AMSTRAD
BASIC

a tutorial guide

AMSTRAD BASIC

a tutorial guide

Part 1
FIRST STEPS

First edition 1984

Amstrad Consumer Electronics plc
Brentwood House
169 Kings Road
Brentwood
Essex

Amstrad BASIC

A Tutorial Guide
Part 1: First Steps

SOFT 111 ISBN 1 85084 000 8

- Programming by Dave Atherton
- Written by Dave Collier and George Tappenden
- Production by Peter Hill and Ray Smith
- Printed in England by Horwood Printers

CONTENTS

PREFACE

This is Part 1 of a self-study course on programming in BASIC using the Amstrad CPC 464 Colour Personal Computer. The two datacassettes that accompany this written text contain computer programs which are an integral part of the course.

Datacassette A contains:

- Programs to help explain the principles of simple and entertaining programming
- Games for your amusement and to help you get used to using a computer

Datacassette B contains:

- Self-assessment tests to make sure you have understood the concepts described in each chapter

Further and more advanced programming principles are covered in Part 2 of this course, *More BASIC*.

1 WHAT IT'S ALL ABOUT

If you are reading these words you are almost certainly the proud owner of an Amstrad CPC 464 Microcomputer. Its superb display and excellent sound quality will have already opened up an exciting new world of fun and excitement for you. Also you will have realised that to do more than run standard games and programs, you need to learn the CPC 464's language – BASIC.

BASIC

BASIC is the world's most popular computer language. It is also the ideal language for the beginner since it is possible to write your first program after only a few hours of study. The satisfaction to be gained from this is enormous for, make no mistake, programming is fun. It can even become a hobby in itself.

So, what is programming? Well, you have to remember that a computer can do many things but it can't think. You have to do its thinking for it. This thinking – that is, working out what needs to be done to achieve an end – is in the form of a series of instructions known as a program. Fed into a computer, a program can make it become, for example, an arcade game, or a word processor, or a machine that looks after your accounts.

You will probably find that programs written specially for the CPC 464 will not work unchanged on other computers. This is because the Amstrad BASIC used by the CPC 464 contains many unique commands and functions not available on less sophisticated equipment.

BASIC has its own vocabulary, the same as any other language. This vocabulary is made up of 'keywords' and you are about to learn what these keywords are, and what they mean to the CPC 464. Each time a new keyword is described in this manual it is printed in the outside margin so that you can easily flip back through the book to refresh your memory on individual keywords.

How to use this book

Each chapter of this book represents about one or two evenings' work. Typically it will contain:

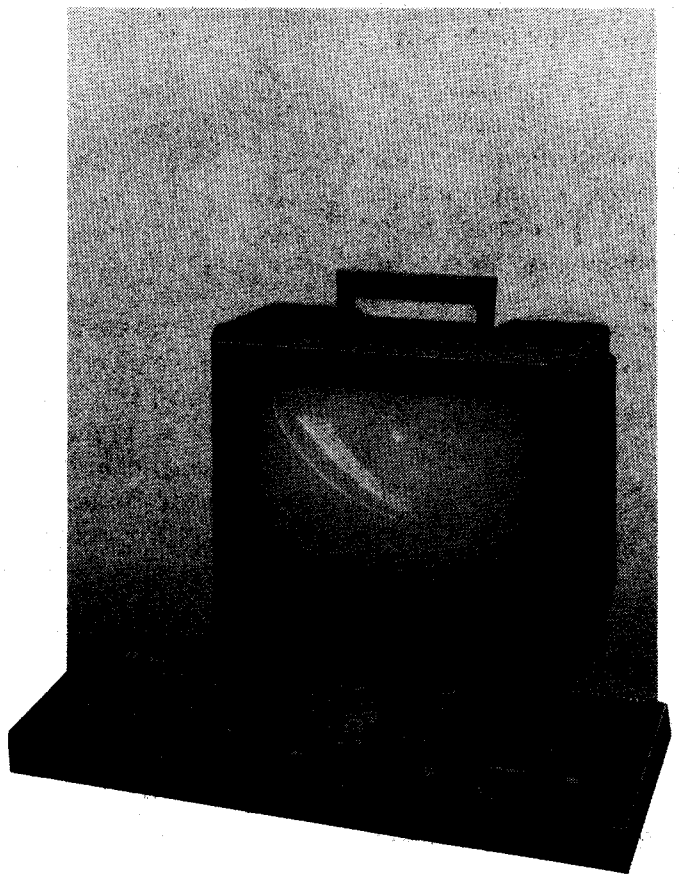
- Written explanation
- Practical work on the computer
- Examples for you to program yourself

There are exercises to reinforce what you have learned, and there is a programmed self-assessment test to go with each chapter.

Don't skip chapters. New information is introduced progressively through the book and is built on to the knowledge obtained from previous chapters. If you think a chapter or a section of a chapter looks a bit complicated, just read it quickly once or twice and then work through it slowly. Make sure that you understand by means of the self-assessment tests.

After completing this part of the course you should be able to write simple, reliable programs for your own purposes. Part 2 of this course, *More BASIC*, explains the more advanced features of Amstrad BASIC and will teach you how to write rather more complicated programs.

2 SETTING UP AND GETTING DOWN TO IT



Firstly you have to unpack your CPC 464. If you have already done this you can skip the next few paragraphs.

When you open the boxes they should contain the following items:

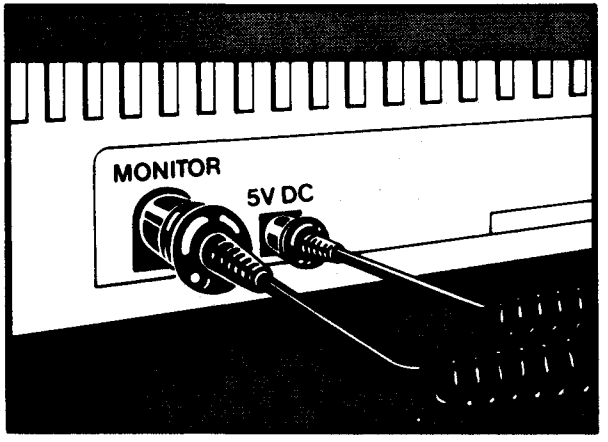
- CPC 464 Colour Personal Computer
- GT 64 Monitor, CTM 640 Colour Monitor
- MP1 Modulator/Power supply (optional)
- *Amstrad CPC 464 User Guide*
- Demonstration cassette

Find a reasonably large desk or table in a quiet part of the house and place the CPC 464 and its monitor (or MP1 Modulator/Power Supply and domestic TV receiver) on it. Carefully insert the two leads from the front of the monitor or MP1 into their sockets in the back of the CPC 464 (see diagram). Do not plug the monitor or MP1 into the 13-amp mains yet. Make sure that the plugs are fully in place in the back of the CPC 464 but do not use too much force. Plugs and sockets can be damaged by constant plugging in and out so,

if possible, try to find a permanent home for your CPC 464 – at least for the duration of this course.

If you are using a domestic TV receiver with your CPC 464, make sure that you have two 13-amp sockets available, or use a double adaptor so that you can plug in the TV and the MP1 at the same time. Then connect the co-axial lead from the MP1 into the aerial socket of the TV and tune the TV to channel 36.

Find yourself a nice comfortable chair to sit on and then arrange the monitor or TV so that it is 1 metre (about three feet) from your nose – working with your face too close to a screen can be very tiring and may cause eye strain.



Place this book in such a position that you can easily read it while using the keyboard and still be able to watch the screen. It is also a good idea to put the *Amstrad CPC 464 User Guide* somewhere within easy reach.

Then plug into the mains and switch on. You should get the following words on the screen:

```
Amstrad 64K Microcomputer  (v1)

© 1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.ϕ

Ready
■
```

If you don't see this (yellow characters on a blue background if you have a CTM 640 Colour Monitor or a colour TV) turn off and try again or, if you are using a TV, check that it is properly tuned to channel 36. If you still have difficulty, check if the ON indicator of

the CPC 464 is showing red. If not, verify that:

- The CPC 464's ON/OFF switch is ON
- There is no general power failure
- The power lead is firmly plugged into the CPC 464
- The fuse in the 13-amp plug is intact

If all these tests fail, contact your dealer for further advice.

RUN

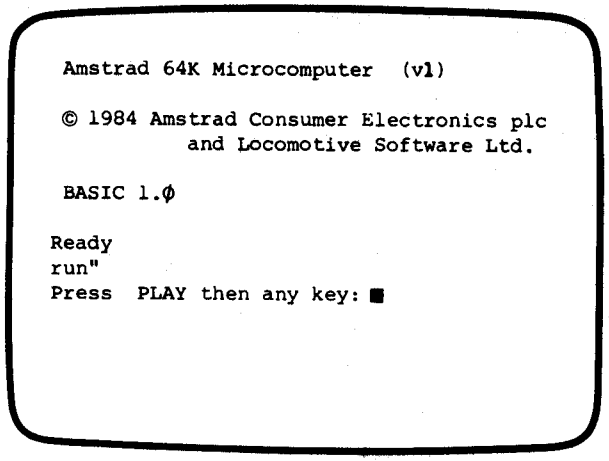
HELLO

Once you have the 'welcome' on your screen, we can start getting down to it. 'Ready' means that the computer is ready for you to enter commands or a program. The square blob is known as the 'cursor', and shows you where the next thing you type on the keyboard will be placed.

So away we go. Put the cassette, *First steps in BASIC – Datacassette A*, into the datacorder and type the following on the keyboard:

`run"` ENTER

To type " you have to hold down one of the keys marked SHIFT, on either side of the lower row of letters of the keyboard, while you press the key marked " (it is over the '2' on the left-hand side of the top row). Pressing the ENTER key signals to the CPC 464 that you have finished typing and that you expect it to do something. If you have typed the command correctly the CPC 464 will reply by adding another line to the message on the screen:



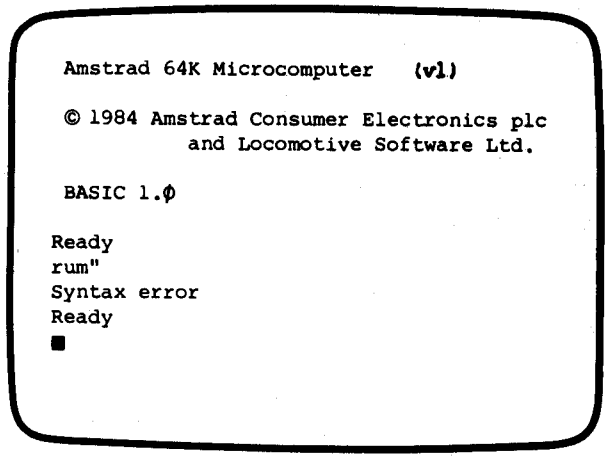
Make sure that the tape is at the beginning by pressing REW on the datacorder, and then follow the instructions. Push down PLAY and then press the ENTER key. You will hear a high-pitched sound from the built-in loudspeaker as the first program is read into the CPC 464.

If you make a mistake while typing in (and before you press ENTER) use the

DEL

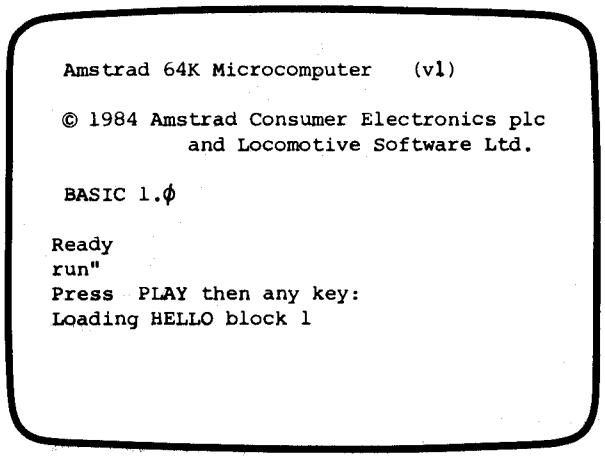
key. This is the DELete key, which backspaces and deletes the last character you typed. You can then retype the letter you got wrong.

If you have made a mistake and then pressed ENTER, don't worry. The CPC 464 will merely put another line on the screen as shown in the following example:



If you get this message you will just have to start all over again and retype the line.

All being well (and if you have typed in the command correctly), the following message will appear briefly on the screen:



This message will be followed almost immediately by the HELLO program, which will give you a friendly welcome to the world of computers.

When you have seen the HELLO program several times through, look at the top left corner of the keyboard and you will see a red key. This is the ESCape key. Press it twice:

ESC ESC

The CPC 464 will reply with a message such as the following:

```
Break in 14ϕ
Ready
■
```

The number need not be 140. And it doesn't mean that either you or the CPC 464 should take time off for the next 140 seconds, minutes, hours, or days. Nor are you expected to smash your way into house number 140 in your street. Whatever the number is, ignore it. We'll learn about line numbers later on. In any case, you will have stopped the HELLO program from running and are now ready to tackle the rest of this chapter.

Incidentally, note that the CPC 464 insists that there is a difference between 0 (nothing or zero) and O (oh, as in 'hello') by putting a diagonal line through it when it is *not* the alphabetic character.

Game number 1

Let's play a game. Computer games may not strike everyone as the best way to spend their time and money, but there is a lot more to them than meets the eye. Firstly you soon become familiar with the machine without the tedium of formal exercises, and secondly games can make you realise that computers are fun.

We'll do something different this time. Type into the CPC 464 the following line:

```
load"simon"
```
ENTER

Don't forget to hold down the SHIFT key to type the quotation marks ("). The CPC 464 will answer with the following message:

```
Press PLAY then any key:
```

Follow the instructions again. Push down PLAY on the datacorder and then press the ENTER key on the main keyboard. If you typed the name in correctly and all is well, the CPC 464 will start running the cassette in the datacorder and loading a program called SIMON. Programs always have names so that you can find them when you want them. This time the program will only be fetched into the CPC 464's memory and it will do nothing until you tell it to.

The CPC 464 will give the message:

```
Loading SIMON block 1
```

changing to:

```
Loading SIMON block 2
```

Don't worry about what this means. After the cassette has stopped in the datacorder, a further message will be given:

```
Ready
```

This is when you have to instruct the CPC 464 what to do with the program it now has in memory. You type:

```
run
```
ENTER

Have fun.

