
10                 2.30259         1
```

5.7 USER DEFINED FUNCTIONS

The BASIC language also allows us to define our own functions—user defined functions—as follows:
100 DEF FN identifier (parameter list) = expression for result
The function must have an identifier, just as a variable, and the parameter list is used to feed data values to be used in computing the function result, i.e. the value returned by the function.

A user defined function can be 'called' or 'invoked' at any time in a program simply by writing FN identifier (parameter list values). The function definition, however, should be written outside any loops in which it is called and certainly always before it is first used. Function definitions are therefore usually written early on a program.

The definition and function 'calls' are illustrated in the following examples.

User defined functions: program examples

(1) This program contains the definitions of two functions:
FNA(r) = $4\pi r^2$ and FNV(r) = $4\pi r^3/3$
to calculate respectively the surface area and volume of a sphere of radius *r*.

Calls are made to the functions: at line 50 to calculate the area for *r* = 5.6; at line 60 to calculate the volume for *r* = 24; and at line 100 to calculate area and volume for a value of *r* input from the keyboard.

```
10 REM *** User defined functions ***
15 REM * for surface area and volume of a sphere
20 CLS
30 DEF FNA(r)=4*PI*r^2
40 DEF FNV(r)=4*PI*r^3/3
50 PRINT "Area ="; FNA(5.6);" for r = 5.6"
60 PRINT "Volume =";FNV(24);" for r = 24"
70 PRINT:PRINT
80 INPUT "Radius =";r
90 PRINT "for r=";r
100 PRINT "Area =";FNA(r);"  Volume =";FNV(r)
```

(2) This program contains the function,
FNI (p,r,t)
which is used to calculate compound interest. The program calls the function in a FOR loop to calculate a table of interest values for $t = 1$ to 10 years; the values for the sum invested p and interest rate r are input from the keyboard.

```
10 REM *** User defined interest function ***
20 CLS
30 DEF FNI(p,r,t)=p*(1+r/100)^t-p
40 INPUT "Sum invested and rate %";p,r
50 PRINT "year","interest"
60 PRINT
70 FOR t=1 TO 10
80 PRINT t,ROUND(FNI(p,r,t),2)
90 NEXT t
```

The display obtain when inputting $p = 500$, $r = 12.5$ is

```
year              interest

 1                 62.5
 2                 132.81
 3                 211.91
 4                 300.9
 5                 401.02
 6                 513.64
 7                 640.35
 8                 782.89
 9                 943.25
 10                1123.66
```

5.8 STANDARD FUNCTIONS FOR STRINGS: STRING SLICING

BASIC provides also several useful functions for processing strings. In this section we consider string 'slicing' functions, i.e. functions that can remove characters from a string to form a smaller string.

The LEFT$ function

The LEFT$ function is used, as its name suggests, to obtain the first N characters of a string starting from the left. A LEFT$ instruction takes the form:
LEFT$ (A$,N)
which would produce a string consisting of the first N characters of A$. Its action is illustrated by running the following short program.

```
10 REM *** LEFT$(A$,N) demo ***
20 a$="123456789"
30 b$="abcdefghijklmnopqrstuvwxyz"
40 PRINT LEFT$(a$,4)
45 PRINT
50 PRINT LEFT$(b$,13)
```

The display obtained is

```
1234 <————————— first 4 characters of a$ starting from left

abcdefghijklm <— first 13 characters of b$
```

The RIGHT$ function

The RIGHT$ function is used to obtain the last *N* characters of a string, ending with the rightmost character. For example, the display obtained when running the following program:

```
10 REM *** RIGHT$(A$,N) demo ***
20 PRINT RIGHT$("Meadows R G",3)
30 PRINT
40 a$="ABCDEFGHIJ"
50 PRINT RIGHT$(a$,6)
```

is

```
R G      <————————————— last 3 characters of "Meadows RG"

EFGHIJ   <————————————— last 6 characters of a$
```

The MID$ function

The MID$ function allows any portion of a string to be obtained. It takes the general form,
MID$ (A$, M, N)
where the whole numbers (integers) *M* and *N* define that the string so formed consists of *N* characters starting from the *M*th character from the left. For example,

```
10 REM *** MID$(A$,M,N) demo ***
20 PRINT MID$("1234abc89",5,3)
```

would produce the output
abc
i.e. a 3 character string starting from the 5th character from the left.

5.9 THE LEN FUNCTION FOR DETERMINING THE 'LENgth' OF A STRING

The LEN standard function gives the length, i.e. the number of characters, in a string. For example
(1)
PRINT LEN "How many characters in this string?" [ENTER]
35 . . . result displayed.
Remember all characters including spaces, punctuation marks, etc. in the string count as part of its 'length'.

(2)

```
10 REM *** LEN(a$) demo ***
20 a$="R G Meadows"
30 PRINT LEN(a$)
40 PRINT
50 PRINT LEN("123456789abcdef")
```

RUN [ENTER]
11 . . . result for LEN(a$)
15 . . . result for LEN("123456789abcdef")

5.10 THE VAL AND STR$ FUNCTIONS

The VALue function converts a 'number' string into the actual number which can then be assigned, for example, to a numeric variable.

The STR$ function is the inverse; it converts numeric data into a string.

The following short program demonstrates their action.

```
10 REM *** VAL and STR$ demo ***
20 a$="4453"
30 n=VAL(a$)
40 PRINT a$,n,n*n
50 PRINT
60 b$=STR$(n)+STR$(n)
70 PRINT b$
```

RUN [ENTER]

```
4453          4453          19829209

 4453 4453
```

The first line of output (line 40) displays
a$ (i.e. "4453") n (i.e. 4453) and n*n
Clearly the VAL function has converted the string "4453" into the number 4453.

The second line of output represents the string b$ formed by combining the strings "4453" and "4453".

5.11 CHARACTER SETS AND CODES: CHR$ AND ASC STANDARD FUNCTIONS

The characters, i.e. letters; digits 0, 1, 2, . . . ; punctuation marks; and arithmetic symbols that appear on the CPC464 keyboard form part of what is known as the character set of the CPC464 computer. The total set also includes many graphics type characters.

Each character is assigned a number code. The function CHR$ ( ) converts this code number to its character equivalent in the character set of the Amstrad CPC464 computer.

The following program displays a portion of the CPC464 character set from $n = 33$ to $n = 126$. This range covers most of the keyboard characters.

```
10 PRINT "Code no n","Character,CHR$(n)"
20 PRINT
30 FOR n=33 TO 126
40 PRINT n,CHR$(n)
50 NEXT n
```

You will obtain a line by line display if you run the program. The corresponding table is shown in fig 5.6

The remainder of the set extends from $n = 127$ to $n = 255$ and consists of graphics-type symbols. Modify line 30
30 FOR n = 127 to 255
to see the actual characters displayed. Remember you can press the ESC key once to pause the program execution (and the display) and press any other key for continue.

One of the most commonly used codes for transmitting information between computers and associated devices such as printers etc. is the ASCII code. The decimal equivalent of the ASCII code of a character can be found using the ASC("X") function.

The ASC function returns the code for the character *X* within the quotation marks or if a string is present the first character of the string. Obviously the character must be a member of the ASCII set. For alphabetic characters, digits, punctuation marks, etc. the CPC464 code is identical to the ASCII codes.

Code no. *n*	CHR$ (n) Character	Code no. *n*	CHR$ (n) Character	Code no. *n*	CHR$ (n) Character
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	↑	126	~
63	?	95	_		
64	@	96	`		

Fig. 5.6 Part of character set for CPC464 computer.

For example,

(1)

PRINT ASC("A") [ENTER]

65 . . . ASCII and CPC464 code for A

(2)

PRINT ASC("a") [ENTER]

97 . . . ASCII and CPC464 code for a

(3)

PRINT ASC("+") [ENTER]

43 . . . code for + symbol

(4)

PRINT ASC("676")

54 . . . code for 6, the first character of the string

(5)

PRINT ASC ("String here") [ENTER]

83 . . . code for S

Exercise problems 5

(1) Use the functions ROUND, INT and FIX appropriately to
(a) round the following to the nearest whole number
4.69, 587.3, −199.9, 0.678, −0.989
(b) specify the following to an accuracy of 2 decimal places
58.632998, −9.9989, 0.46798, $\sqrt{58.67}$

(2) Write a program to display a list of 16 random numbers in the range 1 to 55. Use it to generate a treble-chance forecast!

(3) Write functions that will return the following values
(a) the average of 3 quantities
(b) the volume of a cylinder, given
$V = \pi r^2 h$
where r = radius and h = height of cylinder

(4) Use the LEN function to determine the lengths of the strings
(a) "42 Lynton Avenue, London, NW55"
(b) A$ = "A stitch in time, saves 9"
(c) A$ + B$, where A$ = "123456789" and B$ + "abcdefghijklmnopq"

(5) (a) Find the ASCII (and CPC464 codes) for
Z , ; 9 − = * !
(b) Find the characters corresponding to the following CPC464 codes:
118, 35, 59, 86, 249, 251

6

BASIC GRAPHICS, DRAWING AND PLOTTING

6.1 INTRODUCTION AND SUMMARY

The CPC464 computer incorporates excellent and easy-to-use graphics facilities. In this chapter we consider their use.

We start by explaining the graphics coordinate system and the use of the graphics commands: PLOT, DRAW and ORIGIN. With these commands, as their name suggests, we can plot points, draw lines, set the origin and compose our own computer-aided drawings.

We also consider the use of both screen and graphic cursor commands to LOCATE and MOVE the respective cursors to any position on the screen and their use to label our figures. Applications of graphics to drawing basic shapes, defining and using WINDOWS, and plotting curves, waves and our own graphs are also included.

6.2 THE GRAPHICS SYSTEM AND BASIC KEYWORDS PLOT, DRAW, ORIGIN

1 The graphics coordinate system

The graphics coordinate system is shown in fig. 6.1. The origin, the (0,0) point, is at the bottom left hand corner of the screen 'page'. The horizontal or *x*-axis runs from 0 to 640 units. The vertical or *y*-axis runs from 0 to 400 units.

The units used in the graphics system are called pixels. A pixel is essentially the smallest possible element making up a 'picture'; it is a dot used to form points, lines, etc. We used the pixel units to define the position of points in the graphics system.

2 PLOT and PLOTR for plotting points

PLOT and PLOTR are used to plot points, i.e. to ink in points with respect to the graphics coordinate system so they are visible on the screen.

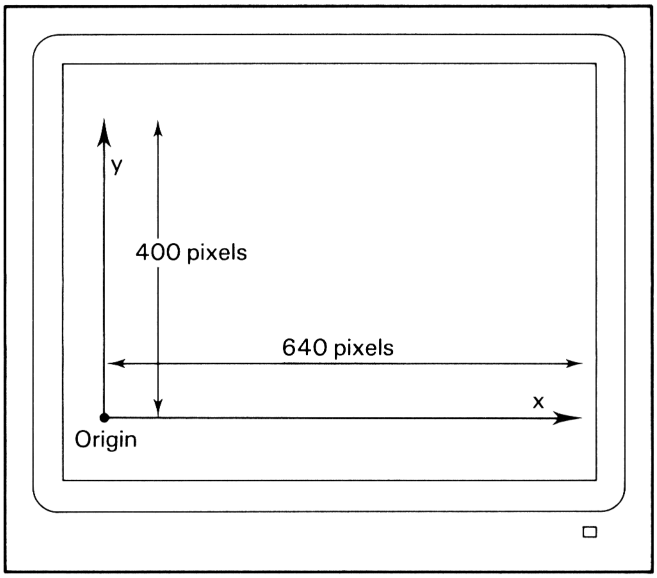


Fig. 6.1 The graphics coordinate system.

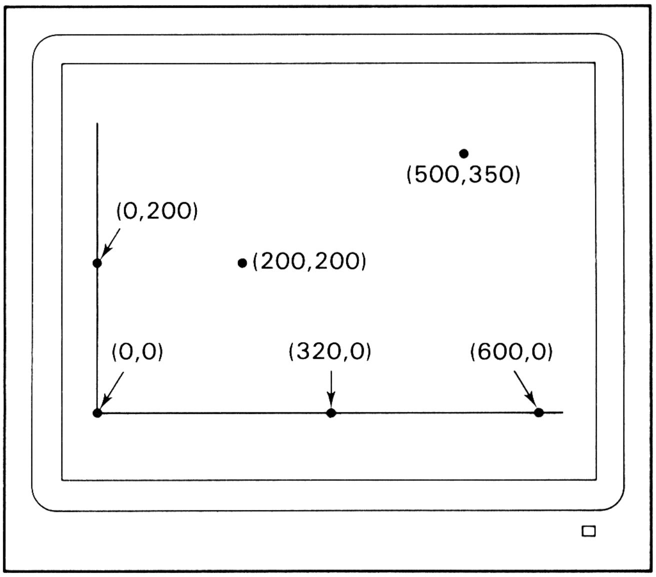


Fig. 6.2 The PLOT x, y command for plotting points with respect to graphics origin.

The command,
PLOT x,y [ENTER]
plots a point with respect to the graphics origin, i.e. *x* pixel units along the *x*-axis and *y* pixel units along the *y*-axis. For example (see fig. 6.2);
PLOT 0,0 . . . inks in the point (0,0) at the origin
PLOT 320,0 . . . plots a point half way along the *x*-axis
PLOT 200,200 . . . plots a point located 200 units from the origin along the *x*-axis and then 200 units vertically upwards.

The command,
PLOTR xr,yr [ENTER]
plots a point relative to the current graphics cursor position. For example, suppose we have set the graphics cursor at the absolute point (200,100) by the command
PLOT 200,100 [ENTER] . . . point A in fig. 6.3
then to plot a second point a further 300 units away in the *x* direction and 190 units in the *y* we can use,
PLOTR 300,190 [ENTER] . . . point B in fig. 6.3

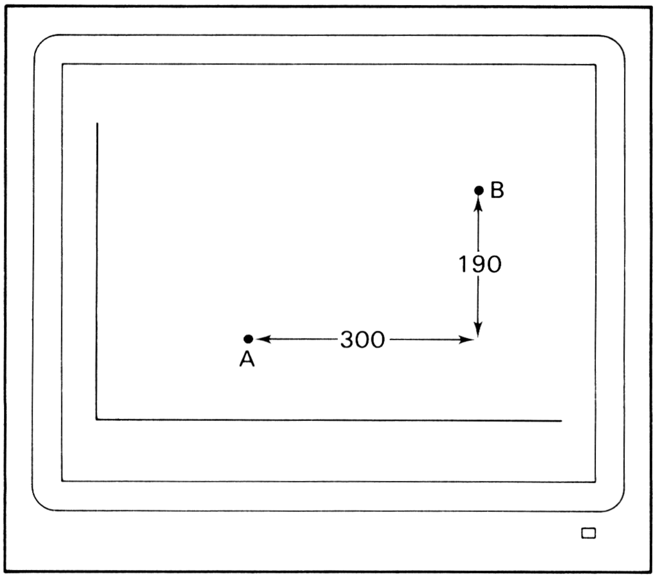


Fig. 6.3 The PLOTR xr, yr command for plotting points relative to the last cursor position.

3 DRAW and DRAWR for drawing lines

The command,
DRAW x,y [ENTER]
draws a straight line on the screen from the current graphics cursor position to the absolute position specified by *x,y*. For example, suppose we set the graphics cursor at the origin by the command,

PLOT 0,0 [ENTER]
then the command,
DRAW 250,300 [ENTER]
draws the line from (0,0) to (250,300), i.e. line OA in fig. 6.4

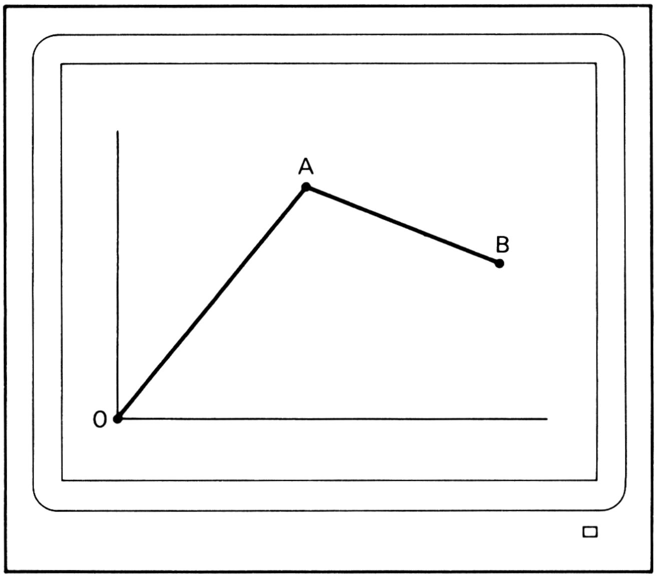


Fig. 6.4 The DRAW x, y and DRAW xr, yr commands for drawing straight lines.

The 'relative' DRAW command,
DRAWR xr,yr [ENTER]
draws a line from the current graphics cursor position to a point a further *xr* units away in the *x*-axis direction and *yr* units in the *y*-axis direction, e.g.
DRAWR 250,−100 [ENTER]
draws the line AB in fig. 6.4.

The following two programs illustrate the application of the PLOT, DRAW and DRAWR graphics keywords.

This program 'draws' the triangle shown in fig. 6.5.

