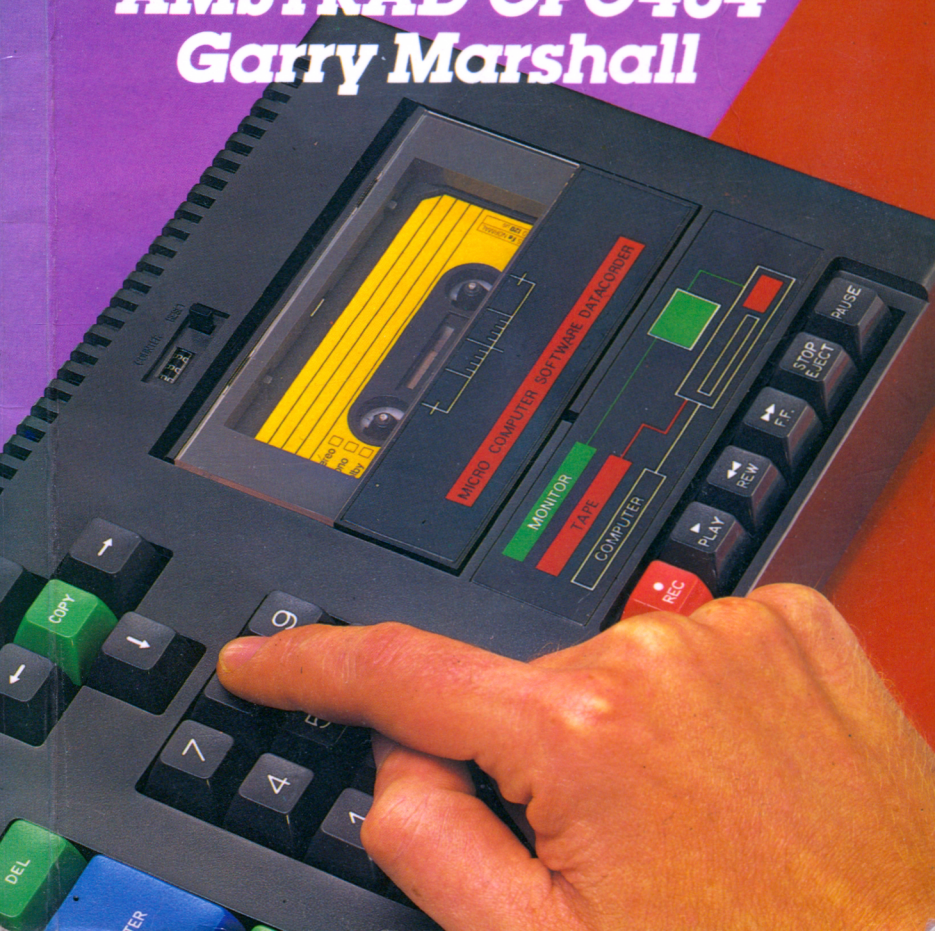
MADE EASY

ARROW

USING YOUR AMSTRAD CPC464

Garry Marshall



Using your Amstrad CPC464 Made Easy

Garry Marshall

Arrow Books

Using your Amstrad CPC464 Made Easy
Garry Marshall

Arrow Books Limited
17-21 Conway Street, London W1P 6JD
An imprint of the Hutchinson Publishing Group
London Melbourne Sydney Auckland
Johannesburg and agencies throughout
the world

First published 1984

Set in 12pt Stymie medium

Printed and bound in Great Britain
by Commercial Colour Press, London E7

ISBN 0 09 938800 6

CONTENTS

LIST OF FIGURES

Foreword

The main aim of this book is to provide, in the easiest terms possible, an introduction to the Amstrad CPC464. After guiding you through the process of getting started, it gives an appreciation of the uses to which the Amstrad CPC464 can be put and then an explanation of how it can be made to carry out these tasks. In this way it provides answers to the key questions "How do I use it?" and "What can I use it for?"

The book has three parts. The first is very much an introduction to the computer. It deals with how to get the computer ready for use, how to tell it what to do, and the kinds of things that you can tell it to do. It also describes other items that can be connected to the computer and used in conjunction with it. The Amstrad CPC464 has a number of sockets where various things can be plugged into it, and is well equipped in this way to provide the basis of an expanding system.

The second part of the book deals with how to tell the computer what to do using its own language, that is, how to program the Amstrad CPC464 in BASIC. The third part explores the uses to which the computer can be put by making use of programs that have already been written by someone else. If a program is already available to make the computer do something that you want it to do, then it is only sensible to make use of it rather than writing a similar program from scratch yourself. These two parts correspond to the two main ways of using the computer. Either you can tell it what to do youself, which is interesting, challenging and edu-

cational, but time consuming. Or you can use a program that already exists to make the computer carry out a particular task that you need.

Whichever way you choose to use the Amstrad, and there is room for both ways, you will find that it is an acquisition that is of immense value. It can be used for profit, for education and for entertainment, and can come to play a part in many different aspects of life. This book can help to get you started and then guide you along the road to using the Amstrad to its fullest.

About the Author

Garry Marshall is Principal Lecturer in the Department of Electronic and Communications Engineering at the Polytechnic of North London. He is a regular contributor to computer magazines, including Computing Today and devises a computer puzzle each week for The Observer. He has written 9 books, the majority on computing, including Learning to Use the PET (Gower), Programming with Graphics (Granada) and Programming Languages for Micros (Newnes).

Chapter 1

Introduction to the Amstrad

The Amstrad is, of course, a personal computer. Outwardly, it has the appearance of a typewriter keyboard, although it has a few extra keys that an ordinary typewriter does not. When it is ready to use, the Amstrad will do nothing until you tell it to or, in some other way, make it. This is because there is almost no limit to the different things that the Amstrad can do, and so it must be told what it is to do before it can be made to do it. Like any other computer, the real strength of the Amstrad lies in its versatility. It is not like a kettle, which can only heat water, or a washing machine which can only perform its simple function. It has many uses.

A consequence of the computer's versatility is that it is not quite as easy to use a computer as it is a kettle or a washing machine. It is necessary to know *how* to tell a computer what it is that you want it to do. It is also necessary to know *what* the computer itself is capable of doing, particularly if you want to take full advantage of it and to be sure that it is being used for all the things that it is capable of doing for you. This means not only that we must be aware of the computers' capabilities but also that we must know how to communicate our needs to it.

Communicating with the Amstrad

You communicate with the Amstrad by using its keyboard. Although it is obviously not necessary to be a skilled typist to key in the short words that are the most

commonly used commands, you will find that as you progress with the use of the computer there is more and more to type and that if you can type 'properly' it will take much less time than if you use one finger 'hunt and peck' keying. So you communicate with the computer by typing at its keyboard and, correspondingly, it communicates with you by showing you what it is doing on the television screen.

Also, when you tell the computer what to do, you cannot use just any words that you choose as you could if you were telling another person what to do. The meaning of "Pass it to me" would, in most circumstances, be quite clear to the person being addressed but to a computer, with no knowledge of the context in which the command was given, the meaning 'it' would not be apparent, and so the command would not be understood and, therefore, could not be obeyed. For just this kind of reason, the instructions that are given to a computer must be expressed in its own language. They will then have absolutely clear and precise meanings to the computer and it will be able to obey them without fail. The computers' own language is known as BASIC. (It stands for Beginners All-purpose Symbolic Instruction Code). And if the prospect of learning another language is off-putting, remember that you have learnt at least one already and that BASIC is really only a small part of one of them (English) that is specially adapted for telling computers to do the kinds of things that they can do.