LOAD

Testing

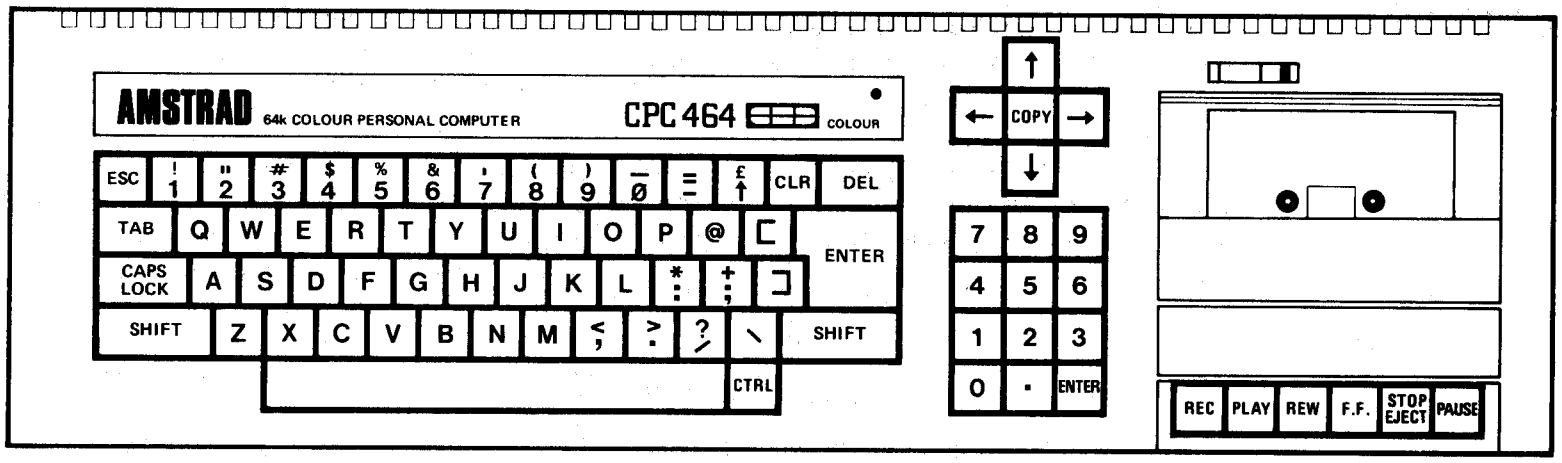
When you are tired of playing SIMON, stop it by pressing the ESCape key twice as explained above. Your first test is to try the same procedure that you used for SIMON to load the first of our Self-assessment Tests, SAT2, from Datacassette B. When you run this program it will ask you questions about this chapter so you can see if you need to go back over anything.

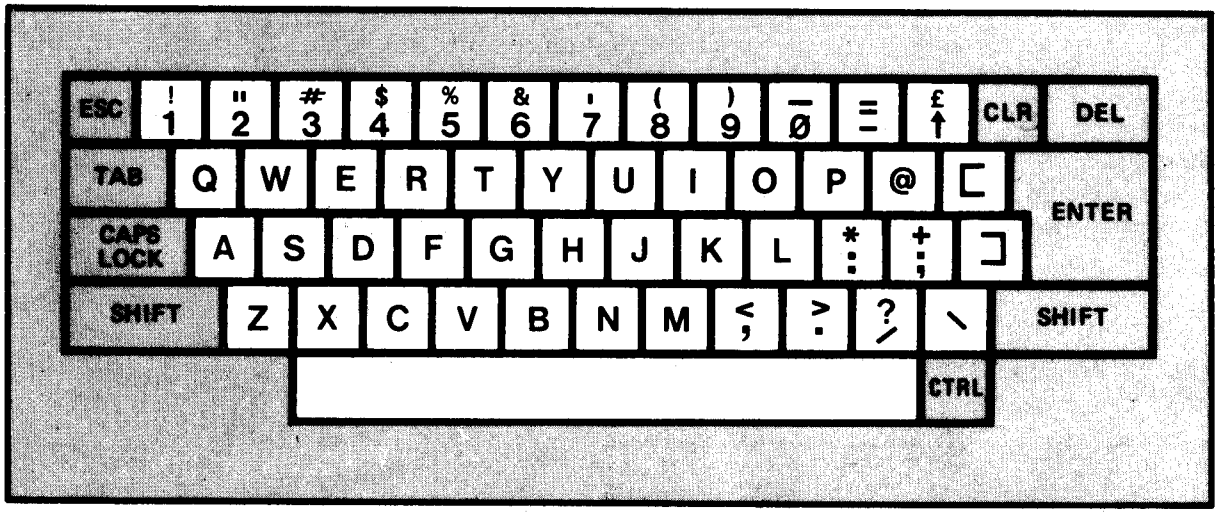
3 USING THE KEYBOARD

We said earlier that you have to learn the BASIC language to communicate with the CPC 464. As you will have realised, you tell the CPC 464 what to do by typing words and numbers on the keyboard. This chapter is all about just that – getting to know the positions of keys and learning how to press the right ones at the right time.

The CPC 464 has five separate groups of keys:

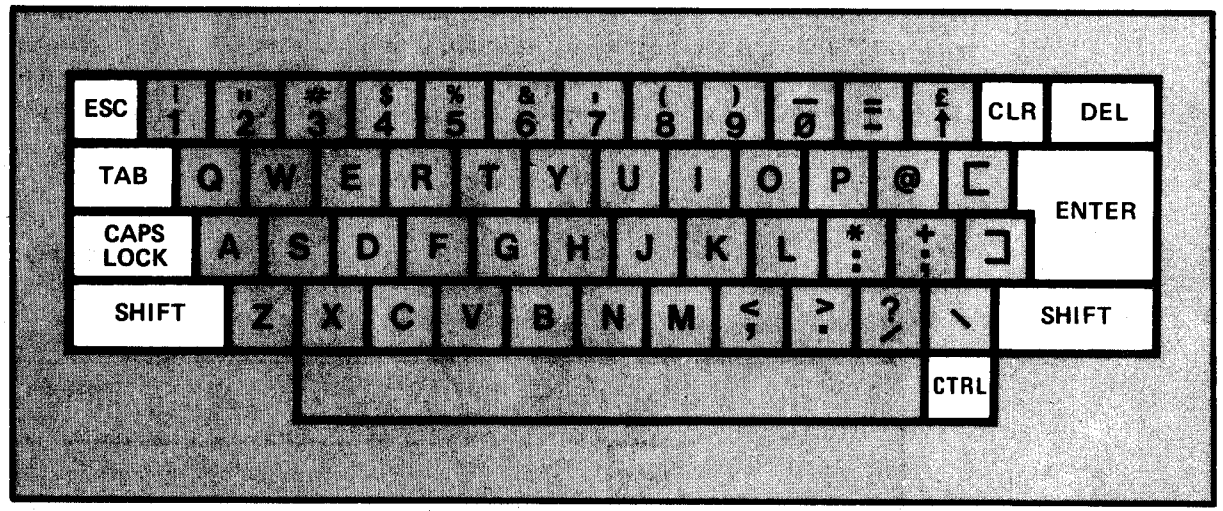
- Character keys
- Control keys
- Numeric keypad
- Cursor keys
- Datacorder controls





Main keyboard: character keys

You have already been using these. If you have ever used a typewriter you will need no further explanation. This part of the keyboard comprises letters, numbers, a lot of punctuation marks known as 'special characters', and a 'space' bar for putting in blanks. If you hold any of these keys down for more than about half a second, it will repeat as though you had pressed it again and go on repeating until you take your finger off.



Main keyboard: control keys

ESC

ESC stands for ESCape. If you press it while a program is running, it will stop the CPC 464 in its tracks, but you may restart the program by pressing any other key on the keyboard. Pressing ESC a second time will bring the CPC 464 back to the READY condition.

TAB

If you press the TAB key you will see a right-facing arrow on the screen. This key will not be used in this part of the course, and its use will be fully explained in Part 2.

SHIFT

SHIFT changes the symbols produced when you press character keys. Holding down SHIFT will give you the capital letters on the letter keys and the upper symbols on the number and special character keys.

CAPS LOCK

Pressing CAPS LOCK gives you capitals on the letter keys until you press it again. It has the same effect as holding your finger on the SHIFT key except that you still get the lower symbols on the number and special character keys.

CLR

CLR (for CLeaR) is rather similar to the DEL key. It deletes a character, but not the one to the left of the cursor as the DEL key does; it 'eats' the character under the cursor, without moving position, and everything to the right of the cursor moves up one place to the left.

DEL

You may remember the DEL key from the previous chapter. When you are typing in lines, pressing the DEL key DELetes the character to the left of the cursor, and then backspaces the cursor to take its place.

ENTER

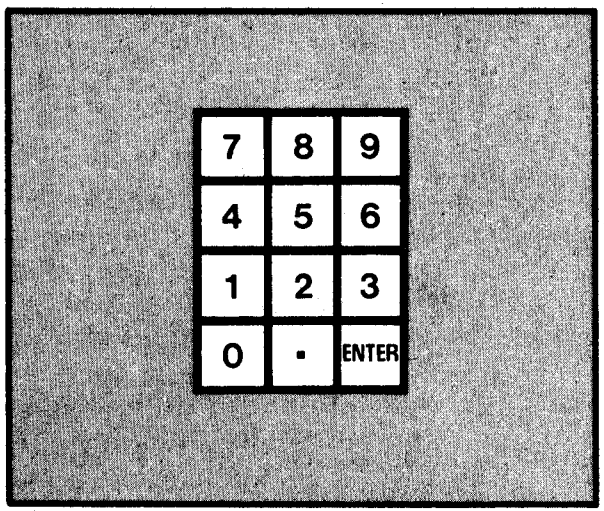
The ENTER key is something like the carriage return key on an electric typewriter. You have to press it at the end of every line you type into the CPC 464 to let it know that you have finished. Normally, the cursor will be taken from the end of the last word or number typed and put at the beginning of the next line down. Sometimes the CPC 464 will also say:

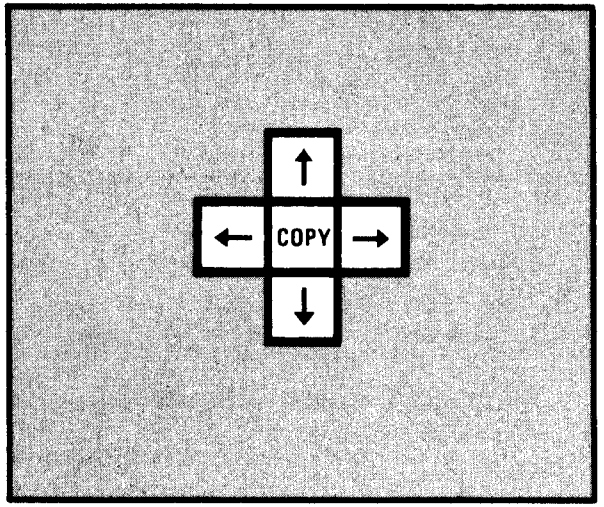
```
Syntax error
```

This is BASIC for 'I don't understand' and is known as an 'error message'. We will see more of these later in the course.

CTRL

This stands for ConTRoL. Holding down CTRL will give you yet another set of symbols on the letter keys and some of the number keys, in addition to those inscribed on the key tops. When used with other control keys, this key also instructs the CPC 464 to do certain things – we will see what they are later on.



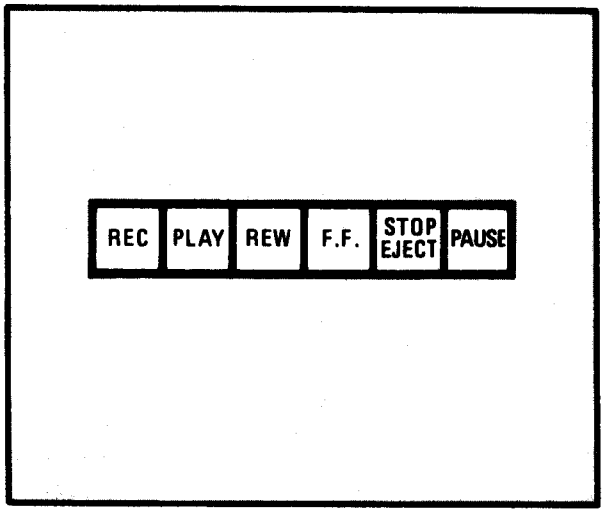


Numeric keypad

These keys are arranged conveniently for typing in lots of numbers. Apart from the extra ENTER key they are identical to the number keys on the top row of the main keyboard except that, unless they are specially programmed, they are not affected by SHIFT or CTRL. This special programming will not be described in this part of the course, but later in this chapter you will see how this ENTER key can have an extra function.

Cursor

The 'arrow' keys are used to move the cursor around the screen in the appropriate directions. The COPY key will be described later on in this course.



Datacorder controls

There is only one difference between these keys and the controls on an audio cassette recorder. After the PLAY (or PLAY and REC) keys have been depressed, the datacorder will not operate until it has been instructed to by the CPC 464.

Practical work

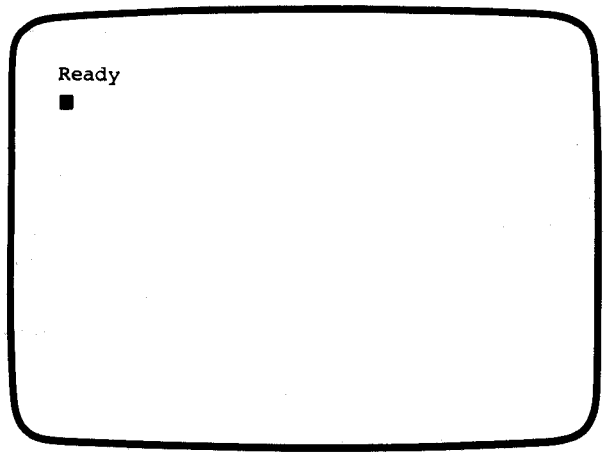
Check that the cassette is properly inserted in the datacorder. If you still have the remains of SAT2 all over the screen from your work on the previous chapter, you can get rid of it by typing the following line:

CLS

```
cls
```
ENTER

This means CLear Screen and, as you will see, it does just that. Try it. No matter what you were doing before, the result will be as follows:

RUN



Magical, isn't it? Now we can run the next program. Type the following line:

```
run "letters"
```
ENTER

Now you can just press keys on the keyboard and see what happens. Try pressing more than one key at a time.

While using this program you can see many of the characters that can be shown on the screen but are not marked on the keytops. This is done by holding down the CTRL key at the same time as you press the letter and number keys, although not all the keys have 'hidden' characters.

Once you have started running a computer program it will go on running until it reaches the end. Or, if it is designed to repeat itself continually, like LETTERS, you have to stop it. You saw above that pressing the ESC key twice will do the trick. Another way of doing it is to 'force a restart'. This has the same effect as turning the power off for a few seconds and then turning it on again. This is how you do it. Hold down

CTRL and SHIFT

and press

ESC

As you will see, the CPC 464 goes back to the 'welcome' screen you saw when you turned power on. From now on we will refer to this as CTRL/SHIFT/ESC.

Warning When you do this the CPC will 'forget' any program it had in memory.

All right so far? The next program is a very simple one but we shall now learn a novel way to RUN it. Hold down

CTRL

Press

ENTER

on the numeric keypad (not the big ENTER key this time).

The CPC 464 will reply exactly as if you had typed in a whole line of instruction. Do you recognise it? You will remember that we said earlier that this ENTER key had an additional function!

REPEAT NAME (the program you just loaded) won't keep you amused for very long so use CTRL/SHIFT/ESC to reset the CPC 464, and then load and run the next program: KEYBOARD.

KEYBOARD is a training program to get you used to the keys on the CPC 464. Once you have spent a little time on it you will begin to remember where things are. It is probably worth coming back to this program from time to time to improve your typing speed.

Game number 2

You must have played this one as a paper game. It's called HANGMAN. You now know several ways to load and run programs but, just to make sure, here is one of them again:

```
run "hangman"
```
ENTER

Testing

Before going on to the next chapter, load and run SAT3.

4 PUTTING THINGS IN THEIR PLACE

You will have seen from the demonstration cassette and the previous programs in this course that the CPC 464 has superb graphics. In this chapter we are going to make a start on what you need to know to draw the appropriate lines and shapes on the screen for your own designs.