```
10 REM *** Drawing a triangle ***
20 CLS
30 PLOT 0,0
40 DRAW 500,0
50 DRAWR 0,400
60 DRAW 0,0
```

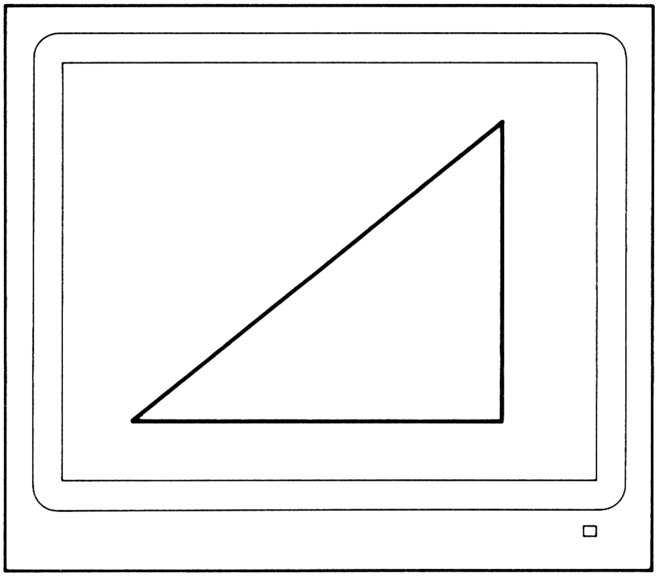


Fig. 6.5 Drawing a triangle.

This program allows you to draw any size of rectangle, with the lower left-hand corner at the origin (0,0) and with dimensions input from the keyboard.

```
10 REM *** Drawing a rectangle ***
20 REM **  Dimensions input from keyboard **
30 CLS
40 PLOT 0,0
50 INPUT "Enter length (not exceeding 630)";l
60 INPUT "and height (not exceeding 390)";h
70 CLS
80 DRAW l,0
90 DRAWR 0,h
100 DRAWR -l,0
110 DRAW 0,0
120 GOTO 120
```

4 ORIGIN

The graphics coordinate system origin can be changed by the command,
ORIGIN, X,Y, [ENTER]
which will set the new origin at the point *X* pixel units horizontally and *Y* pixel units vertically from the original (0,0) point at the bottom left-hand corner of the screen page.

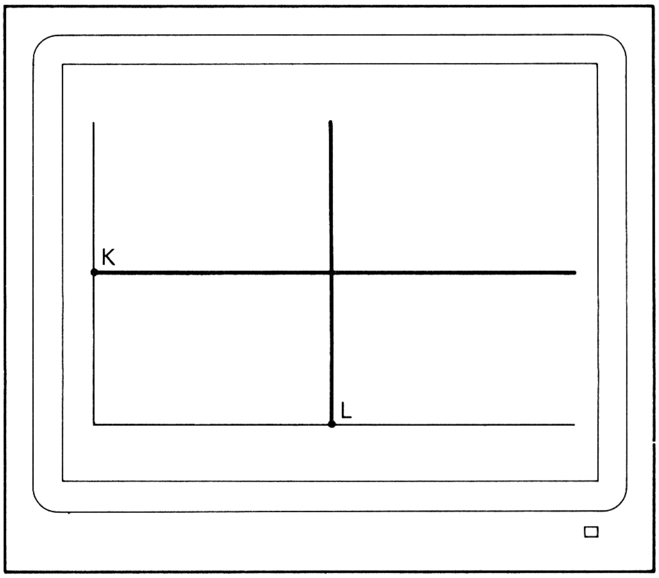


Fig. 6.6 Changing the position of the origin using the command ORIGIN X, Y.

For example, the command
ORIGIN 320,200 [ENTER]
will set a new origin at the centre of the screen. The PLOT and DRAW commands will then plot points and draw lines with respect to this new origin, e.g. see fig. 6.6.

```
PLOT -320,0 [ENTER]      . . . plots point K
DRAW 320,0 [ENTER]       . . . draws the new x-axis
PLOT 0,-200 [ENTER]      . . . plots point L
DRAW 0,200 [ENTER]       . . . draws the new y-axis
```

6.3 DRAWING AND LABELLING SOME BASIC FIGURES

In this section we use the basic graphics keywords to draw some common geometrical figures. We combine these drawings with showing how we may attach character labels using the following screen and graphic cursor commands:

(1) LOCATE (column number, line number) . . . used to locate the screen character cursor to a given column-line position on the screen

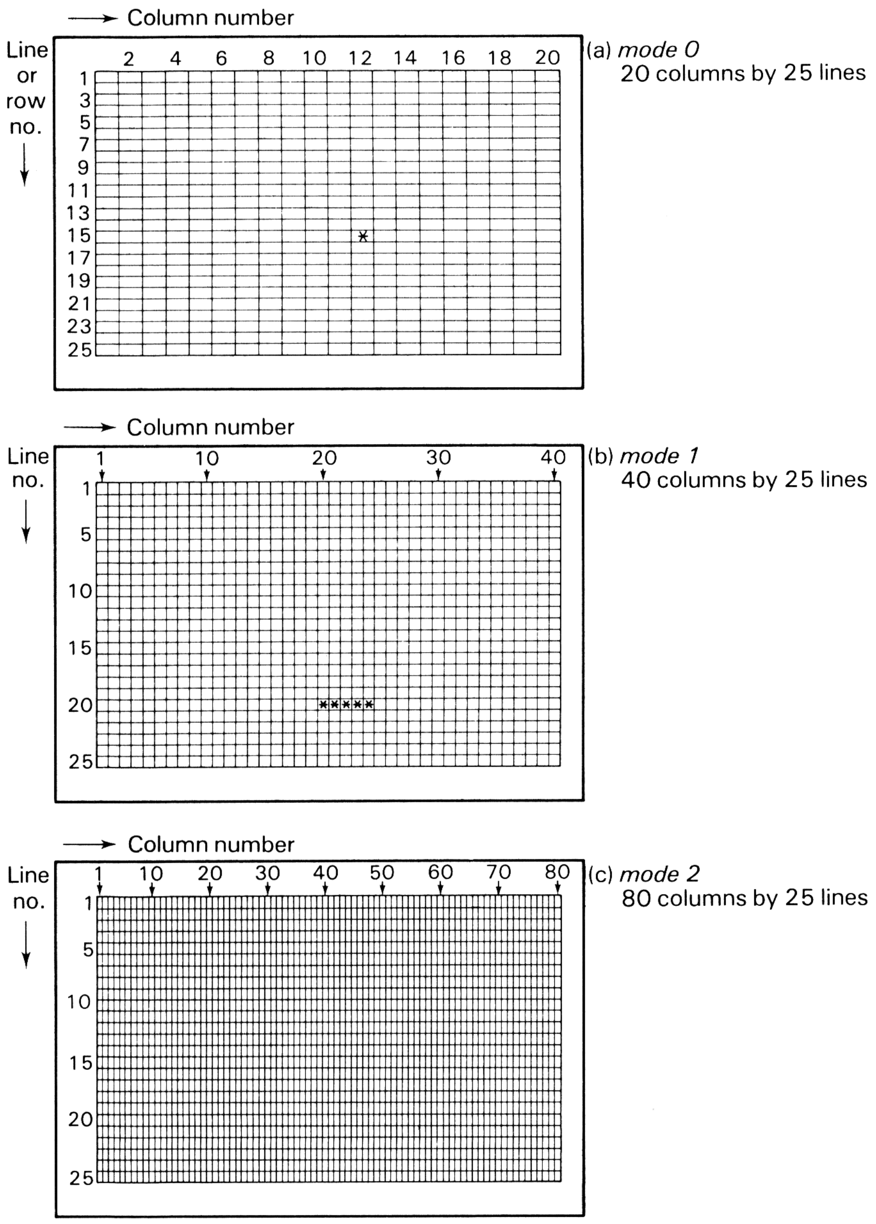


Fig. 6.7 Column-row structure for screen modes. Used for screen cursor LOCATION and also for defining WINDOW dimensions and position (see section 6.5).

In mode 0 the screen is considered as divided into 20 columns and 25 lines or rows (see fig 6.7(a)). Thus the command,

```
LOCATE 12,15 [ENTER]
```

places the screen cursor 12 columns across the screen and 15 lines down the screen, so if we followed this command by,

```
PRINT "*" [ENTER]
```

the star symbol (*) will be displayed at this position.

In mode 1, the normal mode, the screen is divided into 40 columns and 25 lines, so

```
LOCATE 20,20 [ENTER]
PRINT "*****" [ENTER]
```

will display ***** starting at column 20, line 20, as shown in fig. 6.7(b).

Mode 2, the high resolution mode, is considered to be divided into 80 columns by 25 lines (see fig. 6.7(c)).

(2) MOVE and MOVER

Just as LOCATE is used to position the screen character cursor, the keywords MOVE and MOVER are used to position the graphics cursor.

MOVE x,y positions graphics cursor to the point *x* pixels units horizontally and *y* pixel units vertically with respect to the graphics origin.

MOVER xr,yr positions the graphics cursor *xr* and *yr* pixel units from the current graphics cursor position

MOVE and MOVER are essentially identical to PLOT and PLOTR but without actually plotting the point.

(3) TAG is used in conjunction with MOVE and MOVER to attach or 'tag' a character string to be displayed starting at a specific graphics coordinate location. For example,

```
MOVE 320,200 [ENTER]
```

will position graphics cursor at the centre of the graphics system.

```
TAG [ENTER]
```

will now direct any subsequent display to start at this position, so

```
PRINT "centre-point"; [ENTER]
```

would cause *centre-point* to be displayed starting at the graphics position 320,200. Note the PRINT statement should be terminated by a semi-colon (;). If this is omitted the control character $\rightarrow \downarrow$ will be displayed following the string.

TAG is cancelled by TAGOFF.

Drawing some common figures: program examples

(1) The following program 'draws' the four basic types of triangles. LOCATE is also used to label the drawings.

```
5 REM *** Drawing Triangles ***
10 CLS
20 PLOT 0,200
30 DRAW 640,200
40 PLOT 320,0
50 DRAW 320,400
60 LOCATE 5,12
70 PRINT "Right-angled"
80 PLOT 10,240
90 DRAWR 250,0
100 DRAWR 0,150
110 DRAW 10,240
120 LOCATE 25,12
130 PRINT "Scalene"
140 PLOT 340,240
150 DRAWR 200,0
160 DRAWR -140,150
170 DRAW 340,240
180 LOCATE 5,23
190 PRINT "Obtuse"
200 PLOT 30,60
210 DRAW 140,60
220 DRAWR 150,100
230 DRAW 30,60
240 LOCATE 25,23
250 PRINT "Isosceles"
260 PLOT 350,60
270 DRAWR 200,0
280 DRAWR -100,125
290 DRAW 350,60
```

On running the program you will obtain the display shown in fig. 6.8
(2) This program displays regular polygons (many-sided figures). On running the program you are asked to enter the number of sides. Try running the program. With $n=3$ you will obtain a triangle; $n=4$, a square; $n=5$, a pentagon; $n=6$, a hexagon (see fig. 6.8), and so on. If you choose a high value of n, say over 30, the figure will closely resemble a circle.

```
10 REM *** drawing polygons ***
20 CLS
30 INPUT "Enter number of sides";n
40 ORIGIN 320,200
50 PLOT 150,0
60 FOR s=1 TO n
70 x=150*COS(2*PI*s/n)
```

```
80 y=150*SIN(2*PI*s/n)
90 DRAW  x,y
100 NEXT s
110 LOCATE 17,12
120 PRINT "n =";n
```

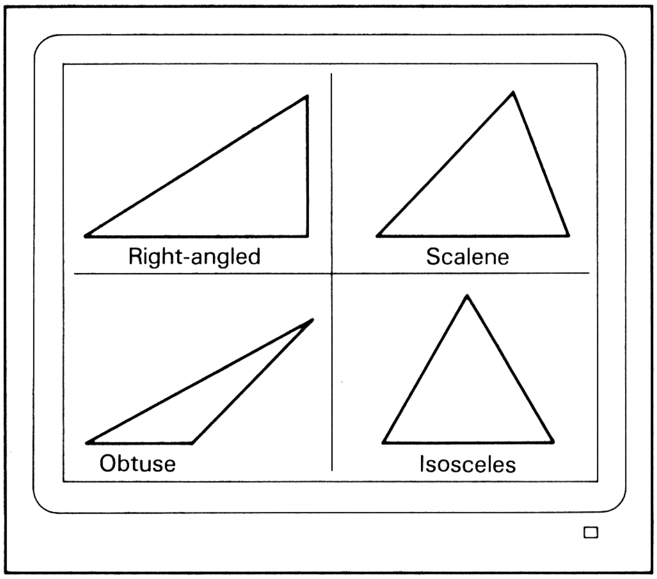


Fig. 6.8 Display produced by triangle program.

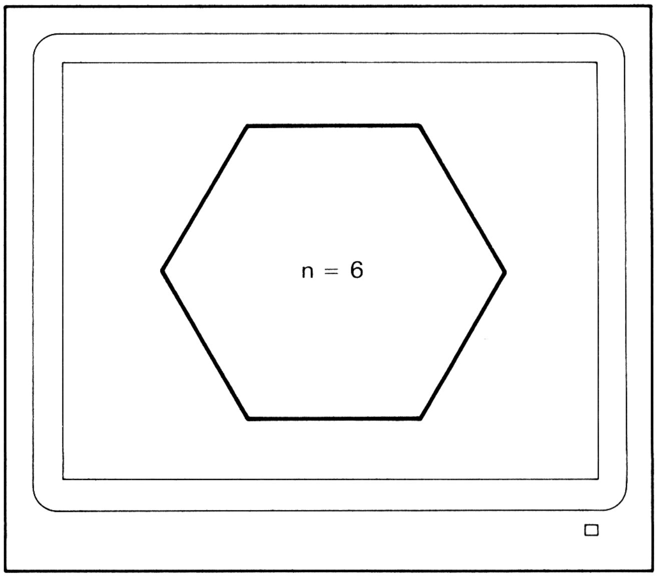


Fig. 6.9 Display produced by polygon program for $n = 6$.

(3) This program demonstrates the use of MOVE. It begins by drawing a square of side 20 with the left-hand corner at the origin. You can then 'MOVE' the square to be drawn at any other position within the graphics coordinate system.

```
10 REM *** MOVE demo ***
20 CLS
30 MOVE 0,0
40 DRAWR 0,20 :DRAWR 20,0
50 DRAWR 0,-20:DRAWR -20,0
60 INPUT "Enter distance x to be moved";x
70 INPUT "Enter distance y to be moved";y
80 MOVE x,y
90 GOTO 40
```

(4) This program can be used for drawing ellipses or circles.

```
10 REM *** Drawing an ellipse ***
20 CLS:ORIGIN 320,200
30 INPUT"length of semi-major axis";a
40 INPUT"eccentricity";e
50 b=e*a
60 FOR x=-a TO a
70 y=b/a*SQR(a^2-x^2)
80 PLOT x,y
90 PLOT x,-y
100 NEXT x
```

Enter in the length of the semi-major axis and the eccentricity (which is 1 for circles) and then the program will plot the ellipse. A typical display is shown in fig. 6.10.

(5) The following two programs attempt to draw in perspective. The first one draws a box (see fig. 6.11(a)), and the second a cylinder (see fig. 6.11(b)).