What the Amstrad can do

So what *can* the Amstrad CPC464 do? In essence it can store and process information. It can also communicate information, but to do this it must first be connected to the item it is to communicate with or to a means of communication such as the telephone network. By

itself, then, the Amstrad can store and process information, but this begs the question of what information is. Information may be numbers or letters. It can also be special symbols or strings of letters, that is, words or sentences. In fact, as far as the Amstrad is concerned, information is anything that can be typed at its keyboard.

Some examples may help to illustrate the forms that information can take and the ways in which it is processed. A word processing program accepts and stores text typed at the keyboard, and so in this case the information that is stored is words, sentences and paragraphs, although the basic elements are letters and punctuation marks. Once it is stored the information can be processed in any way that the word processor permits. Most word processors can arrange the text in lines of a specified length, make the right hand edge of the text neat and tidy by carrying the last word on each line to finish precisely at the end of the line (this is done by putting extra spaces between the words of each line to push the end of the line across to the necessary position) and arrange the presentation of the end of each paragraph by leaving a specific number of blank lines between one paragraph and the next and indenting the beginning of the new paragraph. As a second example, a database program can store information so that it can be retrieved again in much the same way as it can in a filing system. A company may use it to keep the records of all its transactions and an individual can use it to keep the details of all the videotapes in his or her collection. In this case the information can consist of words, for the title of a videotape, and numbers, for the value of a transaction. The information can be processed by sorting all the records into a particular order or to select all the records meeting a particular criterion. To give a third example, a program for an adventure game must

contain descriptions of the various places that can be visited during the game, and these must be stored so that a picture of each can be displayed complete with its treasure and its hazards when it is visited during the course of the game. In this case, the information consists of the descriptions which, in turn, are composed of letters and numbers.

Applications for the Amstrad

The consideration of information has, incidentally, brought out some typical examples of the uses to which the Amstrad can be put: games, for entertainment and relaxation, word processing for anyone who has to deal with words, whether in writing lettters, producing documents or in writing a book, and databases for any application requiring the storage and retrieval of information. It is notable in all these cases that the applications are direct developments of previous practice and that using a computer for them brings with it definite improvements. Computer games are much more attractive and compulsive than their non-computer predecessors. Among the reasons for this are that the computer games are faster moving, more colourful and more realistic than their predecessors. The word processor is the development of the typewriter that has been made possible by the new technology. It makes many aspects of document production much easier than before, including the correction of mistakes, the presentation of documents and the communication of documents. Database programs show to considerable advantage over conventional filing systems. They make the retrieval of information much quicker and easier, particularly when large amounts are involved. There are also slightly unexpected advantages. For example, when recording the details of a company's transactions, the preferred order in which records are stored will be

different for different people. The accountant will want one order, perhaps the most profitable first and the least profitable last. A Salesman will want a different order, perhaps just his clients in the order that he deals with them. It is clear that in a conventional filing system the papers can only be stored in one order. But when the records are stored in a computer it can rapidly sort them into the particular order that best suits the current users.

Other applications for the computer include planning with the use of a special program called a *spreadsheet*; education, using the computer to present or to allow the exploration of a body of knowledge; and problem solving in science and engineering.

Should you write your own programs or buy them?

But no matter what use the computer is put to, it must be instructed as to how to behave. If it is used to solve a problem it must be told how to find the solution: if it is to process words it must be told how to do that. In all these cases, the computer must be given a program, that is a set of instructions that tell it how to perform the task in question. When the program is *run* by the computer it automatically carries out all the instructions in the program it has been given one by one. When it has carried out all the instructions in the program it will inevitably have completed the task. Its ability to be programmed is the secret of the computers' versatility. Giving it one program enables it to do one thing. Subsequently giving it another program can make it do another, perhaps quite different, task.

If you know exactly what you want your Amstrad to do, you can write a program in BASIC to tell it how to do the task as soon as you are familiar with BASIC. On the other hand, if you can find a program already written by someone else that makes the Amstrad do what you want

it to then you can buy the program. Programs specially written for the Amstrad are supplied by Amsoft. This shows that you do not have to be able to write programs in BASIC to be able to use your Amstrad. But if you can write programs you will naturally acquire more familiarisation with the Amstrad and with its capabilities.

Undoubtedly, the only way to get the computer to do *exactly* what you want is to write your own programs. Besides making the computer a really personal tool, this activity proves for many a rewarding and absorbing matter. With a lesser knowledge of BASIC than is necessary to write reasonably sophisticated programs it is possible to amend other peoples' programs so that they approach more closely what you want of them. With no knowledge of BASIC you can only use other peoples' programs to tell your computer what to do. Any owner of an Amstrad wanting to learn about his computer should learn to program it in BASIC. But it can also be used to advantage with general purpose programs that can be bought from suppliers such as Amsoft for applications such as word processing and information storage. Both a knowledge of BASIC and some commercial programs are needed so that you can enjoy, and employ, your Amstrad to the full.

Summary

The Amstrad is a small, rather powerful personal computer. Like any other personal computer, it is extremely versatile and capable of achieving many different tasks. To make it carry out a particular task it must be given a program, that is, a list of instructions, that tells it how to do that task. But when one task is completed it can be given another program to enable it to carry out another, perhaps quite different, task. Its programs must be written in a special computer language called BASIC. By mastering BASIC, the users of the

Amstrad can write programs to make the computer perform any task that they would like done. Programs for specific tasks can also be brought. But whether the Amstrad runs the users' own programs or purchased programs, it can be used for many activities from games through education to business applications such as word processing and the storage and retrieval of information.

Chapter 2

Getting started

This chapter deals with all the matters that you need to know about how to get started with the Amstrad. Actually getting the Amstrad ready to use in the first place is just about as easy as it is with a television set, consisting of little more than plugging it in. After describing how to prepare the Amstrad for use we examine the keyboard and the purposes of the various keys on it. Then a few of the more common commands for the Amstrad are introduced for, as we know, the Amstrad does nothing until we tell it what it is that we want it to do. Finally, we examine the various items that can be plugged into the sockets at the back of the computer, thereby expanding the capabilities of the computer and enabling it to become the heart of a quite large system.

Getting the Amstrad ready for use

The Amstrad is much simpler than most computers to set up for use. The monitor should be plugged into the mains. You will notice that there are two leads coming out of the front of the monitor. One of these is the power lead and the other takes signals from the computer to the monitor so that a display appears on the screen. Plug the two leads into the computer. It is quite clear, if you look at the back of the computer, where they plug in, as there are only two sockets of the right size. Now turn on the monitor by pressing the button at the bottom right hand corner, and the computer by sliding the

switch at the back right hand end. A message should now appear on the screen. This is shown as Figure 2.1. The properly connected Amstrad should now appear as in Figure 2.2.

The meaning of the various items that appear on the screen in the initial display is as follows.