The CPC 464 displays all characters and graphics in a 'window' on the monitor screen. The sides, top and bottom are known as the border, and are never used. You can change the colour of this border though. Enter the following:

BORDER

```
border Ø
```
ENTER

Black, isn't it? Now try:

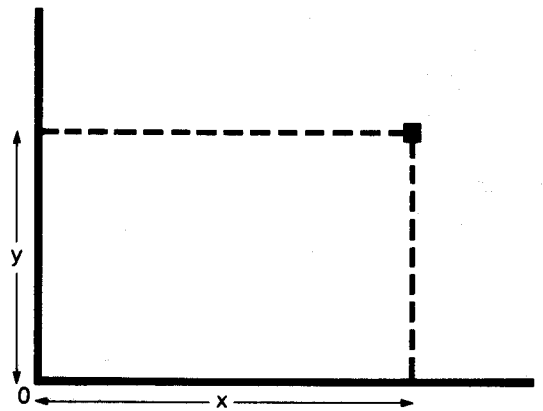
```
border 26
```
ENTER

The opposite extreme. Now you can amuse yourself by trying all the numbers in between.

If you have already read the *User Guide*, you will know that the CPC 464 has a range of 27 colours to choose from. Even if you don't have a CTM 640 Colour Monitor, or a colour TV, going through the numbers in this way will not be a waste of time since the ascending number order corresponds to the equivalent grey scale in black and white.

Coordinates

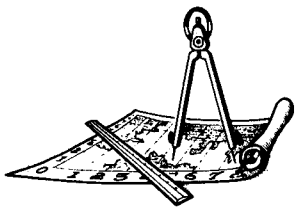
Every line, shape, or character you see on the screen is made up of a number of tiny dots. The position of each tiny dot is described by its 'x' and 'y' coordinates.



The horizontal position is given by 'x' and the vertical position by 'y'.

Run the next program. It's called DRAW. You can draw straight lines on the screen by holding down the TAB key at the same time as you press one of the cursor keys. If you want to move position without drawing, you just press the cursor keys. You can speed things up by holding down a SHIFT key at the same time.

As you can see, the program gives you the 'x' and 'y' coordinates for the current position of the cursor. If you keep the cursor going upwards it will eventually disappear from the window. Note the value of the 'y' coordinate where this happens. Do the same thing for the 'x' coordinate. You will now have discovered that the CPC 464 has a graphics screen 400 points high by 640 points wide.



Try drawing a square in a particular position on the screen. The following is an example:

Put a square 50 × 50 on the screen with its bottom left-hand corner at x = 300, y = 150.

First, move the cursor in the 'x' direction until it reads 300. Then move the cursor in the 'y' direction until it reads 150. If the square is 50 × 50, we have to draw a line from this point to x = 300 + 50 = 350. Draw this line. If you go too far you can 'eat' the line by making the cursor go in the other direction until the 'x' count is correct. The count will now be:

```
x=35Ø  y=15Ø
```

Now draw a vertical line until y = 150 + 50 = 200. Then draw to x = 350 - 50 = 300. And finally y = 200 - 50 = 150, and we are back where we started.

Don't worry if it doesn't come out exactly right - it's the principle that counts.

A la mode

Before we continue with coordinates, let's explore another thing about the screen display.

When you first switch the CPC 464 on, the 'welcome' comes up in characters about twice the size of those on this page. You will have noticed, however, that some of the programs use larger characters. There are, in fact, three sizes of character and they are selected by the keyword MODE.

MODE

Try it out. Get the 'welcome' back on your screen by forcing a restart (CTRL/SHIFT/ESC – remember?), and then type in:

```
mode ø
```
ENTER

You can see that the characters are now twice as wide as they were before. Now enter:

```
mode 1
```
ENTER

We are now back to the size of characters we started with. Now try:

```
mode 2
```
ENTER

And we have characters half the previous width. So there are three modes, namely:

- Mode 0 – 20 characters per line
- Mode 1 – 40 characters per line
- Mode 2 – 80 characters per line

Note that only the width changes and that there are still 25 lines of characters possible on the screen at any one time.

The keyword MODE also affects the screen graphics. But in the case of graphics we don't talk about characters but 'pixels'.

A pixel is the smallest size of dot you can draw on the screen. We saw that the graphics screen is 640 points by 400 points, but the width of the pixel is different for each mode:

- Mode 0 – 4 points wide
- Mode 1 – 2 points wide
- Mode 2 – 1 point wide

Pixel height is always two points and does not vary with the mode.

In position again

So, back to coordinates. Later on we will see how to position text and numbers on the screen, but for the moment we will concentrate on graphics. The next program is called COORGEOM, but before we load it here is a very useful keyword, CAT.

CAT is short for catalogue, and is used to find what programs are on a datacassette. Wind the datacassette back to the beginning by pressing the REW key and then enter:

cat **ENTER**

The CPC 464 will respond by giving the message:

```
Press PLAY and then any key ■
```

It is just as if you had given a LOAD or RUN command except that, instead of loading a program, the CPC 464 will put a 'found' message on the screen such as:

```
DRAW    block 1 $
```

Programs are always stored on cassette in blocks of 2,000 characters. Long programs may comprise many blocks stored individually. The CPC 464 not only puts a message on the screen for each block in turn, but also checks that there are no recording errors and will then put 'OK' at the end of the line.

If you ask the CPC 464 to load a program by name, you will also get 'found' messages for the other programs before it on the tape, but no checking is carried out.

Anyway, by now you should have found COORGEOM, so load up and away you go.

Game number 3 - BOMBER

This is the one you've been waiting for! The chance to zap an extraterrestrial spacecraft. Feed the coordinates into your robot bomber and deliver the plutonium bomb right on target – or perish!

Testing

If you survived that gruelling battle, check your progress by running SAT4 before going on to the next chapter.

5 DRAWING A PICTURE

DRAW

PLOT

The PLOT keyword is going to be the first BASIC 'Command' we are going to look at in detail. Up to now you have been entering things like RUN and BORDER without realising that they are commands or that they have to follow a precise set of rules.

To create commands you often have to add 'arguments' to the keyword to provide the details of the operation to be performed. In the case of PLOT the arguments give the desired position on the screen as specified by the x, y coordinates. For example, enter the following:

```
plot 319,199
```
ENTER

You will now have a yellow dot of 1 pixel almost exactly in the middle of the screen.

If you have trouble remembering which is x and which is y, don't forget that you have to go in through the door of a house before you can climb the stairs, i.e. left-right before up-down. After the PLOT command has been executed, the CPC 464 leaves the graphics cursor at those x, y coordinates until told otherwise.

Here we go with another graphics command:

```
draw Ø,Ø
```
ENTER

The DRAW command has an identical structure to PLOT, except that the arguments give the point to which the line must be drawn – specified, of course, by the x, y coordinates. The diagonal line you will now have on the screen starts at the centre (x = 319, y = 199) and goes down to the bottom left-hand corner (x = 0, y = 0). Now try drawing the same square that we tried in the previous chapter. Here are the commands:

```
move 3ØØ,15Ø
```
ENTER

```
draw 35Ø,15Ø
```
ENTER

```
draw 35Ø,2ØØ
```
ENTER

```
draw 3ØØ,2ØØ
```
ENTER

```
draw 3ØØ,15Ø
```
ENTER

The MOVE command puts the graphics cursor at the x, y coordinates specified in its arguments, without putting anything on the screen.

MOVE

A square program

Until now we have been entering commands for direct execution by the CPC 464. Now we shall learn about entering and storing a program for later execution. Before we do this we have to clean out the CPC 464 by entering the following:

NEW

new ENTER

This has the same effect on the CPC 464's memory as a wet cloth on a blackboard. Although the screen is not cleared, NEW erases any program that had previously been loaded or entered. Only use this command when you are sure that no harm will come of it!

So, here we go with our first stored program. It is the same square again, only this time we have added line numbers:

```
1Ø clg
2Ø move 3ØØ,15Ø
3Ø draw 35Ø,15Ø
4Ø draw 35Ø,2ØØ
5Ø draw 3ØØ,2ØØ
6Ø draw 3ØØ,15Ø
```

Line 10 is another new command for you. Whereas the CLS command clears the screen and puts the text cursor at the top left corner,

CLG

LIST

the CLG command also clears the screen but then puts the *graphics* cursor at the bottom left corner (x = 0, y = 0). These two commands may puzzle you a little since they are apparently very similar. In Part 2, however, you will learn to handle text and graphics on the screen at the same time and their usefulness will become apparent.

Enter the six lines of this program *exactly* as shown, and press ENTER at the end of each line. From now on in this course, you must remember to press the ENTER key at the end of each line.

These line numbers are necessary so that you can indicate to the CPC 464 in which order you want the commands to be stored. Unless told otherwise it will also execute the commands in this order. The line numbers go up in tens so that extra lines can be slotted in if necessary. It doesn't matter in which order you enter the lines. In fact, if you make a mistake in one of the lines and don't notice it until after pressing the ENTER key, you can replace the incorrect line in memory by simply re-entering the line.

When you are satisfied that all is well, RUN the program by entering:

```
run ENTER
```

Pretty, isn't it? The CPC 464 stored your program in memory and only executed it when you gave it the RUN command. And it's still there. You can look at it by entering:

```
list ENTER
```

Changing colour

The three graphics commands we have just learned have an optional extra argument, the INK. Try re-entering line 50 of the square program as follows:

```
5Ø draw 3ØØ,2ØØ,3
```

When you run the program this time, the left-hand side and top of the square will be drawn in red. This is because the '3' we added indicated that the DRAW command should be executed using INK number 3. The CPC 464 will then continue to use this INK until told to change. The range of different INKs that can be specified in DRAW commands depends on the mode being used.

The maximum number of different INKs which can be used for each mode are as follows:

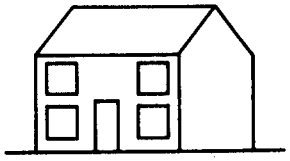
- Mode 0 - 16 INKs
- Mode 1 - 4 INKs
- Mode 2 - 2 INKs

Now try entering the following:

```
ink 3,Ø
```
ENTER

You will immediately see the red line change to black. Do the same thing as we did with BORDER in the previous chapter and try some other colours for INK number 3. You could also try changing INKs 0, 1 and 2 as well.

INK



REM

PAPER

Housing

The next program on Datacassette A is called HOUSE. It uses all the commands we have learned so far plus two more. The first one is REM for REMark. When the CPC 464 comes across REM, it ignores the rest of the line. This allows you to put comments and explanations into programs so that other people (or yourself, if you have forgotten after a period of time) can understand what the program is all about.

The other new command is PAPER, which enables you to specify the background colour of the screen window. The argument for PAPER must be the number of one of the INKs specified for the current mode. If you changed INK 0 when suggested above you will have seen that this was the one automatically selected for the PAPER when the CPC 464 was first switched on.

Before running the program, study the following program listing.

```
10 REM Drawing a house
20 MODE 0
30 CLS
40 REM ** start **
50 BORDER 12
60 INK 0,12:REM yellow
70 INK 1,3:REM red
80 INK 2,6:REM bright red
90 INK 3,9:REM green
100 PAPER 0
110 REM draw front
120 MOVE 100,50
130 DRAW 100,250,1
140 DRAW 400,250
150 DRAW 400,50
160 DRAW 100,50
170 REM draw side
180 MOVE 400,250
190 DRAW 600,250
200 DRAW 600,50
210 DRAW 400,50
220 DRAW 400,250
230 REM draw gable end
240 REM already at start point
250 REM so no need for a MOVE
260 DRAW 500,350
```

(Continued)

```
270 DRAW 600,250
280 DRAW 400,250
290 REM draw roof
300 REM only two lines needed
310 MOVE 100,250
320 DRAW 200,350
330 DRAW 500,350
340 REM draw door
350 MOVE 225,50
360 DRAW 225,140,2
370 DRAW 275,140
380 DRAW 275,50
390 REM draw windows
400 REM left hand bottom
410 MOVE 120,70
420 DRAW 120,130,3
430 DRAW 180,130
440 DRAW 180,70
450 DRAW 120,70
460 REM left hand top
470 MOVE 120,170
480 DRAW 120,230
490 DRAW 180,230
500 DRAW 180,170
510 DRAW 120,170
520 REM right hand top
```

(Continued)

```
530 MOVE 320,170
540 DRAW 320,230
550 DRAW 380,230
560 DRAW 380,170
570 DRAW 320,170
580 REM right hand bottom
590 MOVE 320,70
600 DRAW 320,130
610 DRAW 380,130
620 DRAW 380,70
630 DRAW 320,70
```

Testing

You may like to go over this chapter again before running SAT5. We have covered quite a lot of ground in a short time but are now getting down to real programming.

6 NUMBERS, LETTERS AND WORDS

PRINT

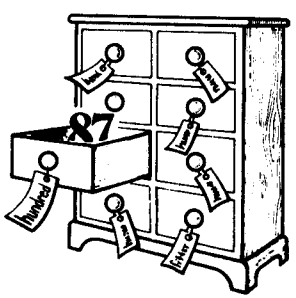
LET

It's about time we learned some of the keywords connected with putting numbers and words on the screen. The first one is PRINT. Try typing this on the keyboard:

```
print 2+2
```
ENTER

As you will have guessed, the answer is put on the next line of the screen and is, of course, 4. Now try this:

```
print "Hello"
```
ENTER



In this case we had to surround what we wanted on the screen by double quotes. The reason for this will become obvious later on. In the meantime we will explore another keyword.

Letting

Sometimes the world of computing seems to be a very strange place indeed. BASIC keywords are normal English words but they can have special meanings for the CPC 464. One of these is the keyword LET. In ordinary algebra you can write:

```
Let x=5
```

You cannot, however, write the following:

```
Let x=x+5
```

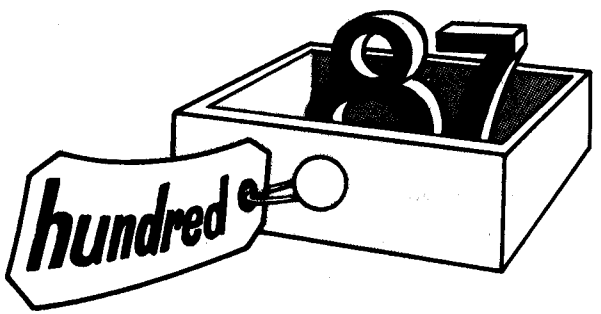
This is perfectly acceptable in BASIC. It means: 'Take the previous value of x, add 5 to it, and then take this number as the new value of x.' But why x? Well, we have just come across something called a variable - and for very good reason.

Variables are like something out of *Alice Through the Looking Glass*. They contain anything you care to put in them. There is nothing in BASIC that will stop you from

writing:

```
Let hundred = 87
```

If you don't like 'hundred' you could use 'h' or 'C', since what we are talking about are labels for empty boxes. When you use LET to define a variable as shown above, the CPC 464 writes the name on an empty box and then puts into it the value given on the right of the '=' sign.