```
10 REM *** Drawing a  box or cuboid ***
15 CLS
20 PLOT 50,50
30 DRAWR 100,0:DRAWR 0,100
40 DRAWR -100,0: DRAWR 0,-100
50 MOVE 150,50
60 DRAWR 71,71:DRAWR 0,100:DRAW 150,150
70 MOVE 50,150
80 DRAWR 71,71: DRAWR 100,0

10 REM *** Drawing a cylinder ***
20 CLS:ORIGIN 320,300
30 INPUT "Radius of cylinder";a
40 INPUT "and height";h
50 CLS
```

```
60 FOR x=-a TO a
70 y= 0.5*SQR(a^2-x^2)
80 PLOT x,y
90 PLOT x,-y
100 NEXT x
110 ORIGIN 320,300-h
120 FOR x=-a TO a
130 y= 0.5*SQR(a^2-x^2)
140 PLOT x,y
150 PLOT x,-y
160 NEXT x
170 MOVE -a,0:DRAW -a,h
180 MOVE  a,0:DRAW a,h
```

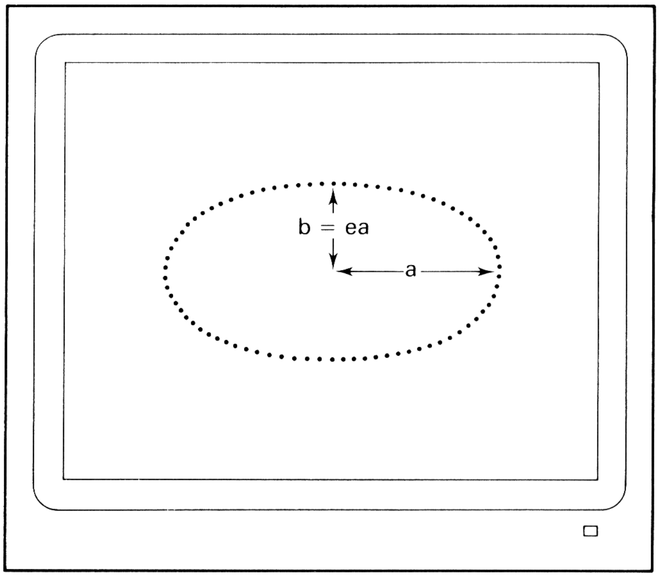


Fig. 6.10 Display produced by ellipse program.

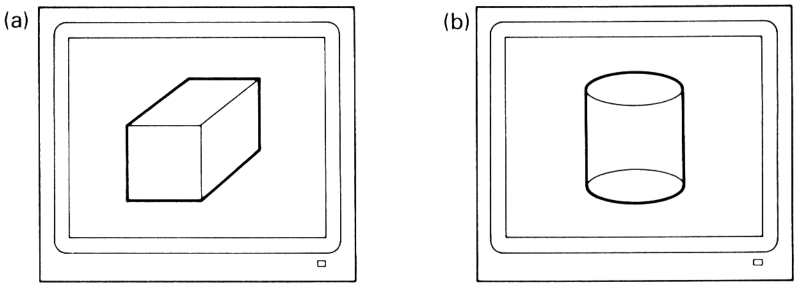


Fig. 6.11 Displays produced by (a) box and (b) cylinder programs.

6.4 APPLICATIONS TO CURVE AND WAVE PLOTTING

We can also use numerical or mathematical expressions rather than just pure numbers in PLOT statements, in much the same way as used in the last section to trace out an ellipse. This allows a direct and very easy way of plotting curves of mathematical equations, functions and waves.

The following program examples illustrate how this is done.

1 Sketch of a parabola

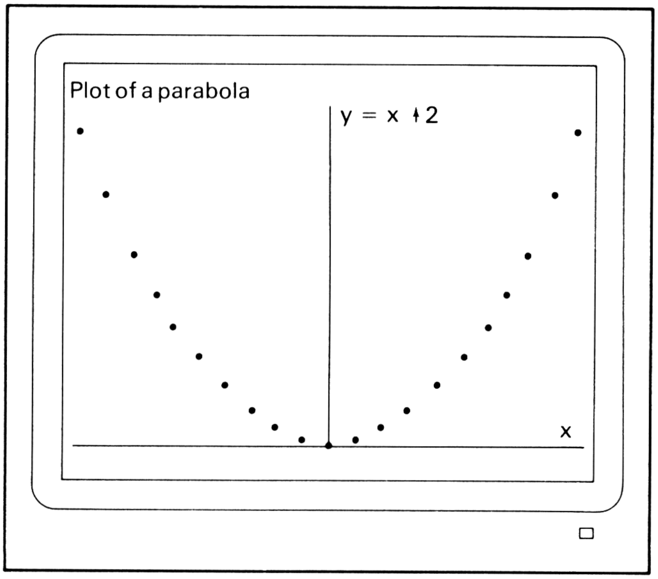


Fig. 6.12 Display obtained on running parabola program.

The program listed below plots the curve of the parabola $y=x^2$ over the range $-20 \leqslant x \leqslant 20$. Note that the origin has been moved to the centre of the horizontal axis (see line 40) and a scale magnification factor of 16 has been used for x values (see line 90). The x and y axes are drawn in using DRAW statements and the actual plotting is accomplished by the FOR loop (lines 100 to 130).

The display obtained on running the program is shown in fig. 6.12.

```
10 REM *** Parabola plot ***
20 CLS
30 PRINT "Plot of a parabola"
40 ORIGIN 320,0
50 PLOT -320,0
```

```
60 DRAW 320,0
70 PLOT 0,0
80 DRAW 0,400
90 xscale=16
100 FOR x=-20 TO 20
110 y=x*x
120 PLOT x*xscale,y
130 NEXT x
140 MOVE 10,360:TAG:PRINT "y=x^2";
150 MOVE 300,20:PRINT "x";
160 GOTO 160
```

Note the 'Ready' caption has been removed from the display by the inclusion of line statement 160. To 'break' program execution and return the cursor to the screen, press the ESC key twice.

2 Drawing spirals

The following program plots two spirals, the first 'travelling inwards' and the second 'growing outwards'. The tightness of the spirals is governed by the value given to *n* in line 50. The number of spiral turns is governed by the end value of *s* in line 140. On running the program the first spiral is displayed (see fig. 6.13). This is followed by a short delay and then the plot of the second spiral is displayed.

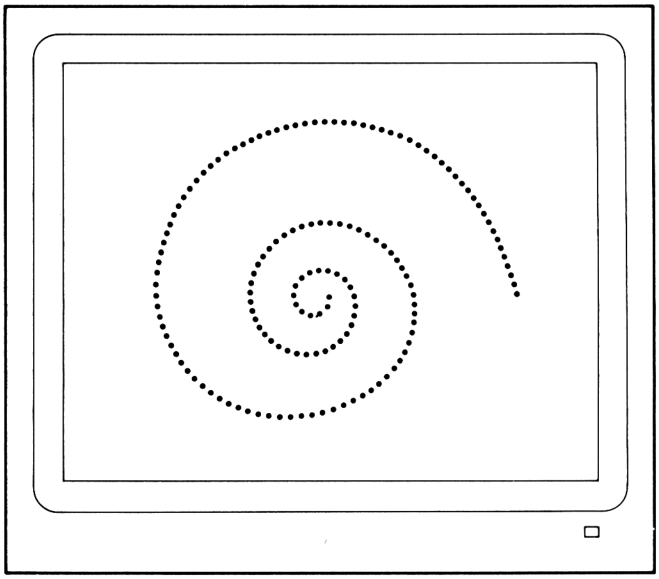


Fig. 6.13 Display from first part of 'Drawing spirals' program.

```
10 REM *** Drawing spirals ***
20 CLS
30 ORIGIN 320,200
40 PRINT "Spiral travelling inwards"
50 n=100
60 FOR s=0 TO 400
70 x= 200*EXP(-s/n)*COS(2*PI*s/n)
80 y= 200*EXP(-s/n)*SIN(2*PI*s/n)
90 PLOT x,y
100 NEXT s
110 FOR delay=1 TO 5000:NEXT delay
120 CLS
130 PRINT "Spiral growing outwards"
140 FOR s=0 TO 400
150 x= 5*EXP(s/n)*COS(2*PI*s/n)
160 y= 5*EXP(s/n)*SIN(2*PI*s/n)
170 PLOT x,y
180 NEXT s
```

3 Plotting sinewaves: Fourier analysis illustration

Although an advanced topic, readers might be interested in trying this program which demonstrates the power of the Amstrad graphics.

The program illustrates graphically an example of an application of a famous theorem known as Fourier's theorem. The latter effectively states that any periodic waveform can be considered to be made up a series of sinewaves whose frequencies are multiples (harmonics) of the waveform frequency. For example, the unit amplitude square wave shown in fig. 6.14(a) can be expressed as a series of cosine waves, i.e.

square wave, $y = \frac{4}{\pi}\,[\cos x - \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x - \frac{1}{7}\cos 7x + \ldots]$

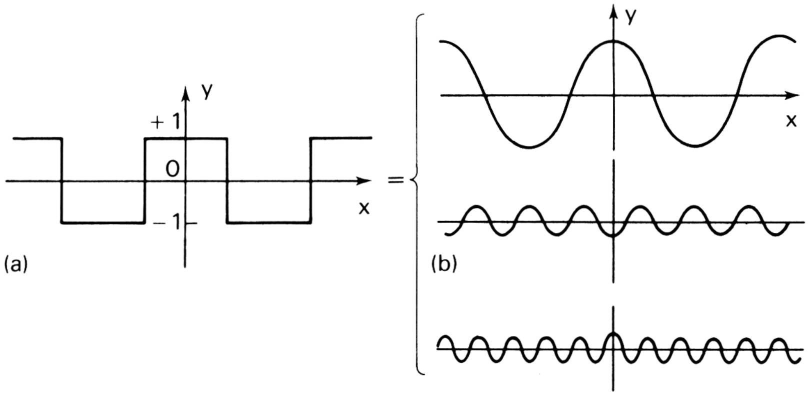


Fig. 6.14 Fourier analysis of a square wave: wave can be considered as made up of a number of cosine waves. (a) Square wave. (b) Fundamental, third and fifth harmonic components.

The first three component waves of the series are:

$\frac{4}{\pi} \cos x$	. . . the fundamental
$\frac{4}{3\pi} \cos 3x$	. . . the third harmonic
$\frac{4}{5\pi} \cos 5x$	. . . the fifth harmonic

```
10 REM *** Fourier Illustration ***
20 CLS
30 ORIGIN 0,300
40 PLOT 0,0 :DRAW 600,0
50 c=4/PI
60 DEG
70 FOR x=1 TO 600
80 y=30*c*COS(x)
90 PLOT x,y
100 NEXT x
110 ORIGIN 0,200
120 PLOT 0,0:DRAW 600,0
130 FOR x=1 TO 600
140 y=30*c*(COS(x)-1/3*COS(3*x))
150 PLOT x,y
160 NEXT x
170 ORIGIN 0,100
180 PLOT 0,0:DRAW 600,0
190 FOR x=1 TO 600
200 y=30*c*(COS(x)-1/3*COS(3*x)+1/5*COS(5*x))
210 PLOT x,y
220 NEXT x
```

The program plots out three waveforms:
the fundamental (see fig. 6.15(a))
the sum of the fundamental and the third harmonic (see fig 6.15(b))
the first 3 terms of the Fourier series, i.e. sum of the fundamental and third and fifth harmonics (see fig 6.15(c)).
Try running the program and then modifying it to include further terms, e.g. the 7th, 9th, etc. harmonics, and investigate how accurately a square wave can be represented by means of a given number of Fourier components.

6.5 USING WINDOW

The CPC464 computer has the very useful facility of defining areas of the screen—i.e. WINDOWs—which we can use to send data for display in a given area.

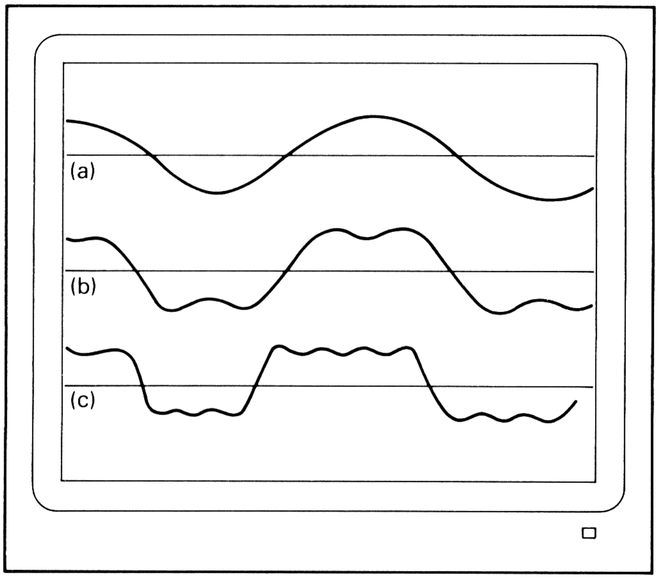


Fig. 6.15 Display produced by 'Fourier Illustration' program.

A window can be created using the statement,
100 WINDOW #n, left col.no., right col.no., top line no., bottom line no.
where #n defines the internal channel which we select to send our PRINT output data to the WINDOW; *n* can take the values 1, 2, 3, 4, 5, 6 or 7

left col. no.	. . . column number defining left-hand side of window
right col. no.	. . . column number defining right-hand side of window
top line no.	. . . line number defining top of window
bottom line no.	. . . line number defining bottom of window

For planning your window size and position use the column-line grids for modes 0, 1 and 2 given in fig. 6.7.

The following two programs illustrate how windows are defined and how data may be directed to a given screen window.

The first program creates 4 different size windows when working in mode 1, as shown in fig. 6.16. If you are working with colour you will note that each window has its own page and pen colour. The PRINT #n, statements are also used to direct output display to the desired windows.

```
10 REM *** Window demo ***
15 REM ** Creation of 4 windows **
20 CLS
30 BORDER 0
40 WINDOW #1,1,10,14,25
```

```
50 PAPER #1,3:CLS #1
60 PRINT #1, "window 1"
70 WINDOW #2,11,40,13,25
80 PAPER #2,1:CLS #2
90 PEN #2,3
100 PRINT #2:PRINT #2,"window 2 here"
110 PRINT #2 :PRINT #2
120 FOR n=1 TO 20
130 PRINT #2,CHR$(250);
140 NEXT n
150 WINDOW #3,1,20,1,13
160 PRINT #3,"Window 3 here"
170 PAPER #3,2:CLS #3
180 PEN #3,0
190 PRINT #3,"Window 3 here"
200 WINDOW #4,21,40,1,13
210 PRINT #4,"Window 4 here"
220 GOTO 220
```

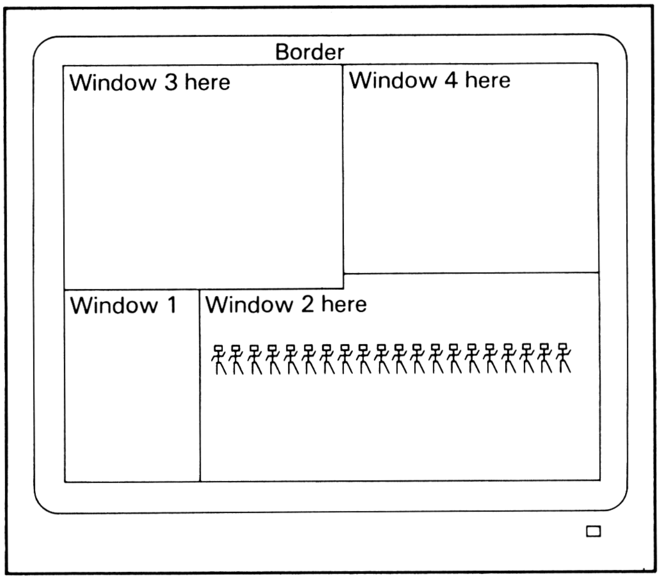
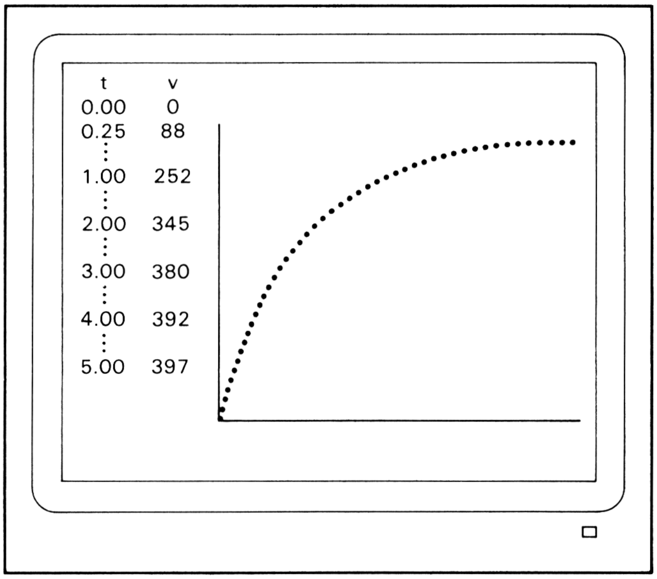


Fig. 6.16 Display produced by 'window demo' program.

The second program creates two WINDOWs—one for outputting a table of results and the other alongside for plotting a graph of the results. Note that the high resolution mode, mode 2, is used for the results table display.

```
10 REM *** window application demo ***
20 REM * results table + graph display *
30 CLS
40 MODE 2
50 WINDOW #1,1,20,1,25
60 PRINT #1,".t      ";" v "
70 PRINT #1
80 FOR t=0 TO 5 STEP 0.25
90 PRINT #1,USING "#.##";t;
100 PRINT #1,TAB(8);INT(400*(1-EXP(-t)))
110 NEXT t
120 WINDOW #2,21,80,1,25
130 ORIGIN 200,0
140 PLOT 0,0:DRAW 400,0
150 PLOT 0,0:DRAW 0,400
160 FOR t=0 TO 5 STEP 0.25
170 PLOT 80*t,400*(1-EXP(-t))
180 NEXT t
190 GOTO 190
```

The display obtained on running this program is shown in fig. 6.17.



```
 t       v
0.00     0
0.25    88
  :
1.00   252
  :
2.00   345
  :
3.00   380
  :
4.00   392
  :
5.00   397
```

Fig. 6.17 Display of table of results and graph plot produced by the second 'window demo' program.

6.6 PLOTTING YOUR OWN GRAPHS

In this final section three examples of programs that can be used to plot your own graphs are presented.

1 Program to plot x,y points with data in DATA statements

In this program the data for each *x,y* point is included in pairs in the DATA statements at lines 150 to 180. A FOR loop is used to both READ the *x,y* values and PLOT the respective points. Both *x* and *y* axes are also drawn and labelled. The range of points is limited in this case to the graphics coordinate system, i.e. *x* from 0 to 640 and *y* from 0 to 400.

```
10 REM *** To plot a graph ***
20 CLS
30  REM *** READing and PLOTting the points ***
40 FOR n=1 TO 14
50 READ x,y
60 PLOT x,y
70 NEXT n
80 REM ** drawing x axis **
90 MOVE 0,0:DRAW 630,0
100 MOVE 600,20:TAG:PRINT "x";
110 REM ** drawing y axis **
120 MOVE 0,0:DRAW 0,400
130 MOVE 20,370:PRINT "y";
140 REM *** x,y data for the point ***
150 DATA 10,30,50,100,102,168,160,225
160 DATA 210,270,275,305,356,322,401,300
170 DATA 445,271,470,226,515,174,552,103
180 DATA 572,60,595,15
```

The graph plot obtained on running this program is shown in fig. 6.18.

2 Program to plot x,y points with data INPUT from keyboard

This program is similar to the first but allows points to be INPUT directly from the keyboard. After completing your entries, input −999. This will break the input loop and immediately commence plotting. [Note arrays considered in the next chapter are used to store the *x,y* data.]