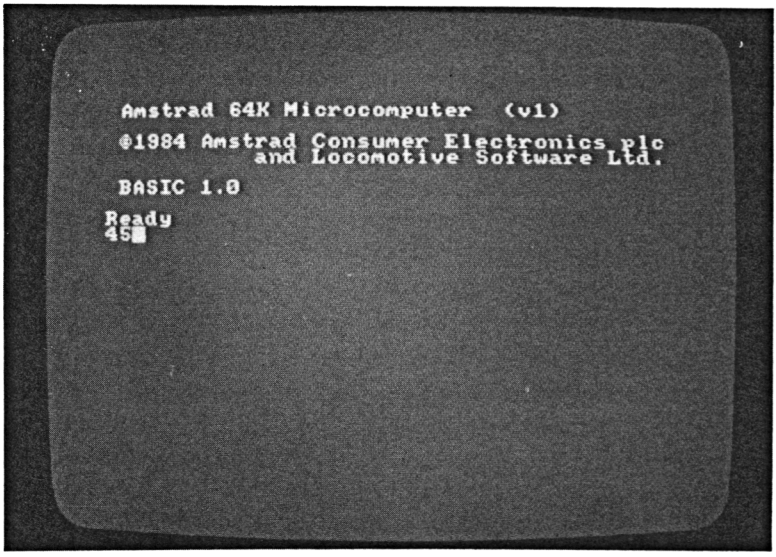


Figure 2.1 The Amstrad initial display.

Amstrad 64K Microcomputer (v1)

The name of your new computer. The next two lines are the copyright and name of the manufacturer.

BASIC 1.0

The language used by this model of the computer. This identifies the version of BASIC that the Amstrad provides to its users for writing programs.

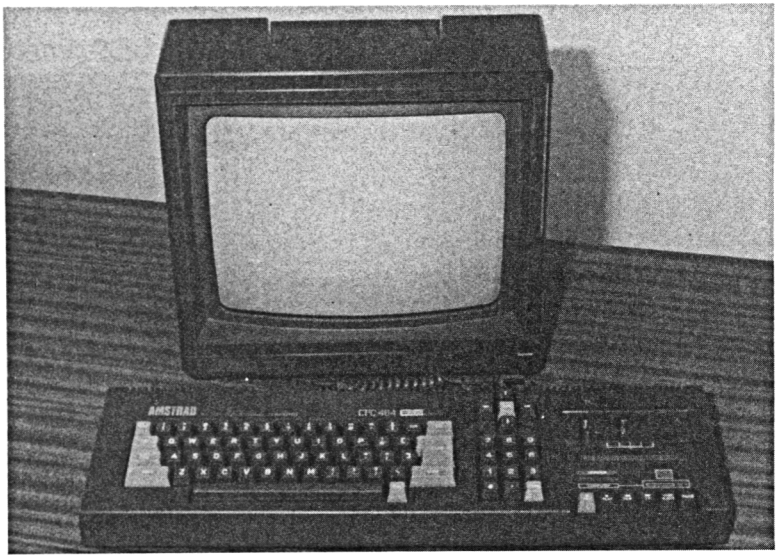


Figure 2.2 The Amstrad properly connected and ready to use.

Ready indicates that the Amstrad is ready to accept a communication from its user, perhaps in the form of a command or a program.

■ The solid square is known as the *cursor*. It indicates, by its position, where the next character to be typed at the keyboard will appear on the screen.

The keyboard

The keyboard of the Amstrad has 74 keys, including the long space bar at the bottom of the keyboard. Most of the keys are black, but some are green, one red and two blue, the distinction between the sets being that the black keys cause a character to be displayed on the screen while, in general, the green keys are used to change the effect of pressing a black key. The red key will stop the computer if pressed once (to restart press another key) and if pressed twice will "BREAK" the program. The blue keys tell the computer that you have finished writing a message, which should now be "entered" into the computer memory. The keys are arranged as shown in Figure 2.3, with the black keys set in rows in the same way as on a typewriter keyboard, in what is known as the QWERTY layout after the first few keys along the top row of letters. The keys all have a full movement. It is quite possible for a trained typist to type at a rapid rate on the keyboard.

When used, the keyboard gives essentially the same effect as a typewriter keyboard, but with some differences. Considering the black keys first, keys marked with just a letter will initially produce the corresponding letter at the position of the cursor on the television screen. Most other black keys have two symbols marked on them, and pressing any of these keys will place the lower of the two symbols on the screen. To give an example, pressing the key marked with a figure eight below a bracket will cause the eight to be displayed. Turning now to the first of the green keys, holding down one of the SHIFT keys which are located at either end of the second row of keys from the bottom of the keyboard, and then pressing one of the keys marked with two symbols will cause the upper symbol to appear on the screen. So holding down a SHIFT key and then pressing the key marked with an eight and a bracket will cause

Figure 2.3 The Amstrad's keyboard.

the bracket to be displayed. Hold down a SHIFT key and press a letter key, this will create a capital letter. If you want to use capitals all the time you can press CAPS LOCK. This will make all the letters appear in upper case, but will still display the lower item of two item keys. That is, if you press '8' an eight will still appear. If you now press CTRL and CAPS LOCK, this will work in the same way as SHIFT LOCK, that is, the letters will be in upper case, but the key marked '8' will produce the other character on the key, that is the bracket.

The key marked CTRL is referred to as the control key. If you press it and CAPS LOCK together again the

board will return to normal. This is typical of the way that the control key is used to cause the computer to switch from one mode of operation to another, and then back again.

At this stage, we know how to type capital and small letters, numbers, punctuation marks and any of the other symbols that appear on the keys. Pressing the long bar at the bottom of the keyboard gives a space. Just by pressing the appropriate keys we can make any character appear on the screen. No matter what symbol we type, it appears at the position marked by the cursor and then the cursor moves one position to the right ready to position the next character to be typed. When a line on the screen has been filled, the cursor automatically moves to the beginning of the next line so that the next character to be typed is placed there. By typing some text you can watch the lines that you type build up on the screen. Since all the keys repeat, merely holding down a key will quickly fill a line or more with the character marked on that key. But the Amstrad will not accept anything you type that consists of 256 or more characters. If you go on to key the 256th character, the Amstrad will make a beep. You will also find that when you are typing on the bottom line of the screen and reach its end, the contents of the screen all automatically move up by one line, or 'scroll' upwards, to leave the cursor at the beginning of a blank bottom line. The previous top line vanishes. So, when typing at the Amstrad's keyboard, characters appear on the screen in essentially the same way as they would appear on paper when using a typewriter. In fact, the Amstrad makes it rather easier because there is no need to perform the equivalent of the typewriter's carriage return to start on a new line, or to change the paper when a page is full.

The functions of the keys that have not been mentioned so far are as follows.

The keys marked with arrows. These keys, positioned in pairs on either side of the space bar, are for moving the cursor around the screen. They move it in the directions marked on the keys so that text can be positioned anywhere on the screen by moving the cursor to the position at the beginning of the text by using these keys before typing the text itself.

ENTER. This key terminates a line of text and at the same time sends it to the computer for the computer to act on. If, for example, a command has been typed then pressing ENTER will send it to the computer which will then obey it.