Try typing this on your CPC 464

`Let hundred = 87` **ENTER**

You will get the reply 'Ready'. Then type:

`print hundred` **ENTER**

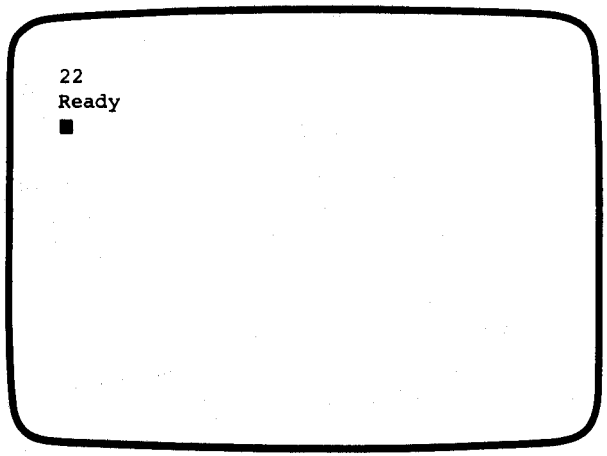
You will now have the following on your screen:

```
Ready
let hundred=87
Ready
print hundred
87
Ready
■
```

So, we are now at a point where we can enter our second program. The following five lines should be entered exactly as shown:

```
1Ø cls
2Ø let a=15
3Ø let b=7
4Ø let a=a+b
5Ø print a
```

Remember to press ENTER at the end of each line. Now try running this program. You should get the following result.



That may have seemed a bit obvious, but the next one might not be:

```
1Ø let a=5
2Ø let b=1Ø
3Ø let a=b
4Ø let b=a
5Ø print a+b
```

Try it for yourself. Can you see why the answer is 20? It may help you to think about numbers being moved from box to box. Remember that it is the variable to the *left* of the '=' sign that is changed by putting a new value into it, and that the CPC 464 is going to step through the program one line after the other.

Strings and things

We have seen that we can define a variable for numbers, and use it as if it were an actual figure. BASIC also allows us to define variables that may contain a series of letters, numbers and special characters. These are known as 'strings'.

A string variable is exactly the same as an ordinary variable except that you must end the variable name with a '$' sign. In addition you must always put quotation marks (") on either side of the characters you wish it to contain, otherwise the CPC 464 thinks you are trying to tell it about some other variable. Enter the following line:

`let a$=Hello` **ENTER**

You will get the error message:

`Type mismatch`

The CPC 464 thought that you were trying to put into the string variable a$ the numerical value of another variable called 'hello'. What you should have entered is:

`let a$="Hello"` **ENTER**

This time you can enter:

`print a$` **ENTER**

Try it.

Now enter the following program. See if you can work out what the results are going to be before you run it.

```
1Ø let a$="5"
2Ø let b$="12"
3Ø let a=5
4Ø let b=12
5Ø let c=a+b
6Ø let c$=a$+b$
7Ø print c
8Ø print c$
```

What's in a name?

You can use any name you like for a variable except that the CPC 464 will object to the use of any BASIC keyword. For example it will not accept:

```
let save=s+p
```
ENTER

This gives a 'syntax error' message. You would have to change this variable name to 'savers' or 'savings'. There is a complete list of keywords at the back of the *CPC 464 User Guide* if you want to find out which ones to avoid.

You have probably noticed that the CPC 464 doesn't mind whether you enter commands in either upper or lower case. But whatever you do it will always show keywords in upper case when a program is listed. If you keep all your variable names in lower case it will make them easier to pick out from a listing.

Savings

Until now, you have either been loading programs from the two datacassettes supplied with this course, or entering them on the keyboard. Now is your chance to save the above program for posterity, or, more to the point, so that we can use it again in this chapter and the next without having to enter it all over again. Take the datacassette out of the datacorder and replace it with a new one which has not had the record tag knocked out. Ordinary cassettes will do although anything larger than a C60 should be avoided because the tape is too thin. You should also beware of the tape leader on ordinary cassettes.

Rewind the cassette to the beginning and enter the following:

SAVE

```
save "variables" ENTER
```

You can choose a different name instead of VARIABLES if you wish, with upper and lower case letters and spaces in between words. You must always, however, put quotation marks on either side of the name. The CPC 464 will reply:

```
Press REC and PLAY and then any key:
```

Follow the instructions. You will then get the message:

```
Saving VARIABLES block 1
```

The CPC 464 will start the datacorder running and you will hear your program being transferred to the cassette. While the program is being saved the cursor will disappear from the screen, but will return when the CPC 464 has finished the operation and gives the 'ready' message. It is always good practice to have two copies of a program in case of accidents to the cassette, so repeat the above to save the program a second time. Don't forget to release the REC key on the datacorder afterwards.

More printing

If you have just run the program above, the CPC will still have the variables in memory. If not, load and run the program again so that we can use the variables to investigate another thing or two about the PRINT command. Enter the following:

```
print a,b,c
```
ENTER

You can see that this puts the three numbers up on the screen on the same line, but 13 character positions apart. It's a nice easy way of tabulating numbers but it only works for modes 1 and 2. Try the same thing in mode 0 to find out why. If you don't want the numbers spaced out you can do the following:

```
print a;b;c
```
ENTER

This time the numbers are printed on the same line but with no spacing between them at all, so we often have to add spaces as in the following example:

```
print "The value of c is ";c;" not ";c$
```

We can put letters and numbers anywhere we like on the screen by means of the keyword LOCATE. The command structure is:

```
locate x,y
```

This looks familiar, doesn't it? Be warned though. These x, y coordinates are not the same as the graphic coordinates. LOCATE moves the *text* cursor to the position on the screen given by the arguments of the command. Text coordinates start at the top left-hand corner of the screen (which has the coordinates x = 1, y = 1), and are counted across and *down* the screen.

Enter the following:

```
75 locate 2Ø,13
```
ENTER

Clear the screen with CLS. If you now run the program again with this new line, you will see that the value of c$ is printed in the middle of the screen, starting at character position 20 in line 13.

BARCHART

The name of the next program on the datacassette is BARCHART. It gives you a visual representation of four numbers between 0 and 290 – the sort of thing you see for election results or opinion polls. This type of program has to wait at certain points for the numbers to be entered into memory through the keyboard before it can continue. The keyword which does this is INPUT and a typical command is as follows:

INPUT

```
1Ø INPUT name$
```

The CPC 464 will wait patiently at line 10 until something is typed in and the ENTER key pressed. The information is then put into the string variable 'name$' and processing continues.

Now run BARCHART before studying the listing below. You will see that the CPC 464 puts a question mark (?), followed by the cursor, when it is waiting for input. It also puts a 'prompt' to let the user know what sort of information it is waiting for. You do this by putting a message after the INPUT keyword as follows:

```
INPUT "Amount (1-29Ø)";a
```

The prompt message must be enclosed by quotation marks (") and separated from the variable by a semi-colon (;).

You can see that many of the lines in BARCHART comprise several commands separated by colons (:). These colons serve the same purpose as starting a command with a line number and finishing it with ENTER. It is a good way of keeping a series of related commands together, particularly if they are very short.

A very good use of the colon is:

```
5Ø b=5Ø:REM bar size
```

Putting a REM alongside a command to explain its function makes the program easy to read at a later date.

```
10 REM 3D Bar Chart
20 REM by Dave Atherton
30 MODE 1
40 BORDER 14:INK 0,14:INK 1,0:INK 2,3:INK 3,24
50 b=50:REM bar size
60 LOCATE 1,23:INPUT "Amount (1-290)";a
70 x=100:PLOT x,55,1
80 DRAW x-b*2,55:DRAW x-b*2,a+55
90 DRAW x-b,a+55+b:DRAW x+b,a+55+b
100 DRAW x,a+55:DRAW x-b*2,a+55
110 MOVE x+b,a+b+55:DRAW x+b,b+55
120 DRAW x,55:DRAW x,55+a
130 LOCATE 1,23
140 PRINT"                                        "
150 LOCATE 1,23:INPUT "Amount (1-290)";a
160 x=260:PLOT x,55,2
170 DRAW x-b*2,55:DRAW x-b*2,a+55
180 DRAW x-b,a+55+b:DRAW x+b,a+55+b
190 DRAW x,a+55:DRAW x-b*2,a+55
200 MOVE x+b,a+b+55:DRAW x+b,b+55
210 DRAW x,55:DRAW x,55+a
220 LOCATE 1,23
230 PRINT"                                        "
240 LOCATE 1,23:INPUT "Amount (1-290)";a
250 x=420:PLOT x,55,3
260 DRAW x-b*2,55:DRAW x-b*2,a+55
270 DRAW x-b,a+55+b:DRAW x+b,a+55+b
```

(Continued)

```
280 DRAW x,a+55:DRAW x-b*2,a+55
290 MOVE x+b,a+b+55:DRAW x+b,b+55
300 DRAW x,55:DRAW x,55+a
310 LOCATE 1,23
320 PRINT"                                                    "
330 LOCATE 1,23:INPUT "Amount (1-290)";a
340 x=580:PLOT x,55,1
350 DRAW x-b*2,55:DRAW x-b*2,a+55
360 DRAW x-b,a+55+b:DRAW x+b,a+55+b
370 DRAW x,a+55:DRAW x-b*2,a+55
380 MOVE x+b,a+b+55:DRAW x+b,b+55
390 DRAW x,55:DRAW x,55+a
```

Game number 4

Some people use a lot of clever-sounding words - especially in the computer world. Our automatic BUZZWORD generator will help you to strike back at them.

Testing

Run SAT6 to see if you need to re-read any of this chapter before going on to the next one.

7 GETTING IT RIGHT

A lot of this chapter is about corrections and changes to programs. The program you entered in the previous chapter will be ideal for this purpose so rewind the cassette and get the program back into memory by entering:

```
load "variables"
```
ENTER

'Variables' is the name you gave to the program when you used SAVE to store it on the datacassette.

Changing lines

When you are entering a program by typing on the keyboard you are almost certain to make mistakes, even if you are just copying from a printed page. The most common mistake is when you hit the wrong key. You usually realise at once what you have done so, before pressing the ENTER key, backspace and rub out the offending characters by using the DEL key.

While you are still entering a line (this is called the 'current' line), you can move the cursor backwards and forwards over this line by using the cursor keys. New characters can be inserted by just typing them in, and you can remove characters by means of the CLR and DEL keys.

In the previous chapter we also saw that you can replace complete lines by merely re-entering them. The CPC 464 will then automatically put the new line in place of the old one.

As you write more of your own programs, you are going to find that they will not usually run properly the first time round. The CPC 464 will give you some error messages as you enter lines, and others when you try to run the program. But it cannot tell you, for example, if you have left out commands or forgot to move the cursor to a new position before drawing a line. This is the reason why the line numbers we have used go up in tens. As you saw in the previous chapter, we added a line between 70 and 80 by entering:

```
75 locate 2Ø,13
```

In extreme cases there is nothing to stop you adding up to nine lines in this or any other position. In the same way, you may delete lines by entering a blank line. The line we added in the above example can be removed as follows:

75 ENTER

The CPC 464 will then eliminate line 75 from the program. Try it and then LIST the program to see what has happened.

Editing

When you need to modify a program that has already been entered or loaded, you can use the EDIT function of the CPC 464. Using the example above, enter the following:

EDIT

```
edit 4Ø
```
ENTER

As you can see, line 40 is displayed on the screen with the cursor over the first character of the line. You then use the left and right cursor keys to position the cursor over the part of the line you want to change as if you were working on the current line. Try this out for yourself by changing the value of b from 15 to 26. Position the cursor over the '1' of '15' and then press the CLR key twice. This will erase the 15. Now enter 26 and press the ENTER key. The cursor need not be at the end of the line. If you now LIST or RUN the program you will see that the CPC 464 has altered the line in the way you wanted.

An easier way of editing lines is to use the COPY key. A second cursor, called the copy cursor, is used to pick out lines or parts of lines from anywhere on the screen.

The copy cursor is obtained by holding down SHIFT while pressing one of the cursor keys. List the program again and try it, positioning

the copy cursor at the beginning of one of the lines. You can see that the normal cursor has stayed in place. Press the COPY key to copy each character onto the current line with the normal cursor. If you hold the COPY key down it will repeat automatically. You can stop copying at any position on the original line and then enter new or changed information on the current line before resuming copying again.

Once you have got used to using the copy cursor you probably won't use the EDIT command very often. The effect is almost identical but with the added advantage that you can leave out of a line the characters that you no longer want.

To let

You may have found it a bit tedious typing in LET at the beginning of lines in the last chapter. It's time to make confession. You don't need to use this keyword in Amstrad BASIC.

Try editing VARIABLES again to remove all the LETs. If you now run the modified program you'll find that it has made no difference at all!

Branch lines

We said earlier that line numbers show the CPC 464 the order in which commands are to be stored, but we also said that they are not always executed in that order. There are times when we want to skip backwards or forwards through a program, and the keyword to use is GOTO. The command is formed by adding a line number, for example:

```
7Ø goto 13Ø
```

This line will make the CPC 464 skip all the lines in a program between 70 and 130. You can do the same thing in reverse:

```
13Ø goto 7Ø
```

The program will now go back to line 70 and execute the lines up to 130 - and continue doing so until you switch off or press the ESCAPE key.

What happens next?

We are faced with decisions every hour of our waking lives, even if they are only trivial things such as choosing whether to drink coffee or tea, or decide which shoes to put on before going out. Having made up our minds what to do next, e.g. drink coffee, tea, or nothing; or put on black, red, or no shoes, we then take an appropriate course of action. The CPC 464 carries out alternative courses of action by means of the keyword IF.

GOTO

IF

The following line could be from a program to check the amount of money you have saved up, the variable 'money' being the current balance:

```
IF money=Ø THEN PRINT "Hard up"
```

THEN

The interesting thing is that the PRINT command will *only* be executed if 'money' is 0 - otherwise the CPC 464 just steps on to the next line in the program.

An even better way of doing the same thing is to include yet another keyword, ELSE, in the following way:

ELSE

```
IF money>Ø THEN PRINT "Rich" ELSE PRINT "Hard up"
```

The sign '>' means 'greater than' or 'more than' and takes care of the situation where you owe the bank money! If 'money' is zero or less, the CPC 464 ignores what is between THEN and ELSE, and executes whatever is after ELSE. ELSE gives you the possibility of an alternative course of action before going on to the next line in the program.

In the previous chapter we looked at a program called BARCHART. There was an input statement which said:

```
Amount (Ø-29Ø)?
```

If yo tried to enter a number over 290 the program didn't seem to notice and drew the bar off the top of the screen. You could stop this from happening by limiting the variable 'a' to its maximum value. For example:

```
IF a>29Ø THEN a=29Ø
```

As you can see, the IF . . . THEN command isn't restricted to PRINT statements, and this line would limit 'a' to a value that would fit on the screen.

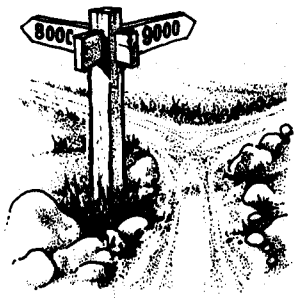
Going places

The IF keyword really comes into its own when you use it with GOTO. It is like a signpost that tells the CPC 464 which part of the program it should execute next. Supposing you wanted to stop people entering numbers larger than 295 for the BARCHART program. The following lines would do the trick:

```
6Ø INPUT "Amount (Ø-29Ø)";a
65 IF a>29Ø GOTO 6Ø
```

Until a number less than or equal to 290 is entered, the CPC 464 will just keep looping back to line 60. This 'trap' will actually stop out-of-range numbers being entered rather than just cutting them down to size.

Bug hunting



You may imagine that if you made a small error in writing one of the instructions above, the CPC 464 could go to the wrong place in a program. You're right. We have just come across one of the classic jargon words in computing - the bug.