```
10 REM *** To plot a graph ***
20 REM *** x,y values INPUT from keyboard ***
30 CLS
```

```
40 DIM x(50):DIM y(50)
50 n=1
60 INPUT "x value of point";x(n)
70 IF x(n)=-999 THEN 120
80 INPUT "y value of point";y(n)
90 PRINT
100 n=n+1
110 GOTO 60
120 CLS
130 REM ** drawing x axis **
140 PLOT 0,0:DRAW 630,0
150 MOVE 600,20:TAG:PRINT "x";
160 REM ** drawing y axis **
170 PLOT 0,0:DRAW 0,400
180 MOVE 20,370:PRINT "y";
190 REM *** plotting the point ***
200 FOR t=1 TO n-1
210 PLOT x(t),y(t)
220 NEXT t
230 GOTO 230
```

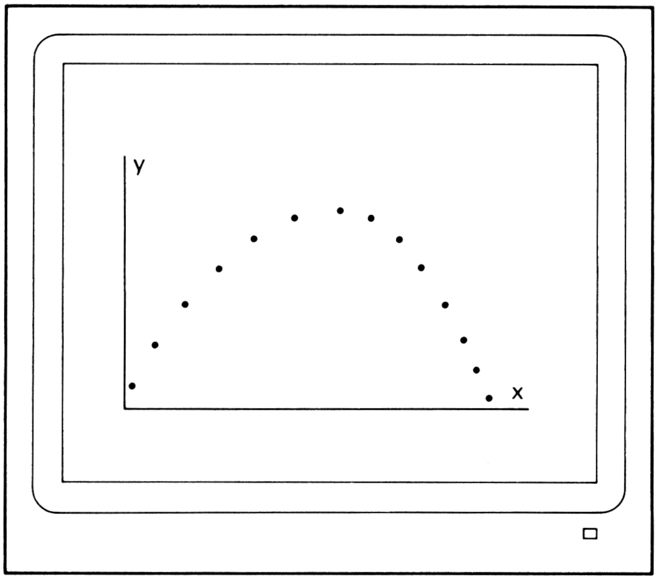


Fig. 6.18 Output display for first 'To plot a graph' program.

3 x–y graph plotting with automatic scaling

This program has an automatic scaling facility so we are not restricted to the range 0–640 for *x* and 0–400 for *y*. As the x–y data is being entered the values are stored and IF statements are used to find the maximum and minimum values. This information is then used to set the scale factors for *x* and *y* and the position of the origin.

```
10 REM *** Graph plotting ***
20 REM ** Automatic scaling **
30 xmin=1E+10:ymin=1E+10
40 xmax=-1E+10:ymax=-1E+10
50 DIM x(50): DIM y(50)
60 CLS
70 n=1
80 INPUT "x value of point";x(n)
90 IF x(n)=-999 THEN 180
100 INPUT "y value of point";y(n)
110 PRINT
120 IF x(n)>xmax THEN xmax=x(n)
130 IF y(n)>ymax THEN ymax=y(n)
140 IF x(n)<xmin THEN xmin=x(n)
150 IF y(n)<ymin THEN ymin=y(n)
160 n=n+1
170 GOTO 80
180 xf=640/(xmax-xmin)
190 yf=400/(ymax-ymin)
200 ORIGIN -xmin*xf,-ymin*yf
210 CLS
220 FOR t=1 TO n-1
230 PLOT x(t)*xf,y(t)*yf
240 NEXT t
```

Exercise problems 6

(1) Display the following on the screen:
(a) a rectangle with corners at (50,50) , (50,300) , (500,300) and (500,50)
(b) a right-angled triangle with a base of 400 units and a height of 250.

(2) Write programs to
(a) draw a circle centre (320,200) and radius 150
(b) draw an ellipse centre (200,200), semi-major axis length 100 and eccentricity 0.7.

(3) Write a program to display the ‘house’ shown in fig. 6.19.

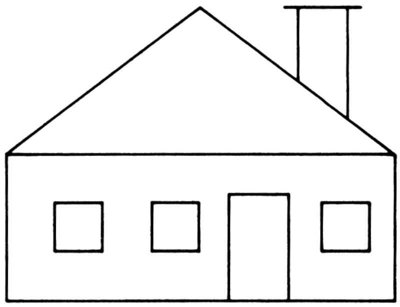


Fig. 6.19 Drawing for exercise problem 3.

(4) Write programs to plot the following functions:
(a) $y = \cos x$ from $x = -320°$ to $+ 320°$
(b) $y = 400e^{-x}$ from $x = 0$ to 5.

(5) Write a program that will plot the following points:
(0,30) , (30,75) , (60,120) , (90,156) , (120,198) , (150,224) , (180,237) , (210,228) , (240,201) , (270,150) , (300,126) , (330,70) , (360,25).

7

APPLICATIONS OF ARRAYS AND FILES IN PROGRAMS

7.1 INTRODUCTION AND SUMMARY

The subject of arrays and files is very often considered as rather advanced and really the domain of the more experienced programmer. However, with CPC464 this is really not so—*arrays*, which are essentially easy-to-access storage 'boxes' for the values we wish to enter, use and process in our programs, and *files*, which allow us to keep permanent records of input data, results, etc., are very easy to use. They provide much greater scope for writing more interesting programs and allows us to extend our activities to a very much wider field of applications.

In this chapter we first explain the meaning of arrays and explain how they may be generated using the DIM statement. We then consider how they may be used: how data may be fed in, processed and output. We consider also application to sorting numerical and alphabetic information in order. Finally the idea of a file for keeping permanent records is introduced and we explain how a file can be created and accessed using the cassette datacorder of the Amstrad CPC464.

7.2 ARRAYS: THEIR MEANING AND DECLARATION USING THE DIM STATEMENT

So far we have used individual variables in our programs. Each variable we used was given a name—its identifier—and this identifier was used to identify a storage location, a 'box' or 'container', to store the data assigned to the variable at a convenient place in the computer's memory. An array is used in the same way and allows us to declare a whole number of storage locations using a single DIMension line statement.

Fig. 7.1 provides a pictorial representation which should help you in understanding the concept of an array as a means of storing not one but several different items of data. Every element in the array can be used to store data and can be individually accessed.

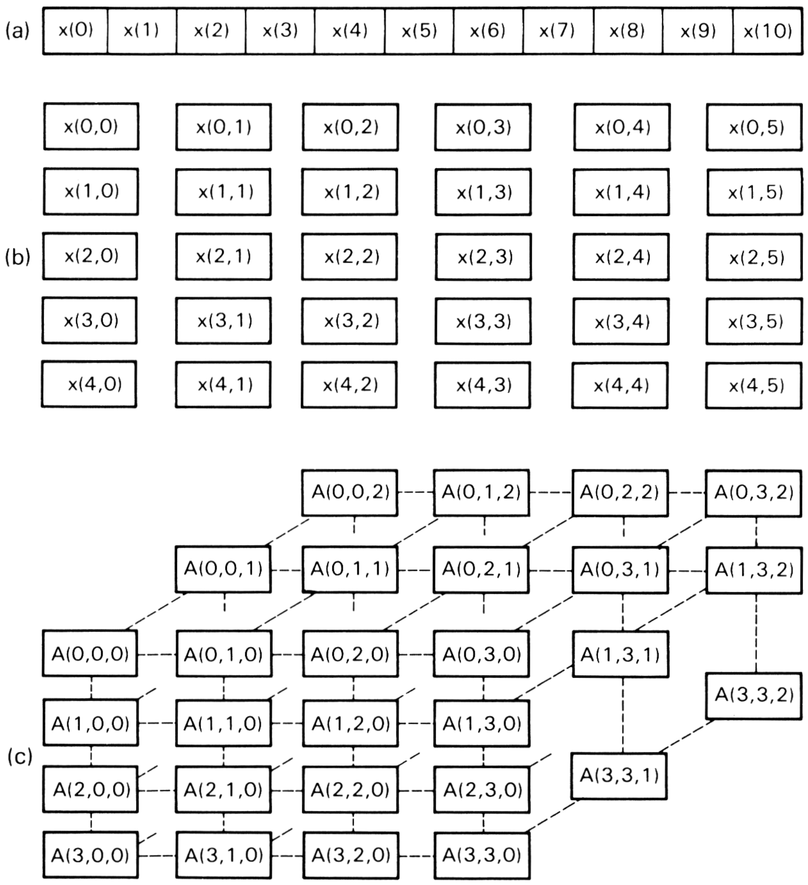


Fig. 7.1 Pictorial representation of arrays: each 'box' in the array can be considered as an individual storage location for an array element variable. (a) Example of a one-dimensional array, generated using the DIMension statement: 100 DIM x (10). (b) Example of a two-dimensional array, generated using the DIMension statement: 120 DIM x (4,5). (c) Example of a three-dimensional array, generated using the DIMension statement: 140 DIM A (3,3,2).

Each array we use in a program is given its own general identifier, just as we assign names or identifiers to variables. Thus we can give for an associated group of variables a collective name—the array identifier—rather than a whole list of different identifiers. We can easily refer to an individual element (i.e. storage location or 'box') by the array identifier plus number subscript(s), the number(s) specified within the brackets in fig. 7.1

The use of arrays saves a great amount of writing space in preparing programs and is very useful for inputting, processing and outputting large volumes of information.

Arrays are created in programs using the DIMension statement:

```
100 DIM array identifier (N)            . . . for 1-dimensional arrays
200 DIM array identifier (N1, N2)          . . . for 2-dimensional arrays
300 DIM array identifier (N1, N2, N3)         . . . for 3-dimensional arrays
```

The numbers *N, N1, N2, N3* within the brackets specify the overall size or DIMensions of the array, i.e. the total number of elements (storage locations for variable values) in the array. When the computer 'sees' a DIM statement it puts aside memory space for the array elements as specified by the numbers within the brackets.

Arrays can be defined for numeric data (real and integer) and for strings. The array identifiers for integer data must end with the % symbol and for strings with the $ sign, just as for 'normal' variables.

Examples of the DIM statement

(1) *One-dimensional arrays*
The statement,

```
100 DIM x (10)
```

defines a one-dimensional array. The array identifier is *x* and the 'dimension' of the array is 10. A dimension of 10 creates 10+1=11 elements (see fig. 8.1(a)). Each element is identified by the array identifier and a subscript or index enclosed within brackets. The subscripts run from 0 to the number specified in the DIM statement. Thus the above statement creates the following 11 elements:
x(0) , *x*(1) , *x*(2) , *x*(3) , *x*(4) , *x*(5) , *x*(6) , *x*(7) , *x*(8) , *x*(9) and *x*(10).
for 'number' data.

The statement,

```
110 DIM name$ (20)
```

would create a string array, identifier name$, dimension 20 and therefore consisting of 20+1 elements for string variable data storage.

(2) *Two-dimensional arrays*
The statement,

```
120 DIM x (4,5)
```

creates a two-dimensional array for numeric (real) variable data consisting of a total of (4+1) × (5+1) = 30 elements (see fig. 7.2 (b)). In this case each element is identified by a pair of subscripts, the first of which can be thought as specifying the row and the second as specifying the column of the element in the array.

(3) *Three-dimensional arrays*
The statement,

```
140 DIM A (3,3,2)
```

creates a three-dimensional array, identifier in this case A, which consists of a total of (3+1)×(3+1)×(2+1)=48 elements for real variable data.

7.3 PROGRAM EXAMPLES ILLUSTRATING USE AND APPLICATION OF ARRAYS

Basically arrays are used, just like individual variables, for storing and processing information. Values for the array elements can be input from the keyboard using the INPUT statement or READ in from DATA statements within the program.

The following examples illustrate some basics concerning the handling and application of arrays

(1) This example illustrates how data can be READ into the array elements and how the stored data in the elements may be accessed and displayed

```
10 REM *** Array demo 1 ***
20 REM * READing DATA into an array *
30 CLS
40 DIM number(10)
50 FOR i = 0 TO 10
60 READ number(i)
70 NEXT i
80 REM * Displaying contents of array *
90 PRINT "item no.","number"
100 FOR i = 0 TO 10
110 PRINT i,number(i)
120 NEXT i
200 DATA 21,72,34,45,51,67,78,83,90,11,212
```

Line 40 defines a 10+1 element array. The FOR loop, lines 50 to 70, READs in data to each of the 11 array elements. The display of this information is produced by the FOR loop, lines 100 to 120. Thus on running the program the following list will be displayed.

```
item no.      number
 0             21
 1             72
 2             34
 3             45
 4             51
 5             67
 6             78
 7             83
 8             90
 9             11
 10            212
```

(2) This second example illustrates the use of both numeric and string arrays, defined respectively at lines 40 and 45. Once again FOR loops are used to READ in and PRINT out the array element data

```
10 REM *** Array demo 2 ***
20 REM * READing DATA into an array
30 CLS
40 DIM number(10)
45 DIM name$(10)
50 FOR i = 0 TO 10
60 READ name$(i),number(i)
70 NEXT i
80 REM * Displaying contents of both arrays *
90 PRINT "item no."," name","number"
95 PRINT
100 FOR i = 0 TO 10
110 PRINT i,name$(i),number(i)
120 NEXT i
200 DATA sport car ,21,saloon 1L,72
210 DATA saloon 2L,34,saloon 3L,45
220 DATA saloon XL,51,estate 1.5L,67
230 DATA estate 3L,78,hatch-back,83
240 DATA tyre T1,90,tyre t2,11
250 DATA battery,212
```

RUN [ENTER]

```
item no.       name          number

 0            sport car        21
 1            saloon 1L        72
 2            saloon 2L        34
 3            saloon 3L        45
 4            saloon XL        51
 5            estate 1.5L      67
 6            estate 3L        78
 7            hatch-back       83
 8            tyre T1          90
 9            tyre t2          11
 10           battery          212
```

(3) This program is a modification of (1) and (2) to include a full list of 3 items (name, number and price) and also to work out the total value of the items.

Three arrays price (10), number (10), name$(10) are defined at lines 35, 40 and 45 respectively. A FOR loop and the READ statement is used to feed in values to the respective elements from the DATA statements 200 to 250. The FOR loop at lines 100 to 120 PRINTs out the information and at the same time makes a continuous calculation of number (*i*) x price (*i*) (see line 115) to determine the total value (sum) of all items on the list.

```
10 REM *** Array demo 3 ***
20 REM * READing DATA into an array *
30 CLS
35 DIM price(10)
40 DIM number(10)
45 DIM name$(10)
50 FOR i = 0 TO 10
60 READ name$(i),number(i),price(i)
70 NEXT i
80 REM * Displaying contents of all arrays *
85 sum=0
90 PRINT "item no."," name","number";TAB(35)"price"
95 PRINT
100 FOR i = 0 TO 10
110 PRINT i,name$(i),number(i);TAB(34)price(i)
115 sum=sum+number(i)*price(i)
120 NEXT i
130 PRINT
140 PRINT "Total value =";sum
200 DATA sport car ,21,5450,saloon 1L,72,3985
210 DATA saloon 2L,34,5675,saloon 3L,45,7950
220 DATA saloon XL,51,11250,estate 1.5L,67,5950
230 DATA estate 3L,78,8750,hatch-back,83,6980
240 DATA tyre T1,90,23.50,tyre t2,11,38.90
250 DATA battery,212,36.55
```

RUN [ENTER]

```
item no.        name           number   price

 0             sport car        21      5450
 1             saloon 1L        72      3985
 2             saloon 2L        34      5675
 3             saloon 3L        45      7950
 4             saloon XL        51      11250
 5             estate 1.5L      67      5950
 6              estate 3L      78      8750
 7              hatch-back     83      6980
 8              tyre T1        90      23.5
 9              tyre t2        11      38.9
 10             battery        212     36.55

Total value = 3196601.5
```

is the display obtained on running the program.

(4) The following example illustrates the READing in and display of numerical data using a two-dimensional array. Once again we use FOR loops but in this case one FOR loop within another one. The '*i*' loop sets the row while the '*j*' loop runs across the row reading the various columns. Try running the program. You will obtain 6 rows (i.e. 5+1) of 5 columns (4+1) of the data contained in the DATA statements

```
10 REM *** Two-dimensional array example ***
20 CLS
30 DIM x(5,4)
40 REM ** READing in DATA **
50 FOR i = 0 TO 5
60 FOR j = 0 TO 4
70 READ x(i,j)
80 NEXT j
90 NEXT i
100 REM ** Display of Array Element values **
110 FOR i = 0 TO 5
120 FOR j = 0 TO 4
130 PRINT x(i,j);
140 NEXT j
150 PRINT
160 NEXT i
200 DATA 34.6,82.8,67.8,21.8,63.7
210 DATA 90.8,76.4,62.7,23.8,41.4
220 DATA 78.3,55.5,45.3,67.2,11.9
230 DATA 66.8,98.5,72.5,81.9,45.8
240 DATA 76.5,43.9,73.6,64.8,92.7
250 DATA 21.7,34.6,83.7,61.3,56.4
```

(5) Here is an example of the use of arrays with strings: a simple French–English vocabulary test.

Arrays are used to store the English and corresponding French words. On running the program you are asked to type in the French word for a given English one. The computer tells you if you are right. If you are wrong, the computer gives the correct French word.

Trying running the program and then perhaps extend it for more words.