DEL. This key deletes the symbol that has just been typed and, more generally, the key to the left of the cursor's position. It allows typing errors to be corrected as soon as they are made.

CTRL. We have seen that holding down CTRL and pressing CAPS LOCK causes the Amstrad to switch between giving capital letters and small letters. By contrast, holding down CTRL and pressing P always causes a 'beep'.

COPY. If shift is pressed the cursor can be moved using the arrow keys, but this is a second cursor called COPY CURSOR. When it has reached a position you wish to COPY, press the COPY key and a copy will appear at the position of the original cursor. By stopping the copy and typing in new letters you can then EDIT your program.

NUMBERS. A second set of number keys is provided for your convenience.

Giving commands to the Amstrad

We give commands to the Amstrad just by typing the correct words, and the Amstrad will obey them as soon as we press ENTER. If you issue a command the computer cannot understand, it will say

Syntax error

The first command we will introduce is for clearing the screen so that we can start work in a position equivalent to that of someone starting to write with a clean sheet of paper. To clear the screen, all that is necessary is to type

CLS

and press ENTER. Try it!

Since we have already seen that computers store information, process it and display the results, let us see next how we can tell the computer to do these things. We can instruct the computer to store an item in its memory by using a command starting with the word 'LET'. Then we write a name, which can be any name we choose as long as it starts with a letter and consists of letters and numbers only. This will be the name under which the item is stored. Then we write an equals sign (=) and finally the item that is to be stored.

To store the number 2.5 under the name 'number', we write

LET NUMBER = 2.5

and as soon as we press ENTER it is done. The word 'let' can be omitted, so that we can write just

```
NUMBER = 2.5
```

but for the moment we will retain 'LET' to show consistency with the other commands that we can give, all of which have a characteristic key word associated with them.

A word, that is, a string of letters, or indeed any sequence of characters, can be stored in nearly the same way. The difference is that we must tell the computer that we are storing something that is not a number, and we do this by placing a dollar sign ($) at the end of the name we choose. Additionally, the word to be stored must be enclosed in quotation marks. By doing these things we can store the word 'AMSTRAD' under the name A$ by

```
LET A$ = "AMSTRAD"
```

or, as before, just by

```
A$ = "AMSTRAD"
```

A complete sentence can be stored just as easily by

```
LET SENTENCE$ = "LET'S USE THE AMSTRAD"
```

or

```
SENTENCE$ = "LET'S USE THE AMSTRAD"
```

If we have a whole number to store, we can store it in the same way as any other number, but it takes up less space in memory if we store it under a name that ends with a percentage sign (%). So, we could write

```
LET TWO = 2
```

but it is better to have

LET TWO % = 2

SUMMARY BOX 1

Storing information

LET name = item

1. The effect of the command is to store the item of information 'item' under the name 'name'.
2. LET may be omitted.
3. 'name' must start with a letter and then can be followed by any letters or numbers.
   It must end with $ if anything other than a number is being stored.
   It should end with % if a whole number is being stored.
4. 'item' can be a number or a string of characters enclosed in quotation marks.

Now that we know how to store items of information in the computer's memory, we can look at how to process them. In its simplest forms, processing can also be done with the LET command. If we consider numbers first, we can write arithmetic expressions exactly as we would in ordinary arithmetic, and using brackets, except that the usual multiplication sign is replaced by an asterisk (*) and the usual division sign by a slash (/). And on the right hand side of the equals sign in a LET command, we can write an arithmetic expression rather than just a number. The computer obeys such a command by finding the value of the arithmetic expression and storing it under the given name. If we type

```
LET X = 2 * 3 + 1.5
```

and press ENTER, the computer will calculate 7.5 as the value of the arithmetic expression on the right and store this number under X, the name given on the left. After this command, typing

```
LET Y = X - 1.1
```

causes 6.4 to be stored under Y because the value of the expression on the right is the number stored under X less 1.1, and this is stored under the name given on the left.

SUMMARY BOX 2

Dealing with numbers

LET name = arithmetic expression

1. **The effect of the command is to find the value of 'arithmetic expression' and to store it under 'name'.**
2. **LET may be omitted.**
3. **'name' is any name under which a number may be stored.**
4. **'arithmetic expression' is any properly written arithmetic expression that involves names under which numbers are stored, numbers and the arithmetic operators +, −, *, /.**

Words and character strings can also be processed using the LET command, and we will give one simple example of how this can be done. The plus sign can be used with words, as well as with numbers, although in this case it takes a very different meaning from its usual arithmetic one. The result of 'adding' one character

string to another is a single character string consisting of the first string followed at once by the second. Thus, after

```
LET A$ = "BASIC"
```

and

```
LET P$ = "ALLY"
```

the value of A$ + P$ is "BASICALLY", and the effect of

```
LET Q$ = A$ + P$
```

is to store the single word "BASICALLY" under the name Q$.

SUMMARY BOX 3

Dealing with words

LET name = word expression

1. The effect of the command is to find the value of 'word expression' and to store it under 'name'.
2. LET may be omitted.
3. 'name' is any name under which a word may be stored.
4. 'word expression' is any number of words or names under which words are stored all separated by plus signs.
5. The value of 'word expression' is the single word consisting of all the individual words in the expression one immediately after the other

The command for displaying information on the screen begins with the word 'PRINT'. When followed by a string of characters inside quotation marks it will print these characters on the screen. By typing

```
PRINT "THIS IS CLEVER"
```

and pressing ENTER, we cause the message

```
THIS IS CLEVER
```

to appear on the screen.

But if PRINT is followed by a name, or a list of names, it causes the items stored under these names to be displayed on the screen. In this way, we can examine items that we have stored in the computer's memory and also the results of processing them. So, after giving the commands

```
LET F = 1.2
LET G = 2 * F + 0.1
LET A$ = "BASIC"
LET B$ = A$ + "ALLY"
```

we find that

```
PRINT G
```

gives 2.5, that

```
PRINT B$
```

gives BASICALLY, and that

```
PRINT A$, F, B$
```

gives BASIC 1.2 BASICALLY

SUMMARY BOX 4

DISPLAY

PRINT character string
and
PRINT list of names

1. The effect of PRINT followed by a string of characters inside quotation marks is to display the string of characters.
2. The effect of PRINT followed by a list of names is to display successively the items stored under each name in the list.

More commands

The commands that have been explained in the previous section for storing, manipulating and displaying information are much the same on any computer. In this section we shall examine some commands that are much more specific to the Amstrad.

So far, the Amstrad's display has shown yellow letters and symbols on a blue background, giving the same appearance as yellow letters printed on blue paper. But the Amstrad is a colour computer, and we can change the colours of its display. The commands that are provided for this are PEN and PAPER. As you have probably guessed, PEN is for changing the colour in which letters and symbols are displayed on the screen, and PAPER is for changing the background colour against which they are displayed. The names of these commands are easy to remember because PEN has the same effect as if you change the colour of the ink in your pen when writing on paper, and PAPER gives the same effect as changing the colour of the paper you are writing on. We must tell the Amstrad what colour we want to change to, and this is done by following PEN and

PAPER by a number from 0 to 26 which represents a colour.