Machines develop faults, people make mistakes. Earlier on we said that computers can't think, and that the programmer has to do a computer's thinking for it. If the programmer *does* make a mistake, and even the best are not immune, it usually is only obvious after an attempt is made to run the program. The CPC 464 does what it was told to do but that may not be what the programmer originally intended.

De-bugging is the process of going through a program and correcting the errors of logic or understanding that it contains. There is no shame or ignominy in writing a program with bugs in it. Admittedly, experienced programmers have fewer bugs to remove when they have finished a program than do beginners, but this is just a matter of knowledge and practice. In time you will find that you will introduce fewer and fewer bugs into your programs.

It is good practice to check through a program before running it. One way of doing this is to give it a 'dry run' - going through the program line by line and writing down the values of variables and the products of sums. Let's go through an example:

```
1Ø a=Ø
2Ø print a
3Ø a=a+1
4Ø if a<4 then goto 2Ø
```

Before entering this into the CPC 464, get a piece of paper and write down the following three headings:

Step Line number Val. of a

Now go through the program and execute each instruction exactly the way the CPC 464 would. This is how it should look:

Step	*Line number*	*Val. of a*
1	*10*	*0*
2	*20*	*0*
3	*30*	*1*
4	*40*	*1*
5	*20*	*1*
6	*30*	*2*
7	*40*	*2*
8	*20*	*2*
9	*30*	*3*
10	*40*	*3*
11	*20*	*3*
12	*30*	*4*
13	*40*	*4*

The technique of using a dry run is a good way of forcing yourself to see the program from the machine's point of view. Any problems often then become obvious. When you have just written a program your main difficulty can be that you know exactly how you expect it to work and can't bring yourself to see how it might not!

Another technique is to add temporary PRINT instructions at certain points in the program to show the value of the variable at that point.

A simple example would be to add the following line to the above program.

```
35 print "Line 35 a= ";a
```

You can also use the command STOP. For example:

```
35 stop
```

The CPC 464 will STOP executing the program at this point and you can ask it to print the variable or variables that interest you by direct print commands. The program is restarted by entering:

```
cont
```
ENTER

This stands for CONTinue. Remember that it won't work if any lines are added or deleted after the program has been STOPped.

Renovation

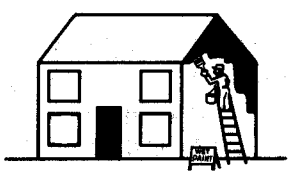
Yes, it's back to the house again. This time it could do with a lick of paint here and there so get out the colour charts and load the program DECO.

STOP

This isn't really a game, just an entertaining way of combining all the commands you have learned so far. You may also like to modify the program to give your picture a bit of individuality. You should now know enough to do this using the techniques described at the beginning of this chapter.

CONT

Don't worry about any unfamiliar keywords in the listing of DECO given on the next few pages. All will be made clear in later chapters.



```
10 REM deco
20 MODE 0
30 CLS
40 REM ** start **
50 BORDER 12
60 INK 0,12:REM yellow
70 INK 3,3:REM red
80 INK 6,6:REM bright red
90 INK 9,9:REM green
100 PAPER 0
110 REM draw front
120 MOVE 100,50
130 DRAW 100,250,3
140 DRAW 400,250
150 DRAW 400,50
160 DRAW 100,50
170 REM draw side
180 MOVE 400,250
190 DRAW 600,250
200 DRAW 600,50
210 DRAW 400,50
220 DRAW 400,250
230 REM draw gable end
240 REM already at start point
250 REM so no need for a MOVE
260 DRAW 500,350
270 DRAW 600,250
280 DRAW 400,250
```

(Continued)

```
290 REM draw roof
300 REM only two lines needed
310 MOVE 100,250
320 DRAW 200,350
330 DRAW 500,350
340 REM draw door
350 MOVE 225,50
360 DRAW 225,140,6
370 DRAW 275,140
380 DRAW 275,50
390 REM draw windows
400 REM left hand bottom
410 MOVE 120,70
420 DRAW 120,130,9
430 DRAW 180,130
440 DRAW 180,70
450 DRAW 120,70
460 REM left hand top
470 MOVE 120,170
480 DRAW 120,230
490 DRAW 180,230
500 DRAW 180,170
510 DRAW 120,170
520 REM right hand top
530 MOVE 320,170
540 DRAW 320,230
550 DRAW 380,230
560 DRAW 380,170
```

(Continued)

```
570 DRAW 320,170
580 REM right hand bottom
590 MOVE 320,70
600 DRAW 320,130
610 DRAW 380,130
620 DRAW 380,70
630 DRAW 320,70
640 REM *** DECO ***
650 r$=CHR$(18)
660 LOCATE 1,25
670 PRINT"Type a colour (1-15)";
680 FOR i=1 TO 15
690 INK i,i:NEXT i
700 LOCATE 1,1:PRINT r$;
710 INPUT "Roof colour";r
720 FOR i=107 TO 399 STEP 2
730 MOVE i,252:DRAW i+96,349,r
740 NEXT i
750 LOCATE 1,1:PRINT r$;
760 INPUT "Gable end";g
770 FOR i=252 TO 346
780 MOVE i+154,i:DRAW 848-i,i,g
790 NEXT i
800 LOCATE 1,1:PRINT r$;
810 INPUT "End wall";e
820 FOR i=52 TO 248 STEP 2
830 MOVE 404,i:DRAW 598,i,e
840 NEXT i
```

(Continued)

```
850 LOCATE 1,1:PRINT r$;
860 INPUT "Front";f
870 FOR i=52 TO 248 STEP 2
880 MOVE 104,i:DRAW 398,i,f
890 NEXT i
900 LOCATE 1,1:PRINT r$;
910 INPUT "Door";d
920 FOR i=52 TO 138
930 MOVE 229,i:DRAW 268,i,d
940 NEXT i
950 LOCATE 1,1:PRINT r$;
960 INPUT "Window";w
970 FOR j=0 TO 100 STEP 100
980 FOR i=70+j TO 130+j
990 MOVE 120,i:DRAW 180,i,w
1000 MOVE 320,i:DRAW 380,i
1010 NEXT i
1020 NEXT j
1030 END
```

Testing

Running SAT7 will allow you to check how well you understood this chapter. Don't worry if you have to thumb back through the pages before answering a question. Not many programmers work without a reference book at their elbow.

8 HOUSE IMPROVEMENTS

In computer programming there are some tasks, like housework, that need to be done over and over again. The CPC 464, like any other computer, is very good at repetitive tasks. If we give it the appropriate program it will keep repeating it until we tell it otherwise. So, we are going back to the house again to see how we can fill in a lot of details without having to do a lot of programming.

Load the next program on the cassette, MANSION. It may seem to be the same old house, but this time we are going to see how we can produce very clever visual displays by using the principles of the 'loop' and the 'subroutine'.

Looping

Surprising as it may seem, you have already seen a loop in the previous chapter. Although we didn't call it that at the time, the following program contains a loop:

```
1Ø a=Ø
2Ø print a
3Ø a=a+1
4Ø if a<4 then goto 2Ø
```

What this program means is 'Print the value of a, loop back to line 20, and do this until a is greater than 3'.

We could have written the program as follows:

```
1Ø for a=Ø to 3
2Ø print a
3Ø next a
```

If you look at the listing of the MANSION program, printed below, you will see that it no longer ends at line 640. We are first going to look at the new lines from 640 to 680 inclusive.

If you run the program, it will draw the house as usual. When it reaches the STOP command you can run the next bit by entering:

CONT **ENTER**

Try it. What do you think of the fence? It may even keep the neighbour's dog out of the garden!

The key to this operation is line 650, so let's have a close look at it:

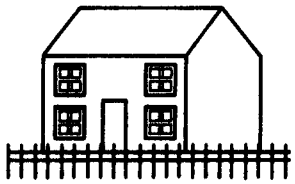
```
65Ø FOR F=Ø TO 62Ø STEP 2Ø
```

The FOR command tells the CPC 464 that it is just going into a loop. 'F = 0' says where the loop starts; 'TO 620' says where the loop stops. The instruction 'STEP 20' gives the distance between fence posts by telling the CPC 464 to increase 'F' by 20 each time it goes round the loop.

Line 660 uses the current value of the variable 'F' to move the graphics cursor to the next fence post position and to draw the line of the post.

The instruction that completes the loop is NEXT, as you can see in line 670. It means 'Give me the next value of F', using the 'step' we described previously.

So, to summarise, the CPC 464 goes through a loop which steps through F = 20, 40, 60, 80, etc., until the 'next' number is 640, i.e. past the end of the loop. The CPC 464 then 'drops through' the NEXT and executes line number 680 to draw the cross beams on the fence.



FOR

STEP

NEXT

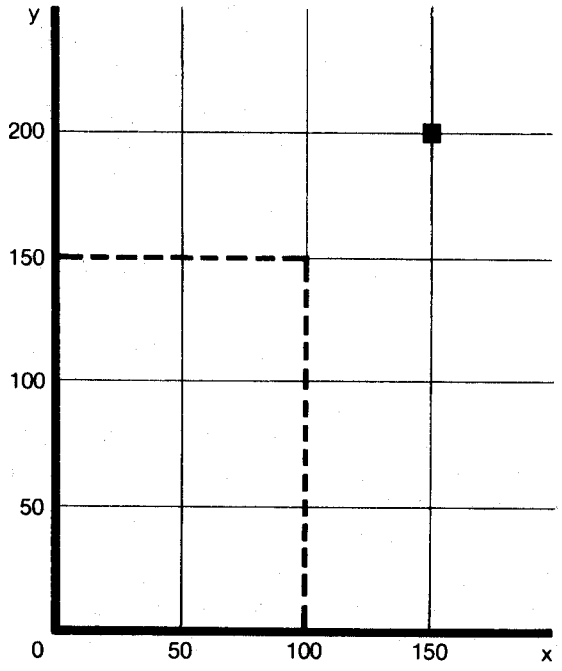
PLOTR

MOVER

DRAWR

Relativity

Before we go on to the next bit, let's just finish off our study of graphics commands with the 'relative' versions of PLOT, DRAW and MOVE. You will remember that with all three commands the arguments are the x and y coordinates measured from the graphics origin of 0,0. For the commands PLOTR, DRAWR and MOVER, the arguments are the required x and y displacement starting from the current position of the graphics cursor.



From the above diagram you can see that the following instruction would move the graphics cursor to x = 150, y = 200:

```
mover 5Ø,5Ø
```
ENTER

The same philosophy applies to the DRAWR and PLOTR commands. Their arguments will always apply to the relative coordinates starting from the current cursor position, and thereby allow the programmer to avoid having to calculate the coordinates every time. Their usefulness will become obvious before the end of the chapter.

Doing the windows

You will not have failed to notice that we haven't finished with the house yet. After the fence-drawing loop there is another STOP command in line 685, followed by some lines which include a new command - GOSUB. Run the program through to the end by entering CONT.

Well, it was about time we had some panes of glass in the windows. This was done by means of a 'subroutine', which is the term used in computer programming for a series of instructions that can be called up repeatedly as and when necessary. If we didn't use a subroutine for the window panes it would have meant writing the same set of instructions 16 times. GOSUB stands for GO to SUBroutine and is always followed by the line number of the first instruction of the subroutine.

GOSUB

```
10 REM mansion
20 MODE 0
30 CLS
40 REM ** start **
50 BORDER 12
60 INK 0,12:REM yellow
70 INK 3,3:REM red
80 INK 6,6:REM bright red
90 INK 9,9:REM green
95 INK 15,15:REM orange
100 PAPER 0
110 REM draw front
120 MOVE 100,50
130 DRAW 100,250,3
```

(Continued)

```
140 DRAW 400,250
150 DRAW 400,50
160 DRAW 100,50
170 REM draw side
180 MOVE 400,250
190 DRAW 600,250
200 DRAW 600,50
210 DRAW 400,50
220 DRAW 400,250
230 REM draw gable end
240 REM already at start point
250 REM so no need for a MOVE
260 DRAW 500,350
270 DRAW 600,250
280 DRAW 400,250
290 REM draw roof
300 REM only two lines needed
310 MOVE 100,250
320 DRAW 200,350
330 DRAW 500,350
340 REM draw door (in red)
350 MOVE 225,50
360 DRAW 225,140,6
370 DRAW 275,140
380 DRAW 275,50
390 REM draw windows (in green)
400 REM left hand bottom
410 MOVE 120,70
```

(Continued)

```
420 DRAW 120,130,9
430 DRAW 180,130
440 DRAW 180,70
450 DRAW 120,70
460 REM left hand top
470 MOVE 120,170
480 DRAW 120,230
490 DRAW 180,230
500 DRAW 180,170
510 DRAW 120,170
520 REM right hand top
530 MOVE 320,170
540 DRAW 320,230
550 DRAW 380,230
560 DRAW 380,170
570 DRAW 320,170
580 REM right hand bottom
590 MOVE 320,70
600 DRAW 320,130
610 DRAW 380,130
620 DRAW 380,70
630 DRAW 320,70
635 STOP
640 REM fence
650 FOR F=0 TO 620 STEP 20
660 MOVE F,0:DRAW F,60,15
670 NEXT F
680 MOVE 0,45:DRAW 620,45
```

(Continued)

```
685 STOP
690 REM frames in windows
700 REM use a square drawing subr
710 size=18
720 MOVE 130,78:GOSUB 900
730 MOVE 156,78:GOSUB 900
740 MOVE 130,103:GOSUB 900
750 MOVE 156,103:GOSUB 900
760 MOVE 130,178:GOSUB 900
770 MOVE 156,178:GOSUB 900
780 MOVE 130,203:GOSUB 900
790 MOVE 156,203:GOSUB 900
800 MOVE 330,78:GOSUB 900
810 MOVE 356,78:GOSUB 900
820 MOVE 330,103:GOSUB 900
830 MOVE 356,103:GOSUB 900
840 MOVE 330,178:GOSUB 900
850 MOVE 356,178:GOSUB 900
860 MOVE 330,203:GOSUB 900
870 MOVE 356,203:GOSUB 900
880 END
890 REM subroutine for square
900 DRAWR 0,size,9
910 DRAWR size,0
920 DRAWR 0,-size
930 DRAWR -size,0
940 RETURN
```

Now you can see why we had to tackle the relative graphic commands before continuing with this part of the chapter. If we have a standard subroutine for a particular task, in this case the drawing of a square, it must be possible to describe that task in a universal way. In our window pane subroutine you can see that the use of DRAWR and the variable 'size' enable us to draw a square of any size, starting at any position on the screen.

A subroutine should start with at least one REM to describe what it does. In a long program with many subroutines it would be foolish not to. In Chapter 9 we will see how important this is and learn what other information must be given. You must, however, end a subroutine with the command RETURN.

When the CPC 464 comes across a GOSUB command it makes a note of where it had got to in the program before going off and executing the commands in the subroutine. The RETURN at the end of the subroutine is a bit like the NEXT instruction in a FOR loop, but whereas NEXT takes you back to the beginning of the loop, RETURN sends the CPC 464 back to the next instruction after the GOSUB that called the subroutine.

A program can use subroutines to draw a picture of a house in the same way that a builder uses subcontractors to build one. Tradesmen such as bricklayers, carpenters, plumbers and tilers are called in to do specific jobs. And these specialists may themselves subcontract certain parts of their tasks. For example, a bricklayer will almost certainly get a labourer to mix his mortar for him and the days when he would have made his own bricks are long past. Furthermore, the same labourer could also work for any of the others trades.