```
10 REM *** English to French test ***
20 REM ** READing in English words **
30 CLS
40 DIM English$(8)
50 FOR i = 0 TO 8
60 READ English$(i)
70 NEXT i
80 REM ** READing in French words **
90 DIM French$(8)
100 FOR j = 0 TO 8
110 READ French$(j)
120 NEXT j
130 N=0
140 IF n>8 THEN END
150 PRINT "Enter French word for ";English$(N)
160 INPUT answer$
170 IF answer$ = French$(N) THEN 180 ELSE 220
180 PRINT "Well done, you're correct"
190 N=N+1
200 PRINT
210 GOTO 140
220 PRINT "Sorry, you're wrong"
230 PRINT "The French word for "
240 PRINT English$(N);" is ";French$(N)
250 PRINT "***************"
260 N=N+1
270 PRINT
280 GOTO 140
290 DATA man,dog,hand,tree,sea,wind
300 DATA thanks,wine,bread
310 DATA homme,chien,main,arbre,mer,vent
320 DATA merci,vin,pain
```

7.4 SORTING NUMERICAL DATA IN ORDER

An important and very interesting application for which the computer is ideally suited is in the sorting of data in order.

The following program segment may be used to sort numerical data in ascending order:

```
500 REM *** Bubble sort routine ***
510 FOR k = 1 TO n-1
520 FOR l = k+1 TO n
530 IF x(l) > x(k) THEN 570
```

```
540 temp=x(l)
550 x(l) = x(k)
560 x(k) = temp
570 NEXT l
580 NEXT k
```

The data to be sorted is first fed into an array $x(n)$ consisting of elements $x(1)$, x(2) . . . $x(n)$. The above routine is then brought into action. It starts by comparing the first data item $x(1)$ with its neighbour $x(2)$ and if the first item is greater then the two values are 'swopped' in position in the array. The comparison–swop action then continues with first and third elements and so on until the minimum value heads the list, i.e. $x(1)$ stores the minimum value. The routine is then repeated with the second item in the list and so on to the last term, so at the end of the second sort sequence the first two terms are in order. Execution then moves on to the third term etc. and the whole comparison–swop action repeated until we finally end up with all elements in ascending order. A pictorial representation for the simple case of sorting five values in ascending order is illustrated in fig. 7.2.

The 'bubble sort' routine, so called because the data 'bubbles' up or down in the sort, may be easily changed to sort in descending order. Simply change the greater than symbol (>) in line 530 to the less than symbol (<), i.e.
530 IF x(l) < x(k) THEN 570

The following program shows the application of the bubble sort method to sort 30 numbers contained in DATA statements:

```
10 REM *** Sorting numbers in order ***
20 REM **  First storing numbers in an array **
30 DIM x(30)
40 FOR i = 1 TO 30
50 READ x(i)
60 NEXT i
70 n=30 :REM no.of terms to be sorted
80 GOSUB 500
90 REM ** display of ordered numbers **
100 FOR i = 1 TO 30
110 PRINT x(i);
120 NEXT i
130 END
500 REM *** SORT subroutine ***
510 FOR k = 1 TO n-1
520 FOR l = k+1 TO n
530 IF x(l) > x(k) THEN 570
540 temp=x(l)
550 x(l) = x(k)
```

```
560 x(k) = temp
570 NEXT l
580 NEXT k
590 RETURN
600 DATA 89,34,87,9,34,76,12,88,91,10
610 DATA 9,32,71,94,61,85,8,43,21,42
620 DATA 4,87,41,76,43,95,110,39,97,1
```

On running the program you will obtain within 10 or so seconds a display of the numbers in ascending order, i.e.
1 4 8 9 9 10 12 . . .

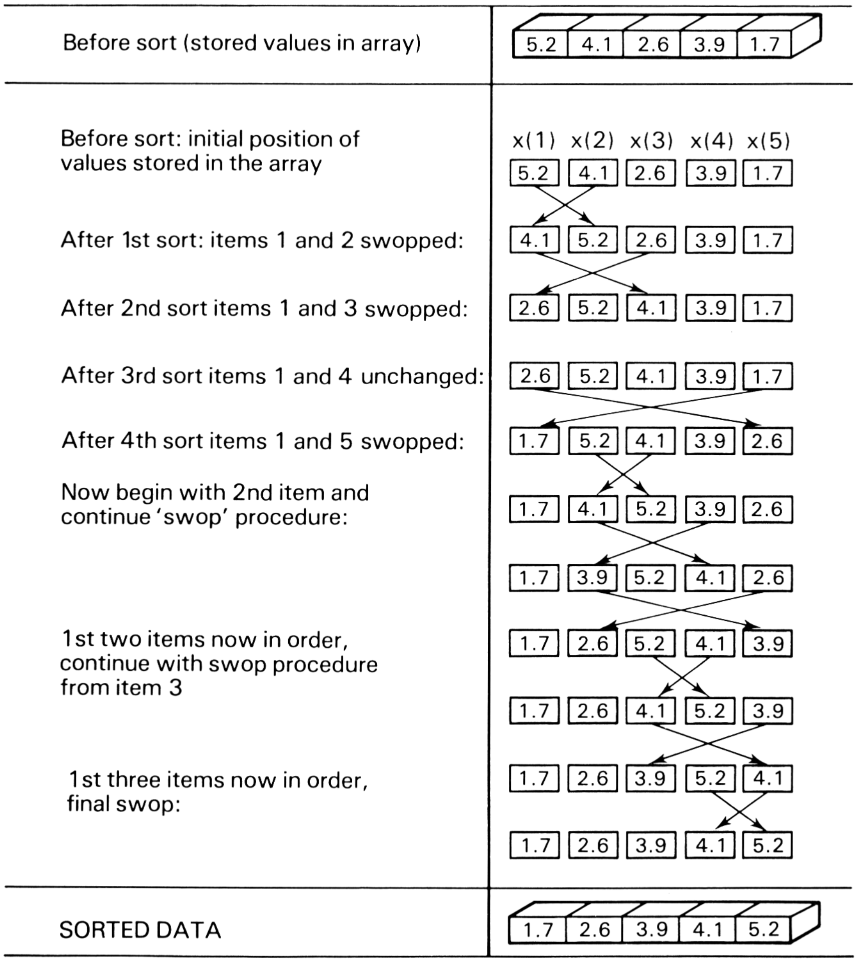

Before sort (stored values in array)
5.2 4.1 2.6 3.9 1.7
Before sort: initial position of values stored in the array
x(1) x(2) x(3) x(4) x(5)
5.2 4.1 2.6 3.9 1.7
After 1st sort: items 1 and 2 swopped:
4.1 5.2 2.6 3.9 1.7
After 2nd sort items 1 and 3 swopped:
2.6 5.2 4.1 3.9 1.7
After 3rd sort items 1 and 4 unchanged:
2.6 5.2 4.1 3.9 1.7
After 4th sort items 1 and 5 swopped:
1.7 5.2 4.1 3.9 2.6
Now begin with 2nd item and continue 'swop' procedure:
1.7 4.1 5.2 3.9 2.6
1.7 3.9 5.2 4.1 2.6
1st two items now in order, continue with swop procedure from item 3
1.7 2.6 5.2 4.1 3.9
1.7 2.6 4.1 5.2 3.9
1st three items now in order, final swop:
1.7 2.6 3.9 5.2 4.1
1.7 2.6 3.9 4.1 5.2
SORTED DATA
1.7 2.6 3.9 4.1 5.2

Fig. 7.2 Pictorial representation of 'bubble-sort' method.

7.5 SORTING IN ALPHABETIC ORDER

It is a straightforward matter to adapt a numerical sort routine to sort string variable data (e.g. names) into alphabetic order.

Firstly the arrays to hold the data must obviously be defined as string variable arrays so include the $ sign in the array identifier. Secondly use the ASC standard function to convert the first letter of the string variable value to its number code. By doing this number comparison may be made. Remember: ASC (X$(I)) returns the ASCII code for the first character of the string variable X$(I) value; and that these codes are ordered and cover the ranges,
65,66, . . . 90 for A,B, . . . Z
97,98, . . . 122 for a,b, . . . z

Thus modifying the numeric bubble-sort routine to a form for sorting string data in alphabetic order leads to:

```
500 REM *** ALPHABETIC  SORT subroutine ***
510 FOR k = 1 TO n-1
520 FOR l = k+1 TO n
530 IF ASC(x$(l)) > ASC(x$(k)) THEN 570
540 temp$=x$(l)
550 x$(l) = x$(k)
560 x$(k) = temp$
570 NEXT l
580 NEXT k
```

The following program illustrates its use. Array x$(30) is used to store up to 30 names, each name being stored in one of the array elements x$(1), x$(2) . . . x$(30). Since there are only 10 names in our example only 10 elements are required.

```
10 REM *** Sorting in ALPHABETIC order ***
20 REM **  First storing numbers in an array **
25 CLS
30 DIM x$(30)
40 FOR i = 1 TO 10
50 READ x$(i)
60 NEXT i
70 n=10 :REM no.of names to be sorted
80 GOSUB 500
90 REM ** display of ordered numbers **
100 FOR i = 1 TO 10
110 PRINT x$(i)
120 NEXT i
130 END
```

```
500 REM *** ALPHABETIC  SORT subroutine ***
510 FOR k = 1 TO n-1
520 FOR l = k+1 TO n
530 IF ASC(x$(l)) > ASC(x$(k)) THEN 570
540 temp$=x$(l)
550 x$(l) = x$(k)
560 x$(k) = temp$
570 NEXT l
580 NEXT k
590 RETURN
600 DATA White,Brown,Smith,Jones,Avis
610 DATA Clark,Yeabsly,Dennis,Mead,Law
```

On running the program you will obtain the names listed in alphabetic order, i.e.

```
Avis
Brown
Clark
Dennis
Jones
Law
Mead
Smith
White
Yeabsly
```

7.6 USING FILES IN PROGRAMS

In all our programming work so far we have INPUT data to be processed by our program directly either from the keyboard or by using READ statements and OUTPUT the required results to the monitor or tv display for a 'soft copy' or for a 'hard copy' to a printer. As soon as the program has been executed all keyboard entered data and any intermediate results are effectively lost. How can we store, for example, input data on a permanent basis and feed it directly into our program when required, rather than using the keyboard or DATA statement as the source? How can we store data obtained from one part of a program for processing in another (without, of course, using arrays, which, although ideally suited to this task, may take up excessive and valuable memory space in the computer)? How can we store our output data which may be required for subsequent use as input data to another program?

These problems may be overcome by storing input, output and intermediate 'transferable' data in files in an auxiliary memory device, i.e. on the magnetic tape of a cassette or, if you are lucky enough to have a disc drive unit, on disc. We can then, for example, prepare our input data independently, save it on tape and 'read in' the data when required in program execution. Likewise if we wish to save our output 'PRINT' results we can direct these to be 'written' to a cassette tape or disc.

The CPC464 handles *sequential* files where the individual items of data can only be accessed in the order in which they are stored. A sequential file can only be examined one element at a time, starting from the beginning. Unlike an array, whose individual elements can be directly or 'randomly' accessed, data in a sequential file cannot. File size, however, is not specified and a file may be allowed to expand, within the limits of available memory storage space, to any size.

With these introductory ideas in mind, let us now see how easy it is to use files with the CPC464 and its cassette recorder-player.

To create a file and write to a file on cassette

(1) Place cassette tape in the CPC464 datacorder and note counter (trip meter) reading for future reference.
(2) To 'open' a file ready for writing data to, use a statement of the form,
100 OPENOUT "name of file"
(3) To write, i.e. output data, to the file use statements of the form,
110 PRINT #9, "strings"
120 PRINT #9, x, y, z
e.g. 130 PRINT #9, 47.25
will store 47.25 as the first number item of the file. Note in the Amstrad CPC464 channel 9 (i.e. #9) is reserved for transmitting information to and from files stored on cassette tape.
(4) Finally you must always remember to 'close' your file when all entries are completed. This is accomplished by a statement of the form,
140 CLOSEOUT

To access and INPUT data from a cassette file

(1) Place cassette on which file is stored in the CPC464 datacorder and preferably wind on to beginning of wanted file.
(2) To open the file for 'INPUTing' from, use the statement,
150 OPENIN "name of file"
(3) To input data from the file, use statements of the form,
160 INPUT #9,x . . . for numeric data
160 INPUT #9, x$ . . . for string data

These statements will assign *x* or for strings x$ the first data element stored on the file.
(4) Finally, you must close your file when you have completed all your input of data. This is simply done by the statement,
170 CLOSEIN

Examples: using files

(1) This basic program demonstrates how to open a new file and output data to the file. The second part of the program accesses the file just created and inputs the data to the computer from the file and displays the file data on the screen.

```
10 REM *** Opening a file demo ***
20 CLS
30 PRINT "Cassette in place ?"
40 PRINT "note trip count number for ref"
50 PRINT
60 OPENOUT "data 1"
70 FOR n = 1 TO 20
80 PRINT #9,n
90 NEXT n
100 CLOSEOUT
105 PRINT:PRINT
110 REM *** Inputting from file "data 1" ***
120 PRINT "Rewind cassette to beginning"
130 PRINT "of file 'data 1' just created"
135 PRINT"then..."
140 OPENIN "data 1"
150 FOR n = 1 TO 20
160 INPUT #9,X
170 PRINT x
180 NEXT n
190 CLOSEIN
```

Statement 60 opens a new file which we called *data 1*. Up to 16 characters can be used for file names and spaces may be included. Data is written to this file using a FOR loop and the PRINT #9, . . . statement (see lines 70, 80, 90). In this example we are just outputting the numbers *n* from 1 to 20. The file is closed at line 100.

The second part of the program (essentially lines 140 to 190) inputs the data stored in file *data 1* to the computer and using the simple PRINT statement and FOR loop each item is displayed on the screen.

(2) The second example given below is similar except that the data to be stored in the file *results 1* is input from the keyboard (see line 50). Each of the 10 'results' input is saved in the file *results 1*.

The *results 1* file is opened for 'reading from' at line 160. As well as inputting the 'results' on file and displaying each one on the screen, their sum is computed (see line 200). Finally the average of the 10 results is displayed.

```
10 REM *** File demo 2 ***
20 CLS
30 OPENOUT "results 1"
40 FOR n = 1 TO 10
50 INPUT "Recorded result";x
60 PRINT #9,x
70 NEXT n
80 CLOSEOUT
90 PRINT:PRINT
100 REM *** Inputting from file "results 1" ***
110 PRINT "Ensure cassette is rewound so file"
120 PRINT " 'results 1' can be accessed "
130 PRINT
140 PRINT "Results are input on following"
150 PRINT "the instructions...":PRINT
160 OPENIN "results 1"
170 FOR n = 1 TO 10
180 INPUT #9,x
190 PRINT x
200 sum = sum + x
210 NEXT n
220 CLOSEIN
230 PRINT
240 PRINT "Average of results =";sum/10
```

(3) EOF function: End Of File

All files are terminated by an 'End-Of-File' character so when in reading data from a file this is reached, it can be recognised by the computer and used to terminate the reading. The EOF function is very useful in this respect, especially, as is usually the case, when we do not know the exact number of items in the file.

Thus rather than using a FOR loop for inputting data from a file we can use the WHILE loop with the condition, "WHILE NOT EOF" to control the input. For example, the FOR loop in example (2) may be replaced by:

```
170 WHILE NOT EOF
180 INPUT #9,x
190 PRINT x
200 sum = sum + x
210 WEND
```

(4) Here is an example of keeping a telephone directory on file. The first part of the program allows you to create the file and enter names and numbers. Last 'terminating' name is *zzz*. The second half allows you to display the list.

```
10 REM *** Phone no. file ***
20 REM ** Creating list **
30 CLS
40 OPENOUT "phone list"
50 INPUT "surname and phone no.";name.no$
60 WHILE name.no$ <> "zzz"
70 INPUT "surname and phone no.";name.no$
80 PRINT #9,name.no$
90 WEND
100 CLOSEOUT
110 REM ** To display list from file **
120 PRINT:PRINT
130 PRINT "Ensure file in position on cassette"
140 OPENIN "phone list"
150 INPUT #9,name.no$
160 CLS
170 PRINT name.no$:PRINT #8,name$
180 WHILE name.no$ <> "zzz"
190 INPUT #9,name.no$
200 PRINT name.no$ : PRINT #8,name.no$
210 WEND
220 CLOSEIN
```

Exercise problems 7

(1) Write a program which allows you to enter 10 named items and their prices from the keyboard and then display a list of the complete set of items and prices.

(2) Modify the above program so that your display contains item, price, number of items, sub-total of value (i.e. number × price) and, at the end, the complete value of all items.

(3) Using a two-dimensional array write a program which will display a complete table of results for 32 students each having sat the same 4 examinations.

(4) Write a program using arrays to store your data and the graphics commands PLOT, DRAW, ORIGIN, etc., which can plot an *x*-*y* graph to a suitable scale.

As a guide to forming the program, you might consider the following approach:
(a) Define arrays for the *x*,*y* data to be stored, and provide input statement so that the data may either be INPUT or READ.
(b) Input the *x*,*y* data for each point to be plotted.

(c) Decide on a suitable scale factor and origin.
(d) Draw in the *x* and *y* axes using DRAW statements.
(e) Plot your points using the PLOT statement.

(5) Design a program that can accept up to 50 numbers in the range 0 to 100 and sort them in numerical order and display the following:
(a) the maximum, minimum and middle (mode) value
(b) The data values in the following bands:
band 1: 0 to 25; band 2: 26 to 50; band 3: 51 to 75; band 4: 76 to 100.

(6) Write a program that accepts names and phone numbers from the keyboard and then prints out the complete name–phone number list.

(7) Modify the last program so that your print-out of names is in alphabetical order.

8

SOME PRACTICAL PROGRAMS

8.1 INTRODUCTION AND SUMMARY

This final chapter presents a number of programs for general use. They are all simply constructed using the BASIC statements of previous chapters and easy to understand. All the programs can be run directly to perform their tasks but can also be easily modified to provide any extra facility you might need.

The programs included cover the following applications: conversion of weights, lengths, currency, etc.; production of standard letters, multiple copies; checking bills; calculation of tax; interest and return on investments; loan repayment calculations.

8.2 A CONVERSION PROGRAM FOR COMMON MEASURES

To convert units *X* to *Y* multiply by	*X*	*Y*	To convert units *Y* to *X* multiply by
4.0469×10^{-3}	acres	square kilometres	247.1
2.8317×10^{-2}	cubic feet	cubic metres	35.315
0.3048	feet	metres	3.2808
4.546	gallons	litres	0.21998
0.7457	horse power	kilowatts	1.3410
25.4	inches	millimetres	0.03937
1.6093	miles	kilometres	0.62137
0.35402	miles/gallon	kilometres/litre	2.8247
0.56825	pints	litres	1.7598
0.45359	pounds	kilograms	2.2046
0.83613	square yards	square metres	1.1960
1016	tons	kilograms	9.8421×10^{-4}

Fig. 8.1 Conversion table for some common quantities.

The table of fig. 8.1 summarises the conversion factors for converting some commonly used quantities from one system of units (e.g Imperial) to another (e.g. metric). Rather than remember these, why not use the following program which effects conversions automatically?