Before we can learn more about the Amstrad's colour capabilities we must learn about its MODES. The computer will work in any one of three modes, Mode 0, Mode 1 and Mode 2. When you switch on the machine is automatically in Mode 1. This mode has a screen of 40 letters wide, as you may have discovered if you have typed to the end of a line. In Mode 0 the screen is only 20 letters wide so that each of the letters is twice as wide as in Mode 1. In Mode 2 the screen is 80 letters wide so that to fit them all in they have to be half the width of the Mode 1 letters.

Now to change mode you just type MODE 2, for example, and the mode will have been changed. In Mode 0 you can have up to 16 of the 27 colours on the screen at the same time. In Mode 1 you can have four different colours on the screen and in Mode 2 you can have only two colours on the screen.

SUMMARY BOX 5

MODES

MODE 0
MODE 1
MODE 2

1. The effect of this command is to change the mode of the computer.
2. The modes have the following properties.

MODE 0	20 columns	Up to 16 colours on the screen at one time
MODE 1	40 columns	Up to 4 colours on the screen at one time
MODE 2	80 columns	Up to 2 colours on the screen at one time

We can now return to COLOUR!
The 27 colours available are as follows:

Colour number	Colour	Colour number	Colour
0	Black	14	Pastel Blue
1	Blue	15	Orange
2	Bright Blue	16	Pink
3	Red	17	Pastel Magneta
4	Magenta	18	Bright Green
5	Mauve	19	Sea Green
6	Bright Red	20	Bright Cyan
7	Purple	21	Lime Green
8	Bright Magenta	22	Pastel Green
9	Green	23	Pastel Cyan
10	Cyan	24	Bright Yellow
11	Sky Blue	25	Pastel Yellow
12	Yellow	26	Bright White
13	White		

Now each of these colours is put into an ink-well so that you can dip your pen in and write on the screen. The ink-wells have different contents in each mode, and the contents are as follows:

Ink-well number	Colour number in well		
	Mode 0	Mode 1	Mode 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24

4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	Flashing 1,24	20	1
15	Flashing 16,11	6	24

So we have 15 ink-wells and 27 colours. How can we get to the other colours? Well, the command INK will change the contents of one of the ink-wells. First we specify the well we are going to change and then we say what colour we are going to put in that well. So the command INK 3,9 will, in Mode 1, take the bright red ink out of ink-well number number 3 and put in green ink!

How do we get the ink from the well into our pen and onto the screen? This time we use the command PEN and the number of the ink-well we wish to use (note, it is the well number and not the colour number). So from the Table above we can see that PEN 2 will write in colour 20 when we are in Mode 1, which is Bright Cyan. Now, to change the colour of the paper we are writing on we again use the ink-well numbers. So try typing PAPER 3 and the paper will change to the colour of ink-well number 3, which contains colour 6 which is Bright Red.

Type CLS and you will see this better.

If you have typed this correctly you will notice a border suddenly appear around the paper. The colour of the border can also be changed, but this time we use the colour numbers, not the ink-well number. So typing BORDER 26 will change the colour of the BORDER to colour 26, that is Bright White.

There is one more trick to learn. If we use three numbers after the command INK then the ink-well will change from one of the colours to the other and back again. If the well we are changing is the one we are using for PEN or PAPER then either the writing or the background will flash between these two colours.

SUMMARY BOX 6

PEN number
PAPER number

1. The effect of PEN is to set the colour in which characters appear on the screen and of PAPER is to set the colour of the background on which they appear.
2. 'Number' must be a number from 0 to 15. Each corresponds to the ink-well of that number as specified in the above table.

SUMMARY BOX 7

INK number, number

1. The effect of INK is to put the colour represented in the above table as specified by the second 'number' into the ink-well specified by the first 'number'.
2. If a third number is placed after the second the contents of the well will flash between the colour specified by the second and third numbers respectively.

SUMMARY BOX 8

BORDER number

1. **The effect of BORDER is to change the colour of the border to the colour specified in the above table by 'number'.**

Before preceeding any further it is necessary to learn a new command. If you press down the CTRL key, the SHIFT key and the ESC key at the same time the computer will be reset to the same condition as when you first turned it on. This is necessary when trying out the various colour commands and the commands which we will consider next.

The Amstrad also allows you to create drawings on its screen. In this way you can create pictures or *graphics*.

In the computer's memory are various shapes which it can create on the screen. We have seen some of these, in the form of letters of the alphabet. However there are many more shapes than these. To see all the shapes, type in the following: –

```
10 FOR N = 32 TO 255    (Press ENTER)
20 PRINT N;CHR$(N)      (Press ENTER)
30 NEXT N               (Press ENTER)
```

Now type RUN. You have just written a program!

The canvas provided by the Amstrad consists of a rectangular array of boxes. The boxes are arranged in rows and columns, and the number of columns depends, as we have already seen, on the MODE we are using. In Mode 1 there are 40 columns, numbered from 1 to 40. In all modes there are 20 rows, numbered from 1 to 20. If we do not tell the computer where to print a

character it will print it at the present cursor position. If we have just cleared the screen, using CLS this will be at the top left hand corner, position 1,1 that is row 1 and column 1. When you made the program you had written run, it printed a series of numbers and characters. The numbers were the computer's way of remembering what each character was to be. To make character number 250, say, appear, we tell the computer to print CHR$(250). This is a little man, as you will have seen from the program.

So, if we type: CLS:PRINT CHR$(250) a man will appear at position 1,1. To make him appear elsewhere we use the command LOCATE.
LOCATE will move the cursor and then print at the new location.

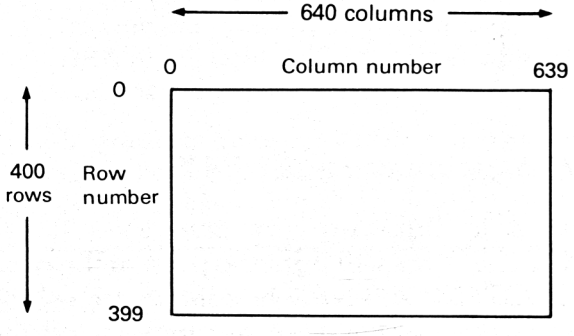


Figure 2.4 The graphics screen of the Amstrad.

So, LOCATE 20,1:PRINT CHR$(250) will print a man at column 20 and row 1. That is, the top row and half way along. Remember, if you now change to mode 0 which

only has 20 columns he will be at the right hand side, and if in mode 2 he will only be a quarter of the way across.

It is however a bit clumsy to only be able to print at 800 box positions when we might want to draw a line. If it had to be drawn in boxes it would appear a bit disjointed. Now, each character in the screen is actually made up of 64 smaller boxes, and each of *these* boxes can be further subdivided. In all there are 640 columns and 400 rows of the smaller boxes, called PIXELS. We can, in GRAPHICS MODE, light up any one of these pixels so that we can draw smooth pictures.

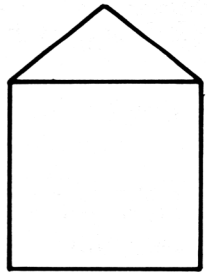


Figure 2.5 A shape created with DRAW.

The command to light up any one pixel is the command PLOT X,Y. This will light the pixel at the position x,y. Try PLOT 320,200, which is the centre of the screen. Note the the cursor does not move, and the script on the screen is still at the same positon as is was before you started. This is because you have moved not the cursor you are used to but the GRAPHICS CURSOR.