You can construct programs in the same manner. A subroutine called by a GOSUB can call another subroutine to carry out part of its task, and so on. The CPC 464 can keep track of many levels of GOSUBs and will still find its way back to where each one was called.

Finishing off

Subroutines are always put at the end of a program. But what happens when the last command has been executed? Well, if you don't take any precautions the program will drop straight into the beginning of the first subroutine with occasionally comic but usually silly results. One way to avoid trouble is to put in an END command. Quite simply it is, as you can see in MANSION:

END

```
8ØØ END
```

When the CPC 464 reaches such a line it stops running the program and goes back to READY.

Another method is to put in a loop such as:

```
8ØØ GOTO 8ØØ
```

This will keep running until either you press ESC or you do a general reset. It's useful if you don't want READY to appear on the screen.

Exercises

Try using the window subroutine to put a nice big picture window in the right-hand end of the house to overlook the garden.

Write a subroutine that will draw a chimney and then use this subroutine to put the chimney on the left, right, or centre of the roof.

Testing

This time you will not only be tested on the subjects covered in this chapter, you will also be given a bit of revision on previous subjects. It is quite reasonable for you to check back through the book to find the answer to a question. No one can be expected to memorise the amount of information you have been given in such a short period of time.

PROGRAM DESIGN

It cannot be overemphasised that there is as much craft in programming as there is art and science. You will remember that we have already said, more than once, that computers can't think. The craft in computer programming is the careful analysis of a problem or task and the conversion of the information obtained into a logical, coherent program that will not use too much memory space and will be fast enough to do the job properly.

Obviously you can't expect to do this without knowledge and experience, but this chapter may help you to acquire certain habits and techniques of good programming that will serve you well in years to come.

Working from objectives

For any program longer than, say, 20 lines, there is only one way to approach the task. You have to break it down into manageable pieces. No matter how good you are, amateur or professional, it is not possible to carry all the aspects of a large program in your head at the same time.

So, the first thing to do when starting on a program is to divide it into parts and write down what each part is supposed to do. It is worth starting a 'project' book to record all these design parameters for each program and its subroutines. If you don't, you may get half way through writing a program and then find that you have forgotten the reason for a specific requirement, or the exact specification for what a subroutine does. From the overall description of what the program does, you must then break it down into an ordered, logical structure of tasks.

And quite how is this done, you ask? Well, there are various methods in use, but we are going to explain just one of them. It makes use of something called Programming Development Language - better known as PDL. Let's take a familiar example. A postman has to go through a series of decisions and actions as he goes on his round. Imagine that you had to program a robot postman to do the same thing. Since robots are controlled by computers, you have to tell them what to do in every case. So take a look at the following:

Program for robot postman

```
Walk to first house in street
For every house in the street
     If there is something for this house
     Then Repeat
                Get packet from bag
                Until no more for this house
          If there is a front gate
          Then open it
                Step through it
                Close gate
          While not at the front door
                Walk towards door
          If any recorded delivery/excess post/parcel
          Then ring doorbell
                Wait for answer
                For each packet
                     In case:
                          Recorded:    get signature
                          Excess:      collect money
                          Otherwise:   hand over package
          Else push everything through letterbox
          While not at entrance
                Walk to entrance
          If there is a front gate
          Then open gate
                Step through
                Close gate
     Else walk to next house
Next house
```

We have written a plan for the program using ordinary English in a very formal and organised fashion.

There are a number of important things to note about the above example. The first is that there are a small number of building blocks for the program, some of which you will recognise as BASIC commands and some not. Another is that the text is indented to show which actions take place inside loops or IF-THEN-ELSE blocks. Pay careful attention to this when writing your own programs in this way or you may get very confused!

Perhaps the term 'routine' needs to be explained. A series of actions (or commands) that perform a particular task are grouped together to form a routine. Look at the second line of the example above:

```
For every house in the street
```

This is the start of a routine that may be repeated for any house in any street, as long as the rest of the program takes care of the rest of the problem.

Within this routine there are several subroutines (yes, this is the *real* meaning of the term). The following one is typical:

```
While not at front door
      walk towards door
```

You can see that this could be handled by a subroutine that took care of walking from any starting point to any finishing point.

The most important point about our example is that not too much attention is paid to defining each action in detail at this stage. If you continued breaking this program down you would find that 'walk' alone would represent pages and pages of work even without the problems of balancing on two feet. But at this level a simple general description is enough.

When all the tasks are broken down successively, we would eventually reach a point where each action of the program corresponded to individual actions concerning our robot's sensors or control mechanisms.

If the above boggles your mind, here is a comforting thought. It would take the average team of programmers many years of hard work to actually write the above program, so it is probably a good idea to scratch it off your list of projects. The reason we chose this example, however, is to show you how even the most complex tasks can be expressed simply.

Exercises

There are several design parameters missing from the above program. See if you can design the subroutines to take care of:

- No answer to the bell
- No letterbox
- Hands too full to open the gate
- Street with house names only

And you can probably think up others as well. Don't be discouraged if you can't solve many of them. The things that we human beings take in our stride can be unbelieveably complicated.

Building blocks

Here are some of the building blocks of our postman program with examples of how they might be translated into Amstrad BASIC.

```
For every house in the street
Next house
```

```
8ØØ FOR house=firsthouse TO lasthouse
.....................
87Ø NEXT
```

```
Repeat
     Get packet from bag
     Until no more for this house
```

```
1ØØØ GOSUB 12ØØØ:REM get packet from bag
.....................
111Ø IF waslastpacket=Ø THEN GOTO 1ØØØ
```

This assumes that the subroutine at line 12,000 sets the variable 'waslastpacket'.

```
While not at the front door
     Walk towards door

      2ØØØ GOSUB 13ØØØ:REM see if we are at door
      2Ø1Ø IF atdoor<>Ø then 2Ø4Ø
      2Ø2Ø GOSUB 14ØØØ:REM walk towards door
      2Ø3Ø GOTO 2ØØØ
      2Ø4Ø REM end of while
```

The subroutine at line 13,000 would set the variable 'atdoor'.

```
In case:
     Recorded:    get signature
     Excess:      collect money
     Otherwise:   hand over packet

      3ØØØ GOSUB 15ØØØ:REM get type of packet
      3Ø1Ø IF packettype=recd THEN GOSUB 16ØØØ:GOTO 3Ø4Ø
      3Ø2Ø IF packettype=excess THEN GOSUB 17ØØØ:GOTO 3Ø4Ø
      3Ø3Ø GOSUB 18ØØØ:REM hand over packet
      3Ø4Ø REM end of "in case"
```

The subroutine at 15,000 sets the variable 'packettype' and the variables 'recd' and 'excess' have been set to suitable values.

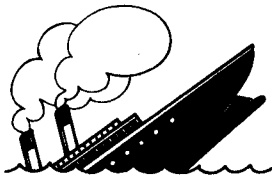
```
If any recorded delivery/excess post/parcel
Then ring doorbell
Else push everything through letterbox
```

Here is one possible translation:

```
4ØØØ GOSUB 1ØØØØ:REM see if recorded/excess/parcel
.....................
4Ø7Ø IF needring<>Ø THEN GOSUB 2ØØØØ ELSE GOSUB 21ØØØ
```

Or you may want to keep the actual routines close to the IF-THEN-ELSE commands for reasons of clarity, rather than use subroutines, so the translation could also be:

```
4ØØØ IF needring=Ø THEN GOTO 41ØØ
..........:REM get answer etc.
4Ø9Ø GOTO 42ØØ
41ØØ REM don't need to ring
..........:REM put everything through letterbox
42ØØ REM end of "IF needring"
```



Routine work

Here is a checklist of what you should work out before you write a routine:

- Name of routine (for your own use – not the CPC 464)
- Names of variables that need to be set before using the routine
- Effect of routine on variables
- Any side-effects of the routine

When you design programs it is a good idea to use separate sheets of paper that can later be put into a ring binder to form your project book. Each sheet should have the above information written at the top. This has two advantages:

1 You can see at a glance what you intend to write.

2 It is easier to identify two very similar sub-routines that could be combined into a single one to save you having to do the same thing twice.

Do you remember the routine that drew the window panes on the house? It started with a REM:

```
REM subroutine for square
```

The only variable set up before calling the subroutine was 'size', and the subroutine merely used the variable but did not change it. Finally, the routine did not have any other effect on other parts of the program or the CPC 464 itself.

Designing routines in this way makes them 'watertight'. You know exactly what to do to use them, you know what they do for you, and you know what they may affect. Not only are programs designed this way much easier to debug; such a design avoids having unexpected results spreading round until the whole program sinks faster than, and just as catastrophically, as the *Titanic*.

Designing programs using this routine-by-routine method means that by the time you have finished, you end up with a sheaf of paper that tells you exactly what to write.

Documentation

As an amateur programmer, you probably work entirely on your own. Thus, any programs you write will tend to be proud possessions which you will be unwilling to either disclose to the outside world, or have subjected to critical scrutiny. Professional programmers don't work like this. Most professionals work in teams on the same program, or set of programs, and therefore must be able to read and understand whatever anyone else has written without difficulty.

BASIC is an accessible, tolerant language that doesn't insist on any rigid structure or form to a program. This is all very well when you are a beginner, but as soon as the tasks become a little more complex it is necessary to start making the programs clear and easy to understand. Otherwise you will find yourself rewriting programs you only finished a month previously simply because you can no longer make head or tail of them, or you can't see how to change them.

The secret of success is REM. And REM and REM again. When you have a branch or a GOSUB in your program just put a REM in to say why and where it is going. Putting GOSUB 4,000, for example, is acceptable until there are 50 other subroutines. When this is the case you need to write (for example again):

```
GOSUB 4ØØØ:REM Move cursor to position
```

We talked about clearly defining what a routine has to do and the variables involved. The way to do this within a program is to put all the appropriate information in a series of REMs at the beginning of each routine. It doesn't have much effect on the processing time and has a magical effect on readability, both for yourself and for other programmers.

10 SOUNDS FANTASTIC

When we said earlier that BASIC programs written for the CPC 464 might not work on other computers, it was particularly with the sound commands in mind. The range and power of the CPC 464's BASIC sound commands are such that few other computers can equal them.

But this creates a problem as well. It would be quite possible to write a separate book on the sound commands alone. So, in keeping with the intention of learning the first steps in Amstrad BASIC, a lot of information has been left out of this chapter.

Having said all that, load the next program from the cassette. It is called ZAPPOW, and will give you an idea of what can be achieved with these sound commands. A listing of this program is given at the end of this chapter so that you can copy the interesting bits into your own programs.

Tuning up

As you may have guessed, the command that produces the sound is SOUND. Enter the following:

```
sound 1,478
```
ENTER

That was middle C. The number 478 is called the 'period', and determines the note to be played. A full list of notes and their associated periods is given in the *Amstrad CPC 464 User Guide*. The 1 is the sound 'channel' to be used. Part 2 of this course describes how to use the other two channels.

Now enter the following:

```
sound 1,478,2ØØ
```
ENTER

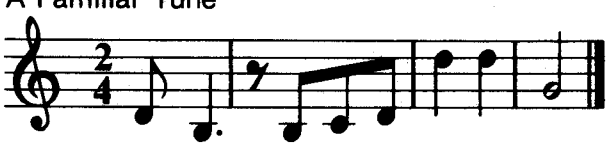
When you press the ENTER key this time the sound will go on for precisely 2 seconds. This is governed by the 200 which defines the 'duration' of the sound in hundredths of a second. If you omit this part of the command, the CPC 464 assumes that the duration wanted is 20. You may now write a program to play a familiar tune. For example:

```
1Ø rem A Familiar Tune
2Ø sound 1,213
3Ø sound 1,253,6Ø
4Ø sound 1,Ø,4Ø
5Ø sound 1,253
6Ø sound 1,239
7Ø sound 1,213
8Ø sound 1,127,4Ø
9Ø sound 1,Ø,1
1ØØ sound 1,127,4Ø
11Ø sound 1,159,6Ø
12Ø end
```

SOUND

Lines 40 and 90 are interesting. To give a 'rest' (music terminology for a pause) between two notes, you must put in a command whose period is zero – silence, in other words. In line 40 we have a rest for two beats; in line 90 an extremely brief rest makes sure that the repeated note does not merge with the first one.

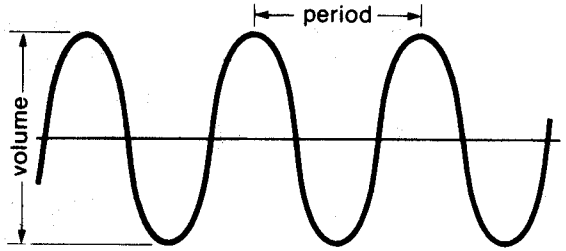
A Familiar Tune



For those of you who are familiar with music, the same tune is shown above in normal notation so that you can compare it with the program.

Sounds BASIC

Before we go any further it is necessary to have a look at how sounds are built up. In its simplest form a sound can be represented graphically by a sine wave, as you can see below.

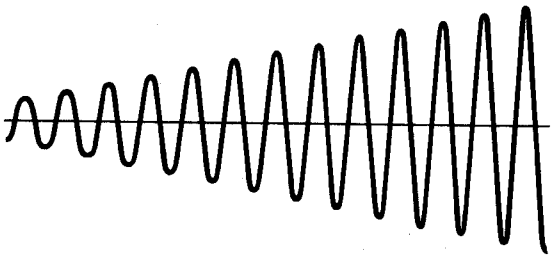


Perhaps it will now be clear what the period part of the SOUND command really means. It is the time that elapses between two waves of the sound, measured in steps of 8 microseconds (millionths of a second). The shorter the time between waves, the higher the sound, and the longer the time between waves, the lower the sound.

The height of the waves is known as the volume, and we can specify this in the sound command. Try entering our middle C line again, only this time put a 2 on the end as follows:

```
sound 1,478,2ØØ,2
```
ENTER

That makes it very quiet doesn't it? The volume can be specified between 0 and 7, 0 meaning no volume at all (yes, it can be useful) and 7 meaning maximum volume. The sound we looked at above started and finished at the same volume. In the real world things aren't usually like that. It is quite normal to have a sound which changes in volume from start to finish.

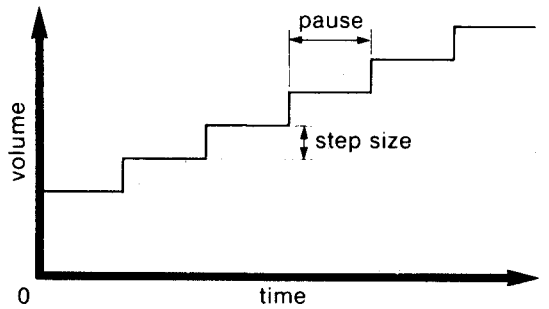


ENV

We do this on the CPC 464 with the ENV command, which gives you an 'envelope' that can vary the volume of a sound throughout its duration. Enter the following two lines and run them:

```
1Ø env 1,6,1,3Ø
3Ø sound 1,478,18Ø,1,1
```

You will probably need to study the following diagram closely to understand how this works. Don't worry about the missing line 20 – we'll come to that in due course.



You can have up to 15 different volume envelopes in one program, and the first 1 in the ENV command above merely identifies it as the first. The next number, 6, means that you want the volume to change in 6 equal steps. The next number, 1, gives the amount by which you want the volume to increase. If this was -1, it would *decrease* the volume. And finally, the 30 gives the length of the step in hundredths of a second.