```
10 REM *** Conversion of units program ***
20 CLS
30 PRINT "To convert :"
40 PRINT "1 gallons to litres...enter 1"
50 PRINT "2 litres to gallons...enter 2"
60 PRINT "3 pounds to kilos...enter 3"
70 PRINT "4 kilos to pounds...enter 4"
80 PRINT "5 feet and inches to meters...enter 5"
90 PRINT "6 meters to feet and inches...enter 6"
100 PRINT "7 degrees C to F...enter 7"
110 PRINT "8 degrees F to C...enter 8"
120 INPUT "now enter your choice ";n
130 PRINT
140 ON n GOTO 160,180,200,220,240,280,320,340
150 PRINT "incorrect entry ": GOTO 120
160 INPUT "enter no. of galls to be converted ";x
170 PRINT x;" galls = ";x*4.546;" litres":END
180 INPUT "enter no. of litres ";x
190 PRINT x;" litres = ";x*0.21998;" galls":END
200 INPUT "enter no. of pounds ";x
210 PRINT x;" pounds = ";x*0.4536;" kilos":END
220 INPUT "enter no. of kilos ";x
230 PRINT x;" kilos =";x*2.2046;" pounds":END
240 INPUT "enter no. of feet ";f
250 INPUT "enter no. of inches ";i
260 PRINT f;" feet";i;" inches = ";
270 PRINT (f+i/12)*0.3048;" meters":END
280 INPUT "enter no. of meters ";x
290 PRINT x;" meters = ";x*3.2808;" feet"
300 PRINT "= ";INT(x*3.2808);" feet";
310 PRINT (x*3.2808-INT(x*3.2808))*12;" inches":END
320 INPUT "enter no. of degrees C ";x
330 PRINT x;" degrees C = ";x*9/5+32;" degrees F":END
340 INPUT "enter no. of degrees F ";x
350 PRINT x;" degrees F = ";(x-32)*5/9;" degrees C":END
360 END
```

On running the program you will first be asked to input the conversion required—the program caters for 8 cases but more, of course, may be added. After this all you have to do is input the quantity and the conversion is carried out immediately.

8.3 PROGRAMS FOR CURRENCY CONVERSIONS

Fig. 8.2

Here are two useful programs for converting pounds sterling to other currencies and vice versa.

The first program provides a direct means of carrying out the conversion but does require you to enter the exchange rate first.

```
10 REM *** Currency conversion ***
20 REM ***    Short program    ***
30 CLS
40 PRINT "Enter 1 for  £to foreign"
50 PRINT "Enter 2 for conversion back to £"
60 INPUT n
70 IF n>2 THEN PRINT "incorrect entry":GOTO 40
80 PRINT:PRINT
90 INPUT "Current exchange rate £1=";rate
100 IF n=1 THEN 120
110 IF n=2 THEN 150
120 INPUT "No. of  ";pounds
130 PRINT "£ ";pounds;" = ";pounds*rate
140 END
150 INPUT "Amount of foreign currency";A
160 PRINT A;" currency = £ ";ROUND(A/rate,2)
170 END
```

The second program contains all the required data within the program for 18 European and American countries. On running the program the 18 countries and corresponding exchange rates are displayed. Update this information if required by amending the DATA statements.

To carry out your conversion all you have to do is to input the country's name (remember, begin this with a capital letter), select 1 or 2 for type of conversion and then input the amount. The result is displayed immediately.

```
10 REM *** Currency conversion program ***
20 REM *** Display of data stored ***
30 DIM land$(18): DIM rate(18)
40 CLS
```

```
50 PRINT "Country";TAB(20)"Exchange rate"
60 PRINT
70 FOR n=1 TO 18
80 READ land$(n),rate(n)
90 PRINT land$(n);TAB(20)rate(n)
100 NEXT n
110 PRINT
120 PRINT "Add data if country not included"
130 REM *** conversion calculations ***
140 PRINT
150 INPUT "Country required ";country$
160 FOR n=1 TO 18
170 IF land$(n)=country$ THEN p=n
180 NEXT n
190 CLS
200 PRINT "enter 1 for £ to foreign"
210 PRINT "enter 2 for foreign to  "
220 INPUT "1 or 2";k
230 INPUT "amount of money";A
240 IF k=1 THEN 260
250 IF k=2 THEN 280
260 PRINT "£ ";A;" = ";A*rate(p)
270 END
280 PRINT A;" foreign currency = £";ROUND(A/rate(p),2)
290 END
300 DATA Austria,25.40,Belgium,73,Canada,1.57
310 DATA Denmark,13.08,France,11.14,Germany,3.62
320 DATA Greece,160,Ireland,1.175,Italy,2240
330 DATA Malta,0.615,Holland,4.09,Norway,10.56
340 DATA Portugal,197,Spain,198,Sweden,10.42
350 DATA Switzerland,2.99,USA,1.1875,Yugoslavia,295
```

8.4 PRODUCING MULTIPLE COPIES, STANDARD LETTERS, ETC.

```
Dear Sir   Dear Madam   Dear Sir   Dear Madam
Dear Sir   Dear Madam   Dear Sir   Dear Madam
Dear Sir   Dear Madam   Dear Sir   Dear Madam
Dear Sir   Dear Madam   Dear Sir   Dear Madam
Dear Sir   Dear Madam   Dear Sir   Dear Madam
```

Fig. 8.3

It is a fairly easy matter to construct programs to print out multiple copies of memos, standard letters, etc. Obviously you require access to a printer to direct your program output to be 'printed' as hard copy. The CPC464 has a standard Centronics parallel port interface and it is a simple matter to connect up a printer. Amstrad supply their own model. The Epson RX, MX and FX series of printers are also excellent choices.

Here are two useful programs. The first provides a means of obtaining a given number of copies of a general note. All 'note' PRINT statements are directed to the printer using PRINT #8, ". . .output to printer". Remember channel 8 (#8) is the internal channel used to communicate from the CPC464 computer to the printer.

```
10 REM *** Production of number of copies ***
20 CLS
30 INPUT "number of copies required";n
40 PRINT "Printing /display follows"
50 FOR x=1 TO n
60 PRINT #8:PRINT #8
70 PRINT #8,"--------------------------------"
80 PRINT #8,"*** SALES 11 April at STOWHAM ***"
90 PRINT #8,"..................."
100 PRINT #8,"...........details here.........."
110 PRINT #8,"..................."
120 PRINT #8,"-------------end-------------"
130 NEXT x
```

The second program illustrates how a standard letter can be produced with the ability to insert common information such as dates, times, places, names, etc. A further advantage is that names to whom the letter is to be sent can be entered from the keyboard, stored in an array and inserted automatically as the letter is being printed.

The program has been written to output all PRINT data to the screen so you can first check the letter contents and ensure all input data has been correctly inserted. When the checks have been made and the program is operating correctly change all the relevant 'letter' PRINT statements to the printer output form, i.e. PRINT #8, ". . .".

```
10 REM *** Standard letter ***
20 CLS
30 PRINT "enter the following for inclusion:"
40 PRINT
50 INPUT "Date of sending letter";date$
60 INPUT "Date of planned meeting";mdate$
70 INPUT "Time of meeting";ltime$
80 INPUT "Room to be used";room$
90 REM *** Entering people's names ***
100 DIM name$(10)
110 FOR n=1 TO 10
120 PRINT "name";n;:INPUT name$(n)
130 NEXT n
140 FOR n=1 TO 10
```

```
150 PRINT:PRINT
160 PRINT TAB(20);"CPC 464 Computers Inc"
170 PRINT TAB(20);"     Ipswitch, Suffolk"
180 PRINT:PRINT TAB(25);date$
190 PRINT "Dear ";name$(n);","
200 PRINT:PRINT " The next introductory course "
210 PRINT "is on ";mdate$;" ."
220 PRINT "The time of the class is ";ltime$
230 PRINT "and will be in ";room$;" ."
240 PRINT
250 PRINT TAB(25);"G P L Alexis"
260 PRINT TAB(21);"Chief Instructor "
270 PRINT:PRINT
280 PRINT "---------------------------"
290 PRINT:PRINT:PRINT
300 NEXT n
```

8.5 A BILL-CHECKING PROGRAM

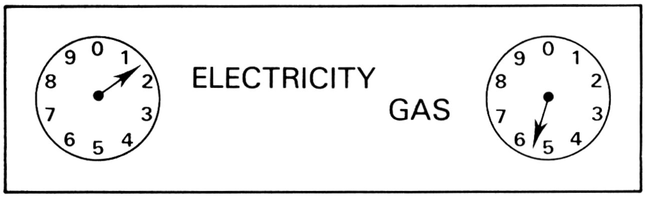


Fig. 8.4

Here is the framework of a program which allows you to check your domestic bills. You are asked to input the relevant data, such as present and previous meter readings, rate in the pound levied, etc. and you can store constant data such as standing charge, unit cost, VAT, etc. in DATA statements.

Three examples are included: house rates, and electricity and gas bills.

```
10 REM *** Bill checking program ***
20 CLS
30 PRINT "Enter ...1 for house rates"
40 PRINT "Enter ...2 for electricity"
50 PRINT "Enter ...3 for gas"
60 INPUT n
70 IF n>3 THEN PRINT "incorrect entry":GOTO 60
80 ON n GOTO 90,150,290
90 REM *** House rates ***
100 rateable.value=467
110 CLS: PRINT "rate bill check"
120 INPUT "Current rate in pound";r
```

```
130 PRINT:PRINT "Annual rates =";rateable.value*r
140 END
150 REM *** Electricity bill check ***
160 CLS:PRINT "Electricity bill check"
170 INPUT "Current meter reading";now
180 INPUT "Previous meter reading";last
190 unit.cost=4.94
200 standing.charge=6.37: vat=0
210 PRINT (now-last);" units at ";unit.cost;" P =";
220 PRINT (now-last)*unit.cost/100
230 PRINT "Standing charge =";standing.charge
240 total=standing.charge+(now-last)*unit.cost/100
250 PRINT "vat at ";vat; " % = ";vat*total/100
260 PRINT "-------------------------------"
270 PRINT "Total payable =";total*(1+vat/100)
280 END
290 REM *** Gas bill check ***
300 CLS:PRINT "Gas bill check":PRINT
310 therm.factor=1.032
320 PRINT "Enter present and previous meter readings"
330 INPUT now,last
340 unit.c=35.2: stand.c=9.9:vat=0
350 PRINT (now-last);" units at ";unit.c;" = ";
360 PRINT (now-last)*unit.c/100
370 total=(now-last)*unit.c/100+stand.c
380 PRINT "Standing charge = ";stand.c
390 PRINT "vat at ";vat; " % = ";total*vat/100
400 PRINT:PRINT "--------------------------------"
410 PRINT "Total payable =";total*(1+vat/100)
```

8.6 CALCULATION OF INCOME TAX

This program enables income tax to be easily calculated. It contains the rates for the 84/85 tax year so these will require updating when the tax rates are changed. All you have to do is enter gross income and total allowances; the program then displays the total tax due.

```
10 REM *** Income tax program ***
20 CLS
30 REM ***   Tax bands 84-85   ***
40 PRINT "taxable income","tax rate"
50 PRINT
60 PRINT "1-15400",TAB(30);" 30 % "
70 PRINT "15401-18200",TAB(30);" 40 % "
80 PRINT "18201-23100",TAB(30);" 45 % "
90 PRINT "23101-30600",TAB(30);" 50 % "
100 PRINT "30601-38100",TAB(30);" 55 % "
110 PRINT "over 38100",TAB(30);" 60 % "
```

```
120 b1=15400 : b2=18200 : b3=23100
130 b4=30600 : b5=38100
140 PRINT : PRINT
150 PRINT "Enter your gross taxable income";
160 INPUT income
170 INPUT "and your allowances ";allow
180 taxinc=income-allow
190 IF taxinc>0 AND taxinc<=15400 THEN 250
200 IF taxinc>15400 AND taxinc<=18200 THEN 270
210 IF taxinc>18200 AND taxinc<=23100 THEN 290
220 IF taxinc>23100 AND taxinc<=30600 THEN 310
230 IF taxinc>30600 AND taxinc<=38100 THEN 330
240 IF taxinc>38100  THEN 350
250 tax=taxinc*0.3
260 PRINT:PRINT "Tax due =";tax:END
270 tax=4620+(taxinc-b1)*0.4
280 PRINT:PRINT "Tax due=";tax:END
290 tax=5740+(taxinc-b2)*0.45
300 PRINT:PRINT "Tax due=";tax :END
310 tax=7945+(taxinc-b3)*0.5
320 PRINT:PRINT "Tax due=";tax :END
330 tax=11695+(taxinc-b4)*0.55
340 PRINT:PRINT "Tax due=";tax :END
350 tax=15820+(taxinc-b5)*0.6
360 PRINT:PRINT "Tax due=";tax :END
```

8.7 CALCULATION OF INTEREST AND RETURN ON INVESTMENT

Two useful programs for calculations of interest and value of investment are given below.

The first can be used to calculate how your money grows in a *regular monthly savings plan* provided by most Building Societies, Banks and Insurance groups. You input whatever monthly sum you wish to save monthly, the current interest rate and the time in years for your savings plan. The program displays the accrued value of your savings at 6 months intervals. Interest is assumed to be compounded every 6 months, as is usually the case for most savings plans. For this reason use is made of the MOD operator (see line 110). MOD provides the remainder after division of two numbers. Thus n MOD 6 = 0 when *n* = 6, 12, 18, etc., and hence line 110 is only actioned when *n* is a multiple of 6.

```
10 REM *** Interest: Regular Savings ***
20 CLS
30 INPUT "Amount to be saved each month";pm
40 INPUT "Rate of interest %";r
50 INPUT "Period of investment";t
```

```
60 PRINT "months","accrued value"
70 PRINT
80 FOR n=1 TO 12*t
90 p=p+pm
100 i=i+p*r/1200
110 IF n MOD 6=0 THEN p=p+i:i=0:PRINT n,p
120 NEXT n
130 PRINT:PRINT
140 PRINT "Total value of investment"
150 PRINT "after ";t;" years = ";p
```

This second program provides a table of values showing how a *lump sum investment* grows. All you have to do on running the program is input the lump sum invested, rate of interest, number of years of investment and whether interest is compounded every 6 or 12 months.

```
10 REM *** Lump sum investment ***
20 CLS
30 INPUT "Lump sum invested";p
40 INPUT "Rate of interest %";r
50 INPUT "Total no. of years invested";t
60 PRINT
70 PRINT "If interest compounded every 6 months"
80 PRINT "enter...6; if every 12 months, enter...12"
90 INPUT g
100 PRINT
110 PRINT "month";TAB(8)"capital value";
120 PRINT TAB(23)"interest for"
130 PRINT TAB(26)"period"
140 PRINT
150 FOR n=1 TO 12*t
160 a=p*(1+r/1200)
170 i=i+(a-p)
180 IF n MOD g =0 THEN 190 ELSE 220
190 p=p+i
200 PRINT n;TAB(8) ROUND(p,2);TAB(23) ROUND(i,2)
210 i=0
220 NEXT n
```

8.8 A LOAN REPAYMENT PROGRAM

The program given below allows you to work out immediately the state of how your loan repayments are progressing. Repayments are assumed to be made on a regular monthly basis.

On running the program you are first asked to input the amount of the loan, the monthly repayment you wish to make, and the interest rate you are going to be charged. The program then provides a display of a table of total repayments made and outstanding debt on a month by month basis until the loan is paid off.