You can draw a line using the command DRAW. DRAW X,Y will draw the line from the current position of the graphics cursor (and you can move this using the PLOT command) to the new position x,y.

SUMMARY BOX 9

GRAPHICS

LOCATE number 1, number 2

1. Moves the Text cursor to the new position in column 'number 1' and row 'number 2'. Printed characters will appear at the new position.
2. The number of columns depends of the MODE you are working in, as previously specified. The number of rows is 25. Position 1,1 is the top left hand corner of the screen.

SUMMARY BOX 10

PLOT number 1,number 2
DRAW number 1,number 2

1. PLOT will light the pixel at the graphics position column 'number1' and row 'number 2'
   DRAW will light all the pixels in a line from graphics position with column number 1 and row 'number 2'.
2. In graphics there are 640 columns and 400 rows. Graphics position 0,0 is the bottom left hand corner of the screen.

Expanding the computer

There are several sockets at the back of the Amstrad at which items of equipment can be connected to it. They are shown in Figure 2.6. The sockets are all clearly labelled. Perhaps the best way to describe how the Amstrad can be expanded is to examine in turn the uses intended for these sockets. We will consider the sockets taking them in order from left to right as they appear when viewing the Amstrad from behind. The uses of the two sockets at the extreme left that are labelled

MONITOR and 5V DC have already been covered in the course of describing how to get the Amstrad ready for use. The purposes of the others are listed below.

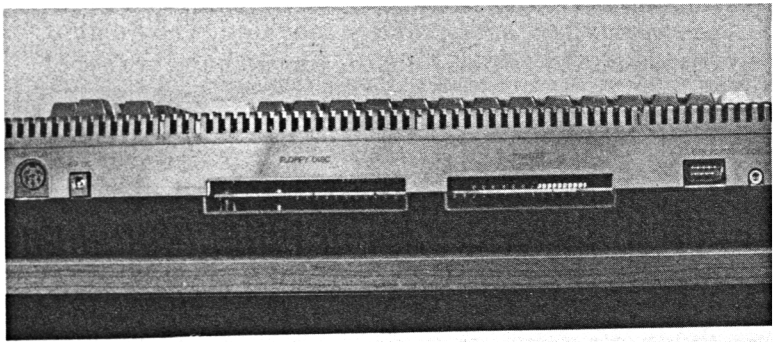


Figure 2.6 The sockets at the back of the Amstrad.

Floppy Disk. If you have tried to load a program using the built in cassette recorder you may have wondered why it took such a long time. The information the computer requires is stored on the tape as a magnetic message, in the same way as your music tape stores information which is decoded by your ordinary casette recorder. The use of disks, which are rather similar in appearance to a 45 rpm record, will speed up this load-

ing process to a remarkable degree. Although you may be prepared to wait while a program loads, if you want to use the computer for a database program, where information is constantly being sent to and recovered from a storage medium, then it is essential to use disks rather than to rely on cassettes.

Printer. This socket is for the connection of a printer to the Amstrad. It provides a standard, so-called Centronics, connection so that any printer of this type can be connected at once to the Amstrad. If the computer is to be used, for example, for word processing, then it is clearly essential to have a printer attached to the Amstrad so that the documents that are produced can be printed. It also allows a printed copy of the program that is in the computer to be made so that the printed copy can be taken away from the computer and examined at leisure, perhaps to correct or amend the program.

User Ports. The main item which will be attached to this socket will be Joysticks. If you intend to use your computer for playing arcade type games, then you will obtain a much more realistic effect if you use joysticks. AMSOFT produce a joystick especially for use with your Amstrad computer.

I/O. When generating sound from your Amstrad, you can feed this through your stereo using the auxiliary input socket and the I/O socket at the back of the computer.

Summary

The Amstrad is made ready for use by a simple procedure that involves plugging it into a monitor. The procedure is foolproof because the cables that are supplied can only be plugged into the sockets for which they are intended. From the keyboard a full range of letters, numbers and symbols can be typed. Generally, they are typed at the black keys, although the green

keys must be used to modify the effects of the black keys in order to allow the full range of characters to be produced. The green keys all have special purposes, but the CTRL key in particular can be used in conjunction with other keys to produce effects ranging from making a beep to changing the colours of the display. Once the keyboard is mastered, commands can be typed for the Amstrad, and this chapter has introduced commands for handling information, for changing the colours in the display and for creating graphics. We are now ready to start to write programs.

Chapter 3

Writing simple BASIC programs

We can now begin to write some simple programs of our own, because the commands that we met in Chapter 2 provide us with a small BASIC vocabulary. As we progress, we shall see that the programs we can create using only the commands with which we are familiar are quite restricted. But BASIC provides more facilities than we have met, and as we feel the need to do extra things with our programs we shall find that BASIC supplies us with the means to do it. In this way, we shall introduce more of the features of BASIC as they are needed to create programs for performing particular tasks. This approach brings out the reasons that BASIC possesses the features that it does as well as illustrating the uses to which they can be put.

First programs

As we know, a program is a list of instructions. In BASIC, an instruction is a numbered command, that is, a command preceded by a number. The number is usually referred to as a line number. An instruction is entered, in exactly the same way as a command, by typing it and pressing the ENTER key. But the computer reacts differently when it receives a program instruction to when it receives a command. Whereas a command is obeyed at once, an instruction is stored. The computer stores a program by storing all the individual instructions of the programs. The reasons for

storing the program are that once it is stored the computer can execute it as often as you like (in contrast to a command, which must be typed every time it is issued) and that it can be corrected or amended just by changing the necessary instructions rather than by re-typing it all.

When the instructions of a program are being entered they can be typed in any order because the Amstrad automatically stores them in the order given by their line numbers, placing the instruction with the lowest line number first, that with the next lowest second, and so on up to the last instruction, which has the highest line number. When the Amstrad executes the program that is stored in it, it does so by taking the instructions one by one, in the same order in which they are stored, and obeying the command part of each.

SUMMARY BOX 11

A program

1. **A program is a list of instructions.**
2. **An instruction is a line number followed by a command.**
3. **Instructions are stored in the increasing order of their line numbers.**
4. **A program is executed by obeying its instructions one by one in the order given by their line numbers.**

We can now write a program, and since it is the first one, we will make it do the simple task of storing two numbers, finding the difference between them and displaying the result. Although this task is small enough for us to plan a way of doing it in our heads, we shall put the planning down on paper because we shall soon come to tasks that do need quite careful planning before

we can begin to write the programs that tell the computer how to carry them out. If we were telling the computer how to do this task by giving it commands, we should begin by storing two numbers in the computer. This can be done by

```
LET FIRST = 6.8
LET TWO = 2.3
```

We could find their difference with

```
LET SUB = FIRST - TWO
```

Storing a caption to identify the result would be a good idea, and could be done by

```
LET C$ = "THE DIFFERENCE IS"
```

so that we could display the result with

```
PRINT C$,SUB
```

We have given five commands, which have been sufficient to accomplish the task. These commands give us the basis of our program. All we have to worry about now is making sure that the computer stores them in the right order so that they will be executed in the proper order when the program is run. To do this we must add line numbers to the commands, to turn them into program instructions, making sure that the command that we gave first gets the lowest line number, that which we gave second the next lowest, and so on. It might seem natural to use the numbers 1, 2, 3... as line numbers, but we shall use 10, 20, 30 and so on. The reasons for this will become apparent later on, but it is basically so that there are gaps between the line numbers in case we want to add some extra lines to the program.