Be careful about the volume specified in the SOUND command. The CPC 464 can actually set the volume at *sixteen* different levels when a volume envelope is specified. The range then becomes 0 to 15 where levels 0, 2, 4, 6, 8, 10, 12, and 14 correspond to the range 0 to 7 that you get when a volume envelope is *not* specified.

Now let's analyse how SOUND and ENV work together in our example. The volume is set at 1 in the SOUND command, and ENV 1 gives six steps of increasing the volume by one level, so the final volume will be 1+6=7. Note that the first step takes place immediately the SOUND command is executed, so the initial volume is 1 + 1 = 2. Note also that the six steps of the envelope multiplied by the pause duration shouldn't be longer than the duration specified in the SOUND command, or otherwise the end of the envelope will be lost.

So now let's have a look at the missing line number 20. In the same way that the volume levels in a SOUND command are modified by a volume envelope, the frequency of the sound can be modified by a 'tone' envelope. The command used is ENT. Here is line 20 in place at last:

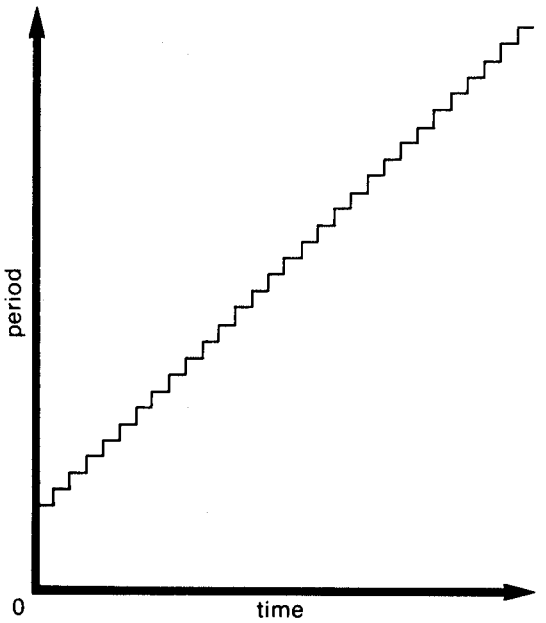
ENT

```
1Ø env 1,6,1,3Ø
2Ø ent 1,18Ø,1,1
3Ø sound 1,478,18Ø,1,1,1
```

As you can see, our SOUND command has acquired yet another 1 to indicate that ENT 1 is to be used.

The structure of the ENT command is very similar to ENV, as you can see from the

diagram below. This time, though, it is the period of the sound that is increased one at a time for 180 steps, each one being one-hundredth of a second long.



If you haven't already done so, run the program above several times. Then change the step value of both the ENV and ENT commands to -1 instead of 1, change the initial volume setting of the sound command to 7, and then run it again.

Noisy sounds

The CPC 464 can add a little noise to any sound to give it extra interest.

You may think that the SOUND command already looks like a Christmas tree since we added the ENV and ENT numbers. Here is positively the last thing you can tack onto it – the noise. Edit line 30 of our program to give you:

```
3Ø sound 1,478,18Ø,1,1,1,5
```

If you run the program now you will notice quite a difference. Try experimenting with the 31 different sorts of noise which can be specified by putting the appropriate number in this part of the command. Leaving it off or putting 0 means zero noise.

Exercises

Add tone and volume envelopes to the 'familiar tune' and then change them to get different sounds. Then add different sorts of noise to some of the sound commands.

Write your own program for a few bars of a familiar tune. Even if you don't know music very well, it is possible to do quite a lot by trial and error. If you do know about music, don't be too ambitious at first. A nursery rhyme is quite sufficient to start with.

Playtime

Well, it isn't really. The next program on the datacassette is called ORGAN. Not only can you pick out tunes directly from the keyboard but you have the opportunity to analyse the program to see how it can be done. You should be starting to understand enough about BASIC to list and understand what a program is supposed to do, although this particular program contains some rather advanced programming. Don't be afraid to try changing things to see what will happen.

As promised, here is the ZAPPOW listing:

```
10 REM Zappow
20 REM by Dave Atherton
30 MODE 1
35 INK 0,1:INK 1,25
40 LOCATE 13,4:PRINT"SOUND DEMO"
50 LOCATE 10,7:PRINT"1. Explosion"
60 LOCATE 10,8:PRINT"2. Dog Barking"
70 LOCATE 10,9:PRINT"3. Siren"
80 LOCATE 10,10:PRINT"4. Toilet Flush"
90 LOCATE 10,11:PRINT"5. Cuckoo"
100 LOCATE 10,12:PRINT"6. Machine Gun"
110 LOCATE 10,13:PRINT"7. Space Invader"
120 LOCATE 5,17:PRINT"Select a sound from 1 to 7"
130 IF INKEY$)"" THEN 130
140 a$=INKEY$:IF a$="" THEN 140
150 IF a$("1" OR a$)"7" THEN 140
160 a=VAL(a$)
170 LOCATE 20,19:PRINT a$
180 IF a=1 THEN GOSUB 280
190 IF a=2 THEN GOSUB 330
200 IF a=3 THEN GOSUB 380
210 IF a=4 THEN GOSUB 430
220 IF a=5 THEN GOSUB 480
230 IF a=6 THEN GOSUB 530
240 IF a=7 THEN GOSUB 580
250 FOR j=0 TO 1000:NEXT
260 LOCATE 20,19:PRINT " "
270 GOTO 130
```

(Continued)

```
280 REM Explosion
290 ENV 1,11,-1,25
300 ENT 1,9,49,5,9,-10,15
310 SOUND 1,145,255,0,1,1,12
320 RETURN
330 REM Dog Bark
340 ENV 1,4,7,10
350 ENT 1,7,-8,3,6,24,2
360 SOUND 1,120,33,8,1,1,3
370 RETURN
380 REM Siren
390 ENV 1,2,9,45
400 ENT 1,2,9,45
410 SOUND 1,150,90,6,1,1
420 RETURN
430 REM Toilet Flush
440 ENV 1,3,-2,85
450 ENT 1,5,-1,51
460 SOUND 1,150,254,11,1,1,8
470 RETURN
480 REM Cuckoo
490 ENV 1,4,12,11
500 ENT 1,5,12,8
510 SOUND 1,165,40,13,1,1
520 RETURN
530 REM Machine Gun
540 ENV 1,21,-5,4
550 ENT 1
```

(Continued)

```
560 SOUND 1,162,82,15,1,1,11
570 RETURN
580 REM Space Invader
590 ENV 1,4,30,19
600 ENT 1,9,49,5,1,-10,26
610 SOUND 1,136,68,15,1,1,0
620 RETURN
```

Testing

It would be quite surprising if you didn't have to keep checking back through this chapter to answer the questions in SAT10. Don't let this worry you. If you already knew all about the SOUND commands of the CPC 464, you wouldn't be reading this book.

11 NUMBER CRUNCHING

As we have said several times the CPC 464, like any other computer, is good at boring repetitive tasks. It is also good at crunching numbers. In fact it is the only thing it *is* good at. Everything we have seen is achieved by the CPC 464's internal processing of lots of numbers at incredible speed. Even the graphic and sound functions work this way.

If you aren't particularly good at maths this chapter won't be easy. But give it a try anyway. The simple arithmetic isn't too bad, and BASIC is fairly straightforward in the way it is presented. In any case the really difficult stuff is not covered in this part of the course.

BASIC arithmetic

The CPC 464 does its sums by using the four familiar arithmetic functions, but two of them have different symbols:

Function	*Meaning*	*BASIC*
+	add	+
−	subtract	−
×	multiply	*
÷	divide	/

Thus, sums written in BASIC look a bit different to what you may be used to.

In Chapter 4 we entered:

```
print 2+2
```

You can do the same thing with the following sums by referring to the list of functions above and using the correct BASIC symbols:

3 + 7	17 − 8
15 × 4	15 ÷ 3
8 ÷ 2	20 × 17

For more complicated sums we have to tell

the CPC 464 that certain operations have to be done before others. The following, for example, is perfectly clear in ordinary arithmetic:

$$\frac{15 \times 7}{7 - 2}$$

In BASIC this is *not* written:

```
15*7/7-2     [ = 13 wrong!]
```

We have to show that the multiplication 15*7 and the subtraction 7 – 2 must be carried out *before* the division. This is done by putting them between brackets as follows:

```
(15*7)/(7-2)     [ = 21 right!]
```

You should now be able to do the sums below:

$$\frac{30}{3 + 2} \qquad \frac{15 \times 6}{9}$$

$$24 - (4 \times 3) \qquad 5 \times (2 + 8)$$

The same rule applies to variables as well as numbers. Enter the following program and then keep changing line 40 so that you can solve the other sums shown below. Don't forget to use the correct symbols and to put brackets in the right place.

```
10 a=2
20 b=5
30 c=10
40 print a+b+c
```

$$a \times a \qquad \frac{b \times c}{b - a}$$

$$c - (a \times b)$$

$$\frac{c}{b - a} \qquad \frac{c - b}{c + b}$$

The first two are straightforward enough, but what did you think of the answers to the other three? You have just had an insight into the way the CPC 464 does its maths. If you count the digits before and after the decimal point for the last three answers, you will find that there are always nine. This is the way the CPC 464 normally displays numbers on the screen. The snag is, though, that sometimes a series of calculations will give a very slightly inaccurate result. Instead of 5.0 you could get:

```
5.0000001
```

or:

```
4.99999999
```

Tricky, isn't it. Fortunately we don't often need such extreme accuracy. Sometimes, in

ROUND

fact, we only want whole numbers, since the difference between 276.25 and 276, for example, is not often very significant. The keyword we can use to give a result as a whole number is ROUND. A typical command would be:

```
x=round(26*17)/1.6
```

Try it. If the part after the decimal point is less than 0.5 the result is rounded *down* to the next whole number, and if it is 0.5 or more, it is rounded *up* to the next whole number.

Another thing you must remember is that, although the CPC 464 puts on the screen numbers that are accurate to 9 digits, it may be holding even more digits in memory. For example, you may end up with a value of:

```
5.ØØØØØØØØ1
```
(ten digits)

The CPC 464 will insist that this is 5.0 every time you ask it! But, and this is important, if the next command is something like:

```
IF a=5 THEN GOTO ....
```

the GOTO will never be executed.

A standard programming technique to avoid such problems is never to use the line above but rather the following:

```
vsmall=Ø.ØØØØØØ1
IF ABS(a-5)<vsmall THEN GOTO ....
```

Don't worry about the keyword ABS. It just gives you the difference between 'a' and 5, as you will see in Part 2.

The next program on Datacassette A is a good crib for multiplication tables as well as being an example of simple maths on the CPC 464. Ask it to give you the 13.87 times table! After running this program it may be worth your while to list it on the monitor screen to see how it is put together.

Elementary logic

There are lots of things we mentioned in earlier chapters without explaining them in detail, the reason being quite simply that it was too soon to have explained them at that point and would have confused rather than helped you. One of these is the use of 'logical operators'.

Cast your mind back to our study of loops. We used signs like '<' and '>'. There is a complete list of these symbols, known as 'logical operators':

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

Even if these terms are unfamiliar, it is fairly clear what they mean. When you need to compare two variables so that you know when to end a loop or decide between two courses of action, these operators give you the means to discriminate between them. The IF commands we studied in Chapter 7 are entirely dependent on these operators.

A subtlety you must remember about logical operators is that they take into account whether a number is negative or positive, so if a = 5 and b = 3:

```
a>b
```

But if a = -5 and b = 3:

```
a<b
```

String logic

Do you remember that we started off saying that the CPC 464 only knew how to handle numbers? How then, you should be asking, can it store and process characters in string variables, as we learned in Chapter 6?

The answer is that each character is identified by a number called its 'character code', so characters are simply treated as a series of numbers. Thus, the logical operators can test strings for their alphabetic order, 'A' being less than 'Z', but this is because of the numeric value of the character code. If we try the following command:

```
IF "Apple"<"Orange" THEN PRINT "Lemon"
```

the answer is a 'lemon' since 'apple' is before 'orange' alphabetically. All capital letters are less than any lower-case letters, and a short string is less than a longer one that starts in the same way, so:

"aardvark" < "abbey"	is true
"love" > "locksmith"	is true
"box" < "BOX"	is false
"Zoo" > "Zookeeper"	is false
"apple" > "Apples"	is true

In this case, the CPC 464 is testing the 'numerical' value of each letter of a word and comparing it with the numerical value of the letter in the same position in the word on the other side of the operator.

Homes and gardens

The house is in pretty good shape now. It has a picture window on one end, a chimney, and a fence to keep the neighbour's dog out. All it needs now is a bit of work on the garden.

Being a practical sort of machine, the CPC 464 is not much use for planning an attractive flower border. But it can help you to plan the vegetable plot. So load the next program from Datacassette A. It's called GARDEN.

The garden is 6 metres long by 4 metres wide and we are going to plant things in 4-metre rows. Different vegetables need to be sown in rows of different widths and each will give a different weight of produce per square metre of cultivation. When you run the program it will ask you how many rows you want of each vegetable, and will tell you if you have any space left for another row or two.

Assuming a good soil, and a reasonable year for weather, you can then get a summary showing the weight of produce that can be expected for each vegetable. The listing for this program is shown below. As you can see, the maths are fairly simple, and are based on the following table:

Vegetable	*Row width (metres)*	*Produce per row (kg)*
Onions	0.30	9
Carrots	0.30	3
Potatoes	1.0	50
Cabbages	0.60	8
Beans	1.0	30
Parsnips	0.50	11

You don't like parsnips? Try altering this program to include something you *do* like.

```
10 REM Garden
20 MODE 1
30 INK 0,0:BORDER 0
40 INK 1,26
50 length=6
60 CLS
```

(Continued)

```
70 PRINT"GARDEN"
80 PRINT"Length remaining :";length;"metres"
90 PRINT"Which vegetable do you want to grow"
100 PRINT
110 PRINT"                width       yield       rows"
120 PRINT"1. ONIONS       0.3m         9Kg       ";onions
130 PRINT"2. CARROTS      0.3m         3Kg       ";carrots
140 PRINT"3. POTATOES       1m        50Kg       ";potatoes
150 PRINT"4. CABBAGES     0.6m         8Kg       ";cabbages
160 PRINT"5. BEANS          1m        30Kg       ";beans
170 PRINT"6. PARSNIPS     0.5m         1Kg       ";parsnips
180 PRINT
190 PRINT"Enter a number between 1 and 6"
200 PRINT"or 7 to show total output"
210 INPUT veg
220 IF veg<=0 OR veg>7 THEN GOTO 190
230 IF veg=7 THEN GOTO 320
240 IF veg=1 THEN GOSUB 470
250 IF veg=2 THEN GOSUB 540
260 IF veg=3 THEN GOSUB 610
270 IF veg=4 THEN GOSUB 680
280 IF veg=5 THEN GOSUB 750
290 IF veg=6 THEN GOSUB 820
300 PRINT "-----------------------------------"
310 GOTO 60
320 REM Summary
340 PRINT"SUMMARY"
350 PRINT"GARDEN OUTPUT IN KILOS"
```

(Continued)

```
370 PRINT
380 PRINT"Onions  :";onions*9;"Kg"
390 PRINT"Carrots :";carrots*3;"Kg"
400 PRINT"Potatoes:";potatoes*50;"Kg"
410 PRINT"Cabbages:";cabbages*8;"Kg"
420 PRINT"Beans   :";beans*30;"Kg"
430 PRINT"Parsnips:";parsnips*11;"Kg"
450 GOTO 450
460 :
470 REM Onions
480 PRINT"Onions"
490 rowwidth=0.3
500 GOSUB 890
510 onions=rows
520 RETURN
530 :
540 REM Carrots
550 PRINT"Carrots"
560 rowwidth=0.3:produce=3
570 GOSUB 890
580 carrots=rows
590 RETURN
600 :
610 REM Potatoes
620 PRINT"Potatoes"
630 rowwidth=1
640 GOSUB 890
650 potatoes=rows
```

(Continued)

```
660 RETURN
670 :
680 REM Cabbages
690 PRINT"Cabbages"
700 rowwidth=0.6
710 GOSUB 890
720 cabbages=rows
730 RETURN
740 :
750 REM Beans
760 PRINT"Beans"
770 rowwidth=1
780 GOSUB 890
790 beans=rows
800 RETURN
810 :
820 REM Parsnips
830 PRINT"Parsnips"
840 rowwidth=0.5
850 GOSUB 890
860 parsnips=rows
870 RETURN
880 :
890 REM Details
900 INPUT"How many rows do you want to plant";rows
910 testlength=length-rows*rowwidth
920 IF testlength<0 THEN PRINT"No room":GOTO 900
930 length=testlength
950 RETURN
```

Testing

You really must run SAT11 to get a lot of practice in handling the mathematical operators of the CPC 464. Most programs require a certain amount of elementary maths for them to be of any use, so it is worthwhile spending as much time as possible in understanding this aspect of the CPC 464. Don't forget that most of the game programs you can buy for the CPC 464 depend on some very high-speed maths!