```
10 REM *** Loan repayment program ***
20 CLS
30 INPUT "Enter total loan";loan
40 INPUT "Enter rate % p.a. charged";r
50 INPUT "Enter proposed monthly repayment";repay
60 IF loan*r/1200>repay THEN 70 ELSE 90
70 PRINT "Repayment is insufficient to cover."
80 PRINT "Increase repayment":GOTO 50
90 month=0:PRINT:PRINT
100 PRINT "month";TAB(10)"total repaid";
110 PRINT TAB(25)"outstanding debt"
120 WHILE loan>0
130 month=month+1
140 loan=loan*(1+r/1200)-repay
150 PRINT month,repay*month,ROUND(loan,2)
160 WEND
170 PRINT "Loan is paid off in "
180 PRINT month;" months"
190 PRINT
200 PRINT "Total repayments =";repay*month+loan
```

ANSWERS TO EXERCISES

Exercise 1.1

(1) 1332
(2) 160800
(3) 9.99514349
(4) 862.95
(5) 49.17647

Exercise 1.2

(1) (a)

```
PRINT 56.7/3.3 [ENTER]
17.1818182          . . . answer displayed
```

(b)

```
PRINT "56.7/3.3=";56.7/3.3 [ENTER]
```

(c)

```
PRINT USING "##.##";56.7/3.3 [ENTER]
17.18          . . . result displayed
```

(3)

```
PRINT 4.62*3.8,57.33/6.3,42378.6-33499.7 [ENTER]
4.62          9.1          8878.9          . . . results displayed
```

(4)

```
PRINT USING "#.#";PI [ENTER]
3.1          . . . result displayed
PRINT USING "#.##";PI [ENTER]
3.14          . . . result displayed
PRINT USING "#.###";PI [ENTER]
3.142          . . . result displayed
```

Exercise 1.3

(1) 4.72944
(2) 726000

(3) 6.37690E−04
(4) 8.695652
(5) 6.294135E−03

Exercise 2

(1)(a)

```
10 REM *** names and birth dates ***
20 CLS
30 PRINT "Name","Birth date"
40 PRINT
50 PRINT "John","10.11.67"
60 PRINT "Mary","14. 7.69"
70 PRINT "Peter"," 2. 3.74"
```

(b)

```
10 REM *** Items  and price list ***
20 CLS
30 PRINT "item","Price"
40 PRINT
50 PRINT "Butter","0.92 per kilo"
60 PRINT "Cheese","1.14 per lb"
70 PRINT "Bacon","1.25 per kilo "
```

(c)

```
10 REM *** League table ***
20 CLS
30 MODE 2
40 PRINT TAB(20)"*** North Division 1 ***"
50 PRINT
60 PRINT TAB(20)"P";TAB(24)"W";TAB(28)"D";
70 PRINT TAB(32)"L";TAB(36)"F";TAB(40)"A";
80 PRINT TAB(44)"Points"
90 PRINT
100 PRINT "Woodside Rovers";TAB(20)"41";TAB(24)"22";
110 PRINT TAB(28)"13";TAB(32)"6";TAB(36)"82";
120 PRINT TAB(40)"31";TAB(44)"79"
130 PRINT "Old Polyonians";TAB(20)"41";TAB(24)"20";
140 PRINT TAB(28)"14";TAB(32)"7";TAB(36)"71";
150 PRINT TAB(40)"39";TAB(44)"74"
```

(2)(a) and (b)

```
10 REM *** Volume of cuboid ***
20 CLS
30 b=2.27
40 l=5.74
50 d=1.86
60 PRINT "volume =";b*l*d
70 PRINT "-----------------"
80 REM *** Volume of cylinder ***
90 PRINT:PRINT
100 r=8.63
110 h=12.4
120 PRINT "Volume of cylinder =";3.142*r*r*h
130 PRINT "------------------------------"
```

(3)

```
10 REM *** Calculations probem 3 ***
20 CLS
30 PRINT "Answer 1 =";3.4*5.67-6.21*4.98
40 PRINT:PRINT
50 PRINT "Answer 2 =";5.37^2+4.46^2
60 PRINT:PRINT
70 p=525
80 v=72
90 c=1.4
100 PRINT "Answer 3 =";p*v^c
```

(4)

```
10 REM *** Simple average program ***
20 CLS
30 sum=20.7+14.5+12.9+13.6
40 PRINT "Average =";sum/4
50 PRINT "------------------"
```

Answer = 15.425

(5)

```
10 REM *** Compound interest program ***
20 CLS
30 p=1000:t=5:r=11
40 a=p*(1+r/100)^t
50 PRINT "Interest =";USING "###.##";a-p
60 PRINT "for p=";p,"t=";t,"r=";r
70 PRINT:PRINT
80 p=25:t=32:r=7.5
90 PRINT "Interest =";USING "####.##";a-p
100 PRINT "for p=";p,"t=";t,"r=";r
```

Answers=(a) 685.06, (b) 227.06

Exercise 3

(1)

```
10 REM *** Calculating average ***
20 CLS
30 READ x1,x2,x3,x4,x5,x6,x7,x8,x9,x10
40 READ x11,x12,x13,x14,x15,x16,x17,x18
50 READ x19,x20
60 sum= x1+x2+x3+x4+x5+x6+x7+x8+x9+x10
70 sum=sum+x11+x12+x13+x14+x15+x16+x17
80 sum=sum+x18+x19+x20
90 PRINT "Average =";sum/20
100 DATA 52,14,37,67,73,11,75,89,19,24
110 DATA 61,49,12,33,47,55,16,71,81,92
```

Answer=48.9

(3)

```
10 REM *** Conversion program ***
20 CLS
30 PRINT "to convert feet and in to metres"
40 INPUT "Enter feet,inches";ft,in
50 m=(12*ft+in)*2.54/100
60 PRINT ft;" feet ";in;" in =";m;" metres"
70 PRINT:PRINT
80 PRINT "To convert lbs and oz to kilos"
90 INPUT "Enter lbs,oz";lb,oz
100 kg=(lb+oz/16)*0.4536
110 PRINT lb;" lb ";oz; "oz = ";kg;" kg"
```

Exercise 4

(1)

```
10  REM *** password ***
20 CLS
30 INPUT "Enter password";pass$
40 IF pass$ = "4401326019" THEN 60
50 PRINT "Incorrect password":END
60 PRINT "Secret message begins, code 2/56#"
70 PRINT "01000110 10001101 00001010"
80 PRINT "01011101 01010101 %% 47/42/18 end"
```

(2)

```
10 REM *** To print copies of memo ***
20 CLS
30 INPUT "No. copies required";n
40 FOR x=1 TO n
50 PRINT "To all club members"
60 PRINT "*******************"
70 PRINT "The annual General Meeting"
80 PRINT "will take place .....
90 PRINT "**************************"
95 PRINT:PRINT:PRINT
100 NEXT x
```

(3)

```
10 REM *** READ and find average ***
20 CLS
30 INPUT "no. of terms";no.terms
40 FOR n=1 TO no.terms
50 READ x
60 sum=sum+x
70 NEXT n
80 PRINT:PRINT
90 PRINT "Average =";sum/no.terms
100 DATA 12,65,98,23,90,34,77,41,32,88
110 DATA 65,89,43,23,9,48,76,83,21,34
```

(4)

```
10 REM *** Selection of courses of action ***
20 CLS
30 PRINT "Enter 1...for action A"
40 PRINT "Enter 2...for action B"
50 PRINT "Enter 3...for action C"
60 PRINT "Enter 4...for action D"
70 PRINT "Enter 5...for action E"
80 INPUT n
90 IF n>5 THEN PRINT "incorrect entry":GOTO 80
100 PRINT:PRINT
110 ON n GOTO 120,130,140,150,160
120 PRINT "Action A....":END
130 PRINT "Action B....":END
140 PRINT "Action C....":END
150 PRINT "Action D....":END
160 PRINT "Action E....":END
```

(5)

```
10 REM *** to determine Max and Min values ***
20 CLS
30 no.terms=10 :GOSUB 80
40 no.terms=20:GOSUB 80
50 INPUT "No. of terms to be read";no.terms
60 GOSUB 80
70 END
80 REM ** Subroutine for Max and Min **
90 xmax=-1E+10:xmin=1E+10
100 FOR n=1 TO no.terms
110 READ x
120 IF x>xmax THEN xmax=x
130 IF x<xmin THEN xmin=x
140 NEXT n
150 PRINT "For ";no.terms;" terms"
160 PRINT "Max value =";xmax
170 PRINT "Min value =";xmin
180 PRINT:PRINT
190 RESTORE
200 RETURN
210 DATA 23,45,78,45,33,29,56,23,45,66
220 DATA 34,89,34,71,34,41,56,78,98,21
230 DATA 121,86,4,67,98,34,67,92
```

Exercise 5

(2)

```
10 REM *** To generate 16 random nos ***
20 REM *** in range 1 to 55 ***
30 CLS
40 FOR n=1 TO 16
50 y=RND(n)
60 PRINT INT(55*y)+1
70 NEXT n
```

(3)

```
10 REM *** User defined functions ***
15 CLS
20 REM ** function for average of 3 **
30 DEF FNAVERAGE(a,b,c)=(a+b+c)/3
40 a=78.54:b=42.74:c=82.64
50 PRINT FNAVERAGE(a,b,c)
60 PRINT FNAVERAGE(88.7,33.3,66.6)
70 PRINT FNAVERAGE(4,6,5)
80 PRINT:PRINT
90 REM ** volume of cylinder **
100 DEF FN vol(r,h)=PI*r*r*h
110 r=7:h=21:PRINT FN vol(r,h)
120 PRINT FN vol(4.5,8.9)
```

(4)(a) 30
(b) 25
(c) 26

```
(5)(a) 90     44     59     57
       45     61     42     33
(b) v     #      ;      V
little man     little man walking
```

Exercise 6

(1)(a)

```
10 REM *** Drawing a rectangle ***
15 CLS
20 MOVE 50,50
30 DRAW 50,300
40 DRAW 50,300
```

```
50 DRAW 500,300
60 DRAW 500,50
70 DRAW 50,50
```

(b)

```
10 REM *** Drawing right angled triangle ***
15 CLS
20 MOVE 0,0
30 DRAW 400,0
40 DRAWR 0,250
50 DRAW 0,0
```

(2)(a)

```
10 REM *** Drawing a circle ***
20 CLS
30 ORIGIN 320,200
40 DEG
50 FOR n=1 TO 360
60 x=150*COS(n)
70 y=150*SIN(n)
80 PLOT x,y
90 NEXT n
```

(b)

```
10 REM *** Drawing an ellipse ***
20 CLS
30 ORIGIN 200,200
40 a=100: b=70
50 FOR x=-a TO a
60 y=b/a*SQR(a^2-x^2)
70 PLOT x,y
80 PLOT x,-y
90 NEXT x
```

(3)

```
10 REM *** Drawing house ***
20 CLS
30 ORIGIN 0,0
40 MOVE 20,0
50 DRAW 600,0
60 DRAWR 0,150
```

```
70 DRAWR -580,0
80 DRAW 20,0
90 MOVE 20,150
100 DRAW 310,300
110 DRAW 600,150
120 x=60:y=40
130 GOSUB 230
140 x=200:y=40
150 GOSUB 230
160 x=500:y=40
170 GOSUB 230
180 MOVE 350,0
190 DRAWR 0,100:DRAWR 80,0 :DRAWR 0,-100
200 MOVE 500,200
210 DRAWR 0,100:DRAWR 30,0:DRAWR 0,-110
220 END
230 REM ** Drawing window **
240 MOVE x,y
250 DRAWR 40,0:DRAWR 0,40
260 DRAWR -40,0:DRAW x,y
270 RETURN
```

(4)

```
10 REM *** To plot cos x ***
20 CLS
30 ORIGIN 320,200
40 MOVE -320,0: DRAW 320,0
50 DEG
60 FOR x=-320 TO 320
70 PLOT x,100*COS(x)
80 NEXT x
90 REM *** Short delay ***
100 FOR d=1 TO 5000:NEXT d
110 REM *** To plot y=400exp(-x) ***
120 CLS
130 ORIGIN 0,0
140 MOVE 0,0:DRAW 600,0
150 FOR x= 0 TO 5 STEP 0.1
160 PLOT 100*x,400*EXP(-x)
170 NEXT x
```

(5)

```
10 REM *** To plot graph using given points ***
20 CLS
30 ORIGIN 0,0
40 MOVE 0,0:DRAW 600,0
50 MOVE 500,25:TAG:PRINT "x-axis";
60 MOVE 0,0:DRAW 0,400
70 MOVE 25,350:PRINT "y-axis";
80 FOR n=1 TO 13
90 READ x,y
100 PLOT x,y
110 NEXT n
120 DATA 0,30,30,75,60,120,90,156,120,198
130 DATA 150,224,180,237,210,228,240,201
140 DATA 270,150,300,126,330,70,360,25
```

Exercise 7

(1)

```
10 REM *** To enter and display list ***
20 CLS
30 DIM item$(10):DIM price(10)
40 FOR n=1 TO 10
50 INPUT "name of item";name$(n)
60 INPUT "price of item";price(n)
70 NEXT n
80 CLS
90 PRINT "item";TAB(15)"price"
100 PRINT
110 FOR n=1 TO 10
120 PRINT name$(n);TAB(15)price(n)
130 NEXT n
```

(2)

```
10 REM *** Problem 2: display list...etc ***
20 CLS
30 DIM item$(10):DIM price(10)
40 DIM no(10)
50 FOR n=1 TO 10
60 INPUT "name of item";name$(n)
70 INPUT "price of item";price(n)
```

```
80 INPUT "no.of items";no(n)
90 NEXT n
100 CLS
110 PRINT "item";TAB(15)"price";
120 PRINT TAB(22)"no.items";TAB(30)"value"
130 PRINT
140 FOR n=1 TO 10
150 PRINT name$(n);TAB(15)price(n);
160 PRINT TAB(22) no(n);TAB(30)price(n)*no(n)
170 sum=sum+price(n)*no(n)
180 NEXT n
190 PRINT:PRINT
200 PRINT "Total value =";sum
```

(3)

```
10 REM *** Display of exam results ***
20 CLS
30 class.size=3
40 DIM name$(class.size)
50 DIM marks(32,4)
60 FOR n=1 TO class.size
70 INPUT "name ";name$(n)
80 FOR j=1 TO 4
90 PRINT "mark ";j;:INPUT  marks(n,j)
100 NEXT j
110 NEXT n
120 FOR n=1 TO class.size
130 PRINT name$(n);TAB(12);
140 FOR j=1 TO 4
150 PRINT marks(n,j);
160 NEXT j
170 PRINT
180 NEXT n
```

(4) See section 6.6, example 3.

(5)

```
10 REM *** Solution problem 5 ***
20 CLS
30 INPUT "No. of terms";no.terms
40 DIM x(50)
50 FOR n=1 TO no.terms
60 READ x(n)
```

```
70 NEXT n
80 REM *** Sort routine ***
90 FOR k=1 TO no.terms-1
100  FOR l=k+1 TO no.terms
110 IF x(l)>x(k) THEN 150
120 temp =x(l)
130 x(l)=x(k)
140 x(k)=temp
150 NEXT l
160 NEXT k
170 PRINT:PRINT
180 PRINT "Max value =";x(no.terms)
190 PRINT "Min value =";x(1)
200 mid=INT(no.terms/2)
210 PRINT "Mid value =";x(mid)
220 PRINT:PRINT
230 n=1
240 PRINT "Values in 0 - 25 band:"
250 WHILE x(n)<=25
260 PRINT x(n);
270 n=n+1
280 WEND
290 PRINT:PRINT:PRINT "Values in 25 -50  band:"
300 WHILE x(n)>25 AND x(n)<=50
310 PRINT x(n);:n=n+1:WEND
320 PRINT:PRINT:PRINT "Values in 51 -75  band:"
330 WHILE x(n)>50 AND x(n)<=75
340 PRINT x(n);:n=n+1:WEND
350 PRINT:PRINT:PRINT "Values in 76 -100  band:"
360 WHILE x(n)>75 AND x(n)<=100
370 PRINT x(n);:n=n+1:WEND
380 DATA 78,23,67,34,23,90,2,61,9,12
390 DATA 44,82,38,67,12,31,8,44,72,56
400 DATA  17,74,93,11,67,52,83,91,62,10
410 DATA 97,32,34,63,73,39,93,32,31,13
420 DATA 56,28,94,34,81,76,7,82,9,43
```

(6)

```
10 REM *** Name and phone no. list ***
20 CLS
30 DIM name.no$(50)
40 INPUT "No. of names";nmax
50 FOR n=1 TO nmax
```