From this, we get our first program as:

```
10 LET FIRST = 6.8
20 LET TWO = 2.3
30 LET SUB = FIRST - TWO
40 LET C$ = "THE DIFFERENCE IS"
50 PRINT C$;SUB
```

Each line is entered just by typing it and pressing RETURN. To see the program that is stored in the computer at any time, you give the command

```
LIST
```

and, as soon as you press ENTER, the program that is stored in the computer is displayed on the screen. To execute the program that is stored in the computer, give the command

```
RUN
```

If the program that is given above has been typed *exactly* as it is presented, after typing RUN and pressing ENTER you will see the line

```
THE DIFFERENCE IS 4.5
```

before the 'READY' message and the cursor are displayed again. If you do not get this line, and in particular if you get

```
Syntax error
```

do not worry, it is probably because what you have typed is not exactly the same as the program that is listed above. See if you can find where your program differs

and if it does simply retype the line or lines that are different and then run the program again.

SUMMARY BOX 12

Examining and executing a program

LIST
RUN

1. **The command LIST causes the program that is stored in the computer to be displayed on the screen.**
2. **The command RUN causes the program that is stored in the computer to be executed**

Our first program should illustrate that when the computer runs the program stored in it, it selects the command parts of the program instructions one by one, in the order given by the line numbers, and obeys them. The end result is exactly the same as when the commands are typed individually in the same order, but in either case the order is important. To get the commands in the wrong order would be to give the computer the wrong method for carrying out its task. Of course, the computer completes its task much more quickly by running its stored program than it does when we have to type the commands individually .

A program to store these words and then to display a phrase made up from them can be written along the same general lines as the first program. Although it will deal with words rather than numbers, it will store, process and display them in the same pattern. Before we start to enter the new program we should give the command

NEW

to clear the previous program from the Amstrad's memory. If we do not do this, we may well end up with parts of the old program mixed up with the new one, giving a stored program that is neither one thing nor another.

SUMMARY BOX 13

Clearing a program

NEW

1. **This command clears the computers memory of the program that is stored in it, and should be used before entering a new program.**

Our second program is:

```
10 LET A$ = "STORE "
20 LET B$ = "THE "
30 LET C$ = "PROGRAM "
40 LET Z$ = A$ + B$ + C$
50 PRINT Z$
```

Running this program will give the display

```
STORE THE PROGRAM
```

Note that each word is stored with a space following it. If this is not done, when the words are 'added' by line 40, the result will be a string of letters with no gaps and all the words will run on from each other. It is easy to make this program produce another phrase. Typing

```
40 LET Z$ = B$ + C$ + A$
```

will cause this line 40 to replace the previous one, and when the amended program is run it will give the display

Our third program is a graphics program. We can write a program that creates the shape of Figure 2.5 by incorporating the commands that we gave in Chapter 2 for producing this shape into a program. The resulting program is:

```
 10 CLS
 20 INK 1,26
 30 INK 0,3
 40 PLOT 320,200
 50 DRAW 420,200
 60 DRAW 420,100
 70 DRAW 320,100
 80 DRAW 320,200
 90 DRAW 370,230
100 DRAW 420,200
```

The consequence of putting the instructions in the wrong order can be demonstrated graphically with this program, for by changing the line numbers of a few instructions to give the following program:

```
 10 CLS
 20 INK 1,26
 30 INK 0,3
 40 PLOT 320,200
 50 DRAW 420,200
 60 DRAW 370,230
 70 DRAW 320,100
 80 DRAW 420,100
 90 DRAW 320,200
100 DRAW 420,200
```

We find that the shape it produces is that shown in Figure 3.1 which is quite different from that given by the original program.

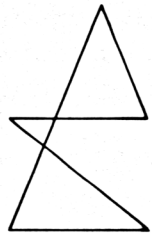


Figure 3.1 A second shape created by DRAW.

Editing

Now that you have typed a few programs, it is worth considering how to correct any mistakes that may occur during typing. These methods of correcting errors apply to the text of programs or to anything else that you enter. They are worth knowing because they are the easiest and quickest ways to correct the errors that inevitably occur while typing.

Whatever you are typing, you can correct a mistake if you notice it as soon as it is made by deleting it with the DEL key. The DEL key always deletes the character to the left of the cursor, and if the mistake that you want to correct is not at the end of the text then, with the cursor at the end of the text, you can always delete all the characters up to the erroneous one and then start typing again from there. For this reason it is probably still preferable to abandon an improperly typed command and retype it properly, particularly as most of the commonly used commands are quite short.

When correcting or changing a program, there are several methods that can be used. Any instruction can be replaced simply by typing a new version of the instruction and giving it the same line number. An

instruction can be inserted at any point in a program by giving it a line number between those of the two instructions betwen which it is to be placed. An instruction can be deleted just by typing its line number and pressing RETURN. But if a fairly long instruction contains just a simple error, then it is better to use COPY to copy the correct part and to type only the correction rather than to retype the whole instruction. Earlier in this book we did mention briefly the use of the COPY key. There are in fact two main ways to edit a line.

EDIT

If you type EDIT and then a line number the line will appear with the cursor over the first character. This is the EDIT cursor. To leave a letter or character as it is, move the cursor over it, using the arrow keys. To delete a character, move one place to the right of the character you wish to delete and press the DEL key. To enter a new character just move to the correct position and type in the new character.

COPY

Use SHIFT and the arrow keys to make the EDIT cursor appear. Move the cursor to the start of the line you wish to edit. Now press COPY for each of the letters you wish to keep. If you wish to remove a letter, move over it using the SHIFT and the arrow key rather than the copy key. To enter a new character just type it in as you reach the correct position.

SUMMARY BOX 14

Editing text

The DEL key deletes the character to the left of the cursor.

SUMMARY BOX 15

Editing a program

1. An instruction is inserted just by typing it and pressing ENTER.
2. An instruction is deleted by typing its line number and pressing ENTER.
3. An instruction is replaced by entering a new instruction with the same line number.
4. COPY can be used to edit instructions.
5. The command EDIT followed by a line number will cause the instruction with that line number to be displayed conveniently for editing together with the EDIT cursor.