12 PLAYING GAMES

Most games, and not only computer games, involve pitting your physical and mental ability against opponents, against chance occurrences, or against the clock. Sometimes you have to play against all three. A chess program or an arcade game will give you many hours of pleasure in this way and you don't need to know anything about games programming.

If you want to *design* games, however, a word of advice is necessary. Go back to your maths books. You cannot expect to program a ball bouncing off a wall if you can't remember the formulae for calculating ricochet angle, speed and trajectory. Admittedly we aren't going to cover this level of detail at this stage of the course, but you must realise that the two games given in this chapter cover only a tiny fraction of what the CPC 464 can really do.

RND

Random events

When you played BOMBER back in Chapter 4, the alien spaceship kept appearing at different places in the bombsight. This was done by making use of a function known as RND (for RaNDom), which is useful for bringing an element of chance into a computer program.

The following is an example of the way RND can be used:

```
move rnd*639,rnd*399
```

RND gives you a decimal number between 0 and 1, so you have to multiply it by the largest number you expect in the command or routine. In the case above, the graphics cursor will end up in a random position on the screen since we multiplied the maximum x and y coordinates by RND.

Try writing a short program to put a small square in a random position on the screen.

Time out

Another useful function in the CPC 464, and not only for games, is TIME. From the moment that the CPC 464 is switched on or restarted, it counts the time elapsed every three-hundredth of a second and stores it in TIME. This count is only suspended when a program is loaded or saved on the datacassette. The following program is an example of how this can be used:

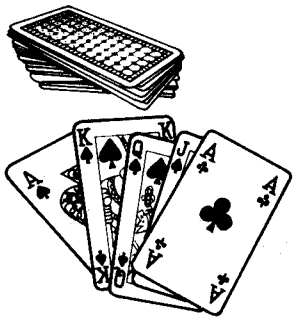
```
1Ø print "Press any key"
2Ø if inkey$="" goto 2Ø
3Ø for t=1 to rnd*5ØØØ:next
4Ø a=time
5Ø print "again"
6Ø if inkey$="" goto 6Ø
7Ø b=time
8Ø print "Reaction time=";(b-a)/3ØØ;" seconds"
9Ø end
```

What we are doing here is to sample the value of TIME before and after the word 'again' appears on the screen, the reaction time being the difference between the two.

You will have noticed another new keyword in lines 20 and 60, INKEY$. It is similar to INPUT except that it gives you only one character and it doesn't need to be followed by ENTER. Here we are using it to detect when *any* key is pressed - any key, that is, except SHIFT, CTRL, CAPS LOCK, and ESC.

TIME

INKEY$



BLACKJACK

Playing cards with the CPC 464 is more fun than playing solitaire all by yourself. The next program on Datacassette A is BLACKJACK. In case you don't know the game, the idea is to keep asking for cards until their total value is as close as possible to 21. If you get more than 21 you have 'bust' and lose the hand. If not, the CPC 464 then deals itself a hand and tries to get a higher score than you without 'busting'.

Other rules are:

- Five cards that add up to 21 or less will beat anything
- Aces can be counted as either 1 or 11
- Picture cards count as 10

Although there are several commands that we will not study until Part 2 of this course, the listing for BLACKJACK is shown below for you to study.

```
10 REM ** BLACKJACK **
20 REM
30 REM ** STARTUP **
40 MODE 1:BORDER 4
50 INK 0,17:INK 1,0
60 LOCATE 16,5:PRINT "BLACKJACK"
70 LOCATE 10,12
80 PRINT"Press a key to start"
90 suit$=CHR$(226)+CHR$(227)+CHR$(228)+CHR$(229)
100 card$="A23456789TJQK"
110 C5$="5 card trick - I win"
120 myace=0:yourace=0
130 mygames=0:yourgames=0
140 WHILE INKEY$="":WEND
```

(Continued)

```
150 CLS
160 REM ** YOUR TURN **
170 yourcards=0:yourace=0
180 yourhand=0
190 LOCATE 20,1
200 PRINT"Games Me:";mygames;
210 PRINT "You:";yourgames
220 y=20:x=5
230 yourcards=0
240 GOSUB 770
250 yourcards=yourcards+1
260 IF value=1 THEN yourace=yourace+1
270 yourhand=yourhand+value
280 GOSUB 830:x=x+5
290 IF yourhand)21 THEN GOTO 690
300 GOSUB 930
310 oneacehand=yourhand
320 IF yourace)=1 THEN oneacehand=yourhand+10
350 IF yourcards=5 THEN 440
360 IF yourhand=21 THEN 440
370 IF oneacehand=21  THEN 420
380 IF yourace=0 AND yourhand(=11 THEN 240
390 IF yourcards=1 THEN 240
400 INPUT"Want another card (Y/N)";q$
410 IF UPPER$(q$)="Y" THEN GOTO 240
420 IF oneacehand(=21 THEN yourhand=oneacehand
440 REM ** MY TURN **
450 y=10:x=5
```

(Continued)

```
460 myhand=0:mycards=0:myace=0
470 GOSUB 770
480 mycards=mycards+1
490 IF value=1 THEN myace=myace+1
500 myhand=myhand+value
510 GOSUB 830:x=x+5
520 FOR delay=0 TO 1000:NEXT
530 IF myhand)21 GOTO 720
540 IF mycards=5 THEN GOSUB 930:PRINT C5$:GOTO 710
550 IF yourcards=5 THEN 470
560 mineA=myhand
570 IF myace)=1 AND myhand(12 THEN mineA=myhand+10
600 IF myhand)=yourhand THEN 640
610 IF mineA)=yourhand THEN myhand=mineA:GOTO 640
630 GOTO 470
640 REM ** TEST RESULTS **
650 GOSUB 930
660 PRINT"I have";myhand;
670 PRINT"and you have";yourhand
680 IF myhand(yourhand GOTO 730 ELSE GOTO 700
690 GOSUB 930:PRINT"You have bust!"
700 PRINT"I win"
710 mygames=mygames+1:GOTO 140
720 GOSUB 930:PRINT"I have bust!"
730 PRINT"You win"
740 yourgames=yourgames+1:GOTO 140
750 END
760 REM ** GENERATE CARD **
```

(Continued)

```
770 card=INT(RND*13)+1
780 suit=INT(RND*4)+1
790 value=card
800 IF value)10 THEN value=10
810 RETURN
820 REM ** PRINT CARD **
830 LOCATE x,y
840 PRINT CHR$(24);"    ";CHR$(24)
850 LOCATE x,y+1
860 PRINT CHR$(24);" ";
870 PRINT MID$(card$,card,1);
880 PRINT MID$(suit$,suit,1);
890 PRINT" ";CHR$(24)
900 LOCATE x,y+2
910 PRINT CHR$(24);"    ";CHR$(24)
920 RETURN
930 LOCATE 1,24:PRINT SPACE$(40)
940 LOCATE 1,24:RETURN
```

Simple Simon

CHR$

As you can see from the listing below, it isn't as simple as all that! Again, there are some commands that will have to wait until Part 2 of this course. There is one keyword in it, however, that you may like to know about; it is CHR$.

Way back in Chapter 3 we saw that we can get some fancy special characters on the screen that don't appear on the keytops. CHR$ (for CHaracteR$) allows you to call them up by their character code. For example, to put a little spaceship on the screen the command would be:

```
print chr$(239)
```

The full list of CPC 464 characters is in the *Amstrad CPC 464 User Guide*.

```
10 cr$=CHR$(13)
20 REM Simon
30 REM **** INSTRUCTIONS ****
40 MODE 1:BORDER 20:INK 0,20:INK 1,1
50 LOCATE 16,2:PRINT CHR$(24);"Simon";CHR$(24)
60 PRINT:PRINT
70 PRINT"In this game, you have to watch the"
80 PRINT"flashing  circles  and remember the"
90 PRINT"pattern. When the sequence ends you"
100 PRINT"must copy it out on the cursor keys"
110 PRINT"The sequence increases by one after"
120 PRINT"each correct attempt.":PRINT
130 PRINT"For example, a circle at the top of"
140 PRINT"the  screen  should be indicated by"
150 PRINT"the up cursor.  The cursor keys are"
160 PRINT"above the numeric key pad,  and are"
170 PRINT"marked as follows:":PRINT
```

(Continued)

```
180 PRINT TAB(20);CHR$(240)
190 PRINT TAB(19);CHR$(242);" ";CHR$(243)
200 PRINT TAB(20);CHR$(241)
210 LOCATE 7,22:PRINT"Press ENTER to continue"
220 LOCATE 5,24:PRINT "there will be a short pause!"
230 WHILE INKEY$()cr$:WEND
240 REM **** SET-UP ****
250 MODE 0
260 WINDOW 7,14,10,16
270 b=17:f=3:REM Background/Foreground
280 BORDER b
290 INK 0,17
300 FOR i=1 TO 15:INK i,b:NEXT
310 x=320:y=70:c=2:GOSUB 940
320 y=330:c=1:GOSUB 940
330 x=120:y=200:c=3:GOSUB 940
340 x=520:c=4:GOSUB 940
350 INK 5,f:PEN 5
360 RANDOMIZE TIME
370 a$=""
380 REM **** DISPLAY SEQUENCE ****
390 a$=a$+CHR$(RND*3+1)
400 FOR i=1 TO LEN(a$)
410 FOR j=1 TO 200:NEXT
420 x=ASC(MID$(a$,i,1))
430 INK x,2*x+1
440 SOUND 1,10+x*100
450 FOR j=1 TO 200:NEXT
```

(Continued)

```
460 INK x,b
470 NEXT
480 FOR i=1 TO 100:NEXT
490 REM **** GET ANSWER ****
500 FOR i=1 TO LEN(a$)
510 WHILE k$)"":k$=INKEY$:WEND
520 FOR L=1 TO 2000:k$=INKEY$
530 IF k$)"" THEN 560
540 NEXT
550 k$=" "
560 k=ASC(k$)-239:IF k(1 OR k)4 THEN 520
570 x=ASC(MID$(a$,i,1))
580 IF k(>x GOTO 730:REM wrong
590 INK x,2*x+1
600 SOUND 1,10+x*100
610 FOR j=1 TO 80:NEXT
620 INK x,b
630 FOR j=1 TO 20:NEXT
640 NEXT
650 REM **** RIGHT! ****
660 CLS:PRINT" RIGHT!"
670 PRINT:PRINT:PRINT" SCORE:"
680 PRINT:PRINT"  ";LEN(A$)
690 FOR j=1 TO 600:NEXT
700 LOCATE 1,1:PRINT"          "
710 GOTO 390
720 REM **** WRONG ****
730 SOUND 1,2000
```

(Continued)

```
740 CLS:PRINT"  Wrong"
750 FOR j=1 TO 300:NEXT
760 PRINT:PRINT"Sequence   was:"
770 FOR i=1 TO LEN(a$)
780 x=ASC(MID$(a$,i,1))
790 INK x,2*x+1
800 SOUND 1,10+x*100
810 FOR j=1 TO 200:NEXT
820 INK x,b
830 FOR j=1 TO 200:NEXT
840 NEXT
850 REM **** END & RESTART ****
860 CLS
870 PRINT"  You"
880 PRINT" scored"
890 PRINT:PRINT"  ";LEN(a$)
900 PRINT:PRINT" PRESS"
910 PRINT" ENTER"
920 WHILE INKEY$<>CR$:WEND
930 GOTO 360
940 REM **** CIRCLES ****
950 r=60
960 FOR i=-r TO r STEP 2
970 h=SQR(r*r-i*i)
980 MOVE x-h,i+y:DRAW x+h,i+y,c
990 NEXT
1000 RETURN
```

Testing

Before you go on to Part 2 of this course, *More BASIC*, get in as much practice as possible in writing your programs, and go through the last of our tests, SAT12. It will give you questions on *all* the chapters of this part of the course and will show you if you need to go back over any topic.

Good Luck!

LIST OF KEYWORDS

The following is a list, chapter by chapter, of all the Amstrad BASIC keywords covered in this book. Not all the variations or extensions have been dealt with since this is, after all, a book for beginners. Part 2 of this course, *More BASIC,* covers further keywords and more advanced programming techniques.

A description of all keywords can be found in the *Amstrad CPC 464 User Guide.*

Chapter 2

RUN
LOAD

Chapter 3

CLS
RUN

Chapter 4

BORDER
MODE
CAT

Chapter 5

CLG
INK
DRAW
LIST
MOVE
NEW
PAPER

PLOT
REM

Chapter 6

INPUT
LET
LOCATE
PRINT
SAVE

Chapter 7

CONT
EDIT
ELSE
GOTO
IF
STOP
THEN

Chapter 8

DRAWR
END
FOR
GOSUB
MOVER
NEXT
PLOTR
RETURN
STEP

Chapter 10

ENT
ENV
SOUND

Chapter 11

ROUND

Chapter 12

CHR$
INKEY$
RND
TIME

LIST OF PROGRAMS

Datacassette A contains the following programs in the same order that they are referred to in this book. Datacassette B contains the Self-assessment Tests (SATs) which the reader should complete at the end of every chapter except Chapters 1 and 9.

Chapter 2

HELLO
SIMON (See also Chapter 12)

Chapter 3

LETTERS
REPEAT NAME
KEYBOARD
HANGMAN

Chapter 4

DRAW
COORGEOM
BOMBER

Chapter 5

HOUSE

Chapter 6

BARCHART
BUZZWORD

Chapter 7

DECO

Chapter 8

MANSION

Chapter 10

ZAPPOW
ORGAN

Chapter 11

MULT TABLES
GARDEN

Chapter 12

BLACKJACK
PONTOON
SIMON

INDEX