```
60 INPUT "Name and phone no.";name.no$(n)
70 NEXT n
80 CLS
90 PRINT "Names and phone numbers"
100 PRINT
110 FOR n=1 TO nmax
120 PRINT name.no$(n)
130 NEXT n
```

(7) Use above program with addition of alphabetic sort routine of section 7.5.

APPENDIX I

SUMMARY OF BASIC KEYWORDS USED IN DIRECT COMMANDS AND PROGRAM STATEMENTS

AUTO is a direct command which automatically assigns line numbers in steps of 10 commencing with 10, i.e. 10, 20, 30 . . . ; pressing the ESC key cancels the automatic line numbering

AUTO 100,20 automatically generates line numbers starting at 100 with increments of 20

BORDER ink number sets the colour of the screen border; ink number table is given in fig. 1.8

CAT displays the names of all files stored on the tape cassette currently in the datacorder (Amstrad integral cassette player)

CLOSEIN is used in conjunction with files to 'close' the file when required input data has been read into computer from the file stored on cassette (see section 7.6)

CLOSEOUT 'closes' a file when all the data from the computer has been written to the file (see section 7.6)

CLS clears the screen display

CONT 'continues' program execution after it has been halted by STOP or END statements or a BREAK command; execution CONTinues at the next line after the STOP or END statements or where BREAK was made (see section 3.4)

DATA stores data within a program; a DATA statement may appear anywhere within a program; see also READ and section 3.3

DEF FN defines a 'user-defined' function (see section 5.7)

DEG sets the 'degree' mode and is useful for finding sin x, cos x, tan x where x is specified in degrees; if DEG is not used the radian measure is assumed

DELETE is a direct command used to delete program lines

DELETE 100 deletes line 100

DELETE −100 deletes all lines up to and including 100

DELETE 100–   deletes all lines from 100 to end of program
DELETE 100–300   deletes range of lines specified
DIM   allocates memory space in the computer for arrays (see section 7.2)
DRAW x,y   draws a straight line on the screen from the current graphics cursor position to the absolute position specified by *x,y* (see section 6.2)
DRAWR xr,yr   draws a straight line from the current graphics cursor position a further (relative) *xr* units horizontally and *yr* units vertically (see section 6.3)
EDIT line number   is a direct command used to call a specific line in the program for editing (see section 2.7)
END   an END statement is used to 'end' program execution and return cursor to the screen; END is implicit in Amstrad BASIC as execution reaches the last line statement so that the last line need not contain END; useful, however for 'ending' execution after jumping in a program
EOF   End-Of-File function used to check whether file input from cassette is at the end-of-file (see section 7.6)
ERASE array identifier   ERASEs an array in a program when no longer required so that the memory space may be reclaimed for other use
FOR . . . TO . . . NEXT   repeatedly executes a group of statements a specified number of times (see section 4.3)
FRE   determines the size of the computer memory which currently remains unused by the BASIC; used as follows
FRE (number)   . . . where *number* = 0 say, returns unused memory size;
FRE (" ")   . . . forces a 'garbage' collection before returning value for unused memory space
GOSUB . . . RETURN   transfers program execution to a subroutine; subroutine is then executed and 'return' made to line immediately following GOSUB statement (see section 4.7)
GOTO   can be used both as a direct mode command and as a BASIC statement to cause a jump in program execution (see section 4.6)
IF . . . THEN
IF . . . THEN . . . ELSE   IF statements are used to make decisions in programs, i.e. to conditionally determine branch points in a program (see section 4.2)
INK   changes the paper and pen colours (see section 1.8 and figs. 1.8 and 1.9)
INPUT   inputs values to the computer directly from the keyboard (see section 3.2) or in the form INPUT#9,x or x$ from a file stored on cassette (see section 7.6)
LET   is used in assignment statements, e.g.
10 LET x=7.98 ; no longer required and LET can always be omitted (see section 2.3)

LIST is a direct command used to list the entire program currently stored in the computer

LIST #8 directs listing of program to printer (see section 2.6)

LIST −100 displays (lists) program from beginning to line 100

LIST 100− displays program from line 100 to end of program

LIST 100−400 lists program statements between lines 100 and 400

LOAD "program name" 'loads' program from cassette to computer; program name is specified within the quotation marks (see section 2.8)

LOCATE 'locates' or positions the character cursor to a given position on the screen (see section 6.3)

MERGE "file program" 'merges' a program from cassette with the program currently stored in the computer; the name of the file program to be merged is specified within the quotation marks

MODE n changes screen mode; mode number
n=0 for large character, multi-colour mode,
n=1 for normal 40 column mode,
n=2 for high resolution 80 column mode
(see section 1.7)

MOVE x,y moves the graphics cursor to the graphics coordinates absolute position *x,y* (see section 6.3)

MOVER xr,yr moves the graphics cursor to the position *xr,yr relative* to the currect graphics cursor position (see section 6.3)

NEW is a direct command which clears the currently stored program and variables from the computer's memory, and allows us to start 'a-new'

NEXT is used as a delimiter at the end of a FOR . . . TO . . . NEXT loop (see section 4.3)

ON n GOSUB . . . statements cause program execution to jump to a subroutine according to the value of the select variable *n* (see section 4.7), return is made after completion of subroutine to line immediately following ON . . . GOSUB . . . statement

ON n GOTO has a similar action to ON . . . GOSUB, executing transfer to a given line number according to the value of the select variable *n* (see section 4.6)

OPENIN "filename" is used in conjunction with files on cassette; opens the file, whose name is specified within the quotes, ready for inputting data from the file to the computer (see section 7.6)

OPENOUT "filename" opens an output file on cassette for outputting data to; name of file so created is specified within quotes (see section 7.6)

ORIGIN X,Y changes position of graphics origin by *X* pixels horizontally and *Y* pixels vertically from 'normal' origin position at the bottom left-hand corner of the screen (see section 6.2)

ORIGIN X,Y , left , right , top , bottom defines new origin and graphics window, all positions defined in graphics pixels

PAPER paper number sets the colour on the screen background (see section 1.8 and fig. 1.9)

PEN pen number sets the colour of the screen characters, line drawings, etc. (see section 1.8 and fig. 1.9)

PI the CPC464 stores π to 9 significant figures as PI, i.e. PRINT PI returns 3.14159265

PLOT x,y plots the point (x,y) with reference to graphics coordinate system (see section 6.2)

PLOTR xr,yr plots the point relative to the current graphics cursor position (see section 6.2)

PRINT. . . is the BASIC output command ; Amstrad BASIC allows the use of ? as shorthand for PRINT

PRINT 52.6*4.9 outputs the results of calculations as a display on the screen (see sections 1.3 and 1.6)

PRINT ". . .string" displays the characters within the quotation marks (i.e. the string) on the screen (see section 1.5)

PRINT #8,. . . directs output to printer (see section 2.6)

RAD sets the radian angular measure mode (see section 5.5)

READ x fetches data from DATA statements and assigns it to variable *x* (see section 3.3)

REM REMark statements are used to include comments in the program; such statements are skipped over in program execution. N.B. A single quote character ' in a line which is not part of a string expression is equivalent to :REM

RENUM automatically renumbers program line statements starting at 10 in increments of 10

RENUM 50,20 renumbers line statements starting at 50 with increments of 20 (see section 2.3)

RESTORE RESTOREs the position of the READ pointer to the first data item in the first DATA statement (see section 3.5)

RESTORE line number RESTOREs pointer to beginning of the DATA statement specified by the line statement number (see section 3.5)

RETURN is used in conjunction with GOSUB statements to RETURN program execution, after subroutine has been completed, to line statement immediately following the GOSUB call (see section 4.7)

RUN is a direct command to start execution of the current program from the beginning (see section 2.3)

RUN line number executes current program starting at the specified line number

SAVE "program name" SAVEs the program currently in the computer on cassette; the program will be saved under the name specified within the quotation marks (see section 2.8)

STOP STOPs program execution at a particular line; execution may be resumed using the CONTinue command (see section 3.4)

TAG is used in conjunction with graphics keywords MOVE and MOVER to attach or 'TAG' a character string at a specific place in the graphics coordinate system (see section 6.3)

TAGOFF cancels the TAG operation; subsequent text is then sent to the previous character (screen) cursor position at the point at which TAG was invoked

WHILE ... WEND loop is used to repeatedly execute a group of statements WHILE some test condition is satisfied (see section 4.4)

WINDOW #n, left column no., right column no., top line no., bottom line no. creates a text window on the screen (see section 6.5)

APPENDIX II

SUMMARY OF STANDARD FUNCTIONS AVAILABLE ON THE AMSTRAD CPC464

Function	*BASIC symbol*	*Examples*	*Action*
Absolute value	ABS(x)	ABS (−9.8)=9.8 ABS (99.7)=99.7	Returns, i.e. determines, the absolute value or magnitude of *x*
ASCII code	ASC("X")	ASC("a")=97 ASC("Word")=87	Returns ASCII code number of the first string character
Arctan	ATN(x)	ATN(1)=π/4 radians =45°	Returns arctan or $\tan^{-1}$ in radians specifying value between −π/2 and +π/2; if DEG command used, returns $\tan^{-1}$ in degrees between −90° and +90°
Character function	CHR$(N)	CHR$(97)=a CHR$(119)=w	Converts Amstrad code number *N* to its equivalent character
Cosine	COS(x)	COS(PI/4)=0.7071	Returns cosine of *x* where *x* is in radians; if the DEG command has been invoked then *x* is specified in degrees
Exponential	EXP(x)	EXP(2.1)=8.1662	Returns e^x where e=2.7182818
Hexadecimal conversion	HEX$(N)	HEX$(65532)=FFFC	Converts the decimal number *N* to hexadecimal (base 16) number

Function	*BASIC symbols*	*Examples*	*Action*
Integer	INT(x)	INT(42.7)=42 INT(−1.4)=−2	Returns the value of x rounded to the nearest *smallest* integer
String slicing from LEFT	LEFT$(A$,N)	LEFT$("Amstrad",3) →"Ams"	Produces the first N characters of a string starting from LEFT
String LENgth	LEN(A$)	LEN("ABC34")=5	Determines the LENgth of the string (i.e. number of characters in string)
LOG (base e)	LOG(x)	LOG(2.0)=0.6931	Returns the natural logarithm (log to base e) of x, $x > 0$
LOG to base 10	LOG10(x)	LOG10(3.0)=0.4771	Returns logarithm to base 10 (common log) of x , $x>0$
LOWER case conversion	LOWER$(A$)	LOWER$("BRAIN") →brain	Converts a string containing one or more upper case characters to the same string but all lower case characters
MAXimum	MAX (no. list)	MAX(3,1,5,0.6)=5	Determines the MAXimum value in the list of numbers or numerical expressions
String slicing from MIDdle	MID$(A$,M,N)	MID$("abcdefgh",4,2) →de	Produces a 'sub' string of N characters starting from the Mth character from LEFT
MINimum	MIN (no. list)	MIN(3,1,5,0.6)=0.6	Determines the MINimum value in list
(PI) π	PI	PRINT PI yields π as 3.14159265	
Random numbers	RND(N)	RND(47) generates random number in range 0.000 . . . to 0.999999999	
String slicing from RIGHT	RIGHT$(A$,N)	RIGHT$("ABCDEFG",3) →EFG	Produces sub-string of last N characters, i.e. N RIGHTmost characters
ROUNDing to a no. of decimal places	ROUND(x,N)	ROUND(9.6745,2)=9.67 ROUND(9.6745,1)=9.7	Rounds number or expression x to N decimal places
SiGN	SGN(x)	SGN(4.2)=1 SGN(0)=0 SGN(−47)=−1	Determines the sign of x; returns −1 if x negative, 0 if $x=0$ and 1 if x is positive

Function	*BASIC symbols*	*Examples*	*Action*
Sine	SIN(x)	SIN(1)=0.84147098 DEG:PRINT SIN(45) →0.7071068	Calculates sine of *x* where *x* is in radians; if DEG command has been invoked then *x* is specified in degrees
SPACE	SPACE$(N)	SPACE$(10) creates 10 spaces	Creates *N* spaces
SQuaRE root	SQR(x)	SQR(2.0)=1.41413562	Determines square root of number or expression *x*
String	STR$(x)	STR$(45.12) →"45.12"	Converts number or numerical expression *x* to a decimal number string
TABulate	TAB(N)	PRINT TAB(8) "8 spaces in" ←8 spaces→8 spaces in	Tabulate function which will cause *N* character spaces from margin to be left before 'printing' a string or numerical result
Tangent	TAN(x)	TAN(PI/4)=1 DEG: PRINT TAN(45) →1	Calculates tan *x* where *x* is in radians; range limited to −200,000 to +200,000; if DEG command has been invoked then *x* must be specified in degrees
Upper case conversion	UPPER$(A$)	UPPER$("Richard") →RICHARD	Converts all characters in a string to upper case characters
VALue	VAL (A$)	VAL("987.3") →987.3 (numeric value)	Converts a 'number' string to the actual number

INDEX

ABS(X) function 63, 147
AND operator 54–6
Arithmetic keys 4, 5
Arrays 102–4
  program examples 105–9
Arrow keys 7, 10–11, 29, 63
ASCII codes 76–7
ASC(X) function 76–7, 147
Assignment statements 27–8
ATN(x) function 66, 147
AUTO command 21, 24, 142

BASIC keywords 21–3, 142–6
Bill-checking program 124–5
Boolean expressions 54–6
BORDER ink colours 15, 16, 142
Box drawing 89, 90
Brackets 11, 12–13
BREAK in program execution 4, 22, 39, 42, 92
Bubble sort 109–11, 112

Calculator use 4–7
CAPS LOCK key 3
CAT command 142
Character sets 76–7
CHR$(n) function 76, 147
CINT(X) function 63
CLEAR command 66
Clearing of screen 7
CLOSEOUT/CLOSEIN statements 114, 142
CLR key 3, 4, 7
CLS command 22, 23, 24, 142
Colon (:) 23
Colours of screen display 15–17
Comma (,) 8, 9
Comparison operators 45, 55, 56
Compound interest programs 50–1, 73, 126–7
CONT command 39–40, 142
COPY key 30
Correcting of mistakes 7–8
COS(x) function 66, 69–70, 147
Currency conversion program 121–2
Cylinder drawing 89–90

DATA statements 37, 38, 40–1, 98, 142
Decision-making statements 43–6, 54–6
DEF FN command 72, 142
DEG command 66, 142
DEL key 3, 4, 7
DELETE commands 22, 142-3
DIM statements 102–4, 143
DRAW command 70, 81–3, 143
DRAWR command 82–3, 143

EDIT command 29–30, 143
Ellipse drawing 89, 90
END statement 46, 143
ENTER key 2, 3, 4, 19
EOF function 116, 143
ERASE 143
ESC key 3, 4, 21
Exponential functions 71
EXP(X) function 71, 147

Files 113–17
FIX(X) function 63, 64
Flowcharts 44, 47, 51
FOR. . .TO. . .NEXT loops 46–51, 143
Fourier's theorem 93
FRE command 143
Full stop (.) 6
Function keys 2–4
Functions defined by user 72–3

GOSUB. . .RETURN statements 59–60, 143
GOTO commands 56–9, 143
Graph plotting 98–100
Graphics system 79, 80

Hard copy 29
HEX$(N) function 147
High-resolution mode 13, 14, 96

Identifiers 26–8
IF. . .THEN. . .ELSE statements 43–6, 143
Income tax program 125–6
INK colours 15–16, 17, 143
INPUT statements 34–6, 98–9, 143
Integer variables 27
INT(X) function 63, 64, 148

Jump instructions 56–60

Keyboard 2–4
Keys, *see* Arithmetic keys; Arrow keys; CLR key; Copy key; DEL key

LEFT$ function 73–4, 148
LEN function 75, 148
LET statement 22, 143
Letter program 123–4
Line numbering 19, 20, 24
Line statements 19, 20, 24
LIST command 21, 24, 144
LOAD command 31–2, 144
Loan repayments program 127–8
LOCATE command 84–6, 144
LOGIO(x) function 71, 148
Logarithmic functions 71
Logic operators 54–6
LOG(x) function 71, 148
Lower/upper case letters 4, 24, 148, 149

MAX function 148
MERGE command 148
MID$ function 74, 148
MIN function 148
Modes 13–14, 144
MOVE command 86, 89, 144
MOVER command 86, 144
Multiple-print program 122–3

NEW command 21, 24, 66
NOT operator 54–5
Numeric variables 26–7

ON. . .GOSUB statement 59–60, 144
ON. . .GOTO statement 58, 144
OPENOUT/OPENIN statements 114, 144
OR operator 54–6
ORIGIN command 70, 83–4, 144

PAPER colours 16–17, 145
Parabola plotting 91–2
Pen colours 16–17
Phone number program 116–17
PI 10, 68, 145, 148
Pixels 14, 79, 80
PLOT command 70, 79–81, 145
PLOTR command 79, 81, 145
Polygon drawing 87–8
PRINT '. . .' command 8, 9, 145
PRINT command 7, 23, 145
  ? as shorthand 7, 145
PRINT USING command 8, 9–10, 145
Printers 29
Program, definition of 18
Program editing 29–30
Program examples 23–5, 119–28
Programming, introduction to 18–20

RAD command 66, 67, 145
Random number generation 64–6
READ. . .DATA statements 34, 37–9, 40–1, 98, 145
Ready condition 2
REM statement 23, 24, 145
RENUM command 22, 145
Repetition statements 46–54
Reserved words 26, 142–9
RESET keys 3, 4
RESTORE statement 40–2, 145
RETURN statement 59, 145
RIGHT$ function 74, 148
RND(N) function 64–6, 148
ROUND (x,N) function 63, 148
RUN command 21, 24, 145

SAVE command 31, 145
Savings interest programs 50–1, 73, 126–7
Scaling of graphs 100
Screen display 13–17
Screen modes 14, 85
Semi-colon (;) 8, 9, 86
Sequential files 114
Setting up 1–2
SGN(X) function 64, 148
SHIFT key 3
Sinewave plotting 70, 93–4
SIN(x) function 66, 68–9, 149
Sorting
  alphabetic order 112–13
  numeric order 109–11
SPACE$(N) function 149
Spiral drawing 92–3
SQR(X) 11, 62–3, 149
Standard functions 62–71, 73–7, 147–9
Start up 2
STEP command 49
STOP statement 39–40, 145
STR$ function 75–6, 149
String slicing functions 73–4
String variables 27–8
Subroutines 59–60
Syntax errors 4, 20

TAB(N) function 8, 9, 149
TAG command 86, 146
TAGOFF command 86, 146
TAN(x) function 66, 149
Triangle drawing 86–7, 88
Trigonometric functions 66–70
True/false testing 44–5, 51–2, 54

Units conversion program 119–20
Upper/lower case letters 4, 24, 28, 148, 189
User-defined functions 72–3

VAL function 75–6, 149
Variables 26–8

Wave plotting 70, 91–4
WHILE. . .WEND loops 51–4, 146
WINDOW command 94–7, 146

XOR operator 54–6

PROGRAMMING THE AMSTRAD CPC464

Make the most of your Amstrad with this practical guide to using and programming the CPC464. You don't need any computing experience to follow the text, which begins with the basics of the Amstrad and its operation and extends to graphics, colour and easy but powerful programming techniques — giving both newcomers and more experienced users enough knowledge and understanding of their machine to write their own programs.
Throughout the book new concepts are explained simply with the help of practical program examples. There are plenty of clear program listings, appendixes of BASIC keywords and standard functions — plus eight original Amstrad programs for your home and business use!

ISBN 0-304-31250-9

9 780304 312504

Meadows PROGRAMMING THE AMSTRAD CPC464 Cassell