More BASIC instructions

In this section, we shall introduce some more BASIC instructions. The ones we have met already allow us to give rather simple programs to the computer, but if we are to write programs for more advanced tasks we shall need instructions that are capable of rather more than those we have met so far. As a basis for introducing the new features and for showing why we need them we shall consider the problem of writing a standard letter to send to one of our friends on her birthday. A simple program to do this using only the commands that we have met already is given below. It stores the lines of the letter one by one, then clears the screen and displays the lines one by one. The program is:

```
10 LET A$ = "DEAR NANCY"
20 LET B$ = "THIS IS TO WISH YOU A HAPPY
   BIRTHDAY"
```

```
30 LET C$ = "AND MANY HAPPY RETURNS OF
   THE DAY"
40 LET D$ = "LOVE"
50 LET E$ = "GARRY"
60 CLS
70 PRINT A$
80 PRINT B$
90 PRINT C$
100 PRINT D$
110 PRINT E$
```

When this program is run, it displays on an otherwise clear screen

```
DEAR NANCY
THIS IS TO WISH YOU A HAPPY BIRTHDAY
AND MANY HAPPY RETURNS OF THE DAY
LOVE
GARRY
```

As soon as you have written this program, it may strike you that the same greeting could be sent to any of your friends if only you could change NANCY to another name. In fact, if you could give the name of the friend whose birthday it is to the computer, it could produce a personal greeting merely by inserting the name in the appropriate place in the otherwise standard letter. By providing the INPUT instruction, BASIC allows information, such as the name of a friend, to be given directly to a program for purposes such as this.

In a program, when the computer reaches an instruction such as

```
10 INPUT N$
```

it first prints a question mark on the screen and then waits for you to type in something from the keyboard

and to press ENTER to show that you have finished. The question mark is displayed as a way of reminding you that you should type something to give to the computer. What you type is then accepted and stored, in this case under the name N$. The computer then goes on to deal with the next instruction. The INPUT Instruction also allows you to display a message rather than just a question mark, so that you can tell the computer to display a suitable message to remind you of what you are supposed to type. In this case we could use:

```
10 INPUT "NAME OF FRIEND";N$
```

This makes the computer display:

```
NAME OF FRIEND?
```

and await your response, accepting it for storage under N$ when you press ENTER.

With the use of this instruction, we can write the program for producing a birthday greeting to any of our friends as:

```
10 LET A$ = "DEAR "
13 INPUT "NAME OF FRIEND";N$
17 A$ = A$ + N$
20 LET B$ = "THIS IS TO WISH YOU A HAPPY
   BIRTHDAY"
30 LET C$ = "AND MANY HAPPY RETURNS OF
   THE DAY"
40 LET D$ = "LOVE"
50 LET E$ = "GARRY"
60 CLS
70 PRINT A$
80 PRINT B$
90 PRINT C$
```

```
100 PRINT D$
110 PRINT E$
```

Note that to get this program we need only edit line 10 of the previous program and insert lines 13 and 17. Further if we had not left a gap between the numbers used for line numbers there would have been no way to insert these extra lines.

SUMMARY BOX 16

Entering data from the keyboard

INPUT name
or
INPUT character string; name

1. **When executed this instruction causes a question mark or, if it is given, 'character string' followed by a question mark, to be displayed and then the computer waits for a response to be typed.**
2. **The response, which is terminated by typing ENTER, is stored under 'name'.**

BASIC provides another way of including information in a program with its READ and DATA instructions. You may wonder why yet another way of handling information is needed. The comparison of BASIC with ordinary English is helpful in answering this. In English there are usually several ways of saying much the same thing, except that each way has its own shade of meaning and emphasis that makes it particularly well suited to a given situation. The same is true with BASIC, where a particular way of telling the computer to carry out some action, and in this case of telling it how to record information, will be more suitable than another. Being able to choose

from different methods of telling the computer to do what we want allows us to pick the one that is most suitable or most convenient.

Various items of data can be made available to the computer when it is runnning a program by placing them in a DATA statement in the program. A DATA statement consists of the word DATA followed by the items, all of which are separated by commas. We can place the four words 'HAPPY', 'BIRTHDAY', 'TO' and 'YOU' in a a DATA statement in this way:

```
DATA "HAPPY", "BIRTHDAY", "TO", "YOU"
```

The order of the items is important, for the first READ statement that is obeyed by the computer in a program reads the first item of data from the DATA statement, the second READ instruction that is obeyed reads the second item, and so on. If we cannot fit all the items of data into one DATA statement, or if we choose not to, then we may use several DATA statements, and the line numbers of the DATA statements indicate the order of the items of data in a quite natural way. The following program will read the words from a DATA statement and display them in the order in which they appear:

```
10 READ A$
20 PRINT A$
30 READ B$
40 PRINT B$
50 READ C$
60 PRINT C$
70 READ D$
80 PRINT D$
90 DATA "HAPPY", "BIRTHDAY", "TO", "YOU"
```

But the words could equally well have been given in four DATA statements as:

```
 90 DATA "HAPPY"
100 DATA "BIRTHDAY"
110 DATA "TO"
120 DATA "YOU"
```

as this presents the items of data in exactly the same order as they appeared in the single DATA statement used originally.

It is probably clear from this that another way of making the computer produce our birthday greeting to Nancy that uses READ and DATA is:

```
 10 CLS
 20 READ A$
 30 PRINT A$
 40 READ B$
 50 PRINT B$
 60 READ C$
 70 PRINT C$
 80 READ D$
 90 PRINT D$
100 READ E$
110 PRINT E$
120 DATA "DEAR NANCY"
130 DATA "THIS IS TO WISH YOU A HAPPY
    BIRTHDAY"
140 DATA "AND MANY HAPPY RETURNS OF THE
    DAY"
150 DATA "LOVE"
160 DATA "GARRY"
```

SUMMARY BOX 17

Including data in a program

READ name
DATA data list

1. When executed, a READ instruction reads an item of data from 'data list' and stores it under 'name'.
2. A DATA statement may contain a 'data list' consisting of several items of data separated by commas. A program may contain several DATA statements.
3. The items in a DATA statement are ordered from left to right. When there is more than one DATA statement in a program the order of the items starts at the first item in the statement with the lowest line number, and finishes at the last item in the statement with the highest line number.
4. The first time that a READ statement is executed in a program, the first item of data is read, the second READ statement to be executed reads the second item, and so on.

Making it easier to write programs

The briefest look at the previous two programs shows that they are very repetitive. Further, if we want to make the computer produce longer messages, writing the program that tells the computer how to go about it will become very boring. The fact that the programs are repetitive suggests that we ought to be able to simplify them. After all, our aim is to make the computer carry out tasks for us, but there is little advantage in this if it takes as much effort to tell the computer how to do them as it would for us to do them ourselves.

In the previous two programs, pairs of instructions such as:

```
READ A$
PRINT A$
```

are repeated over and over. Really, we would like to write down the pair of instructions once only. If we

could do this and then tell the computer that as soon as it has obeyed them once it should go back and do them again, we should be able to tell the computer to do a good deal of work by writing only a few instructions.

BASIC provides us with a primitive means of doing this in the form of the GOTO instruction. It takes the form of the word 'GOTO' followed by a line number, and when it is obeyed it causes the Amstrad to go to the instruction with the specified line number and to obey the command in that instruction next. By using this instruction, our birthday greeting can be produced by the much shorter program:

```
10 CLS
20 READ A$
30 PRINT A$
40 GOTO 20
50 DATA "DEAR NANCY"
60 DATA "THIS IS TO WISH YOU A HAPPY
   BIRTHDAY"
70 DATA "AND MANY HAPPY RETURNS OF THE
   DAY"
80 DATA "LOVE"
90 DATA "GARRY"
```

The program works very well except for one thing which you will notice when you run it. Although the program produces the greeting we expect, it then displays:

```
DATA exhausted in 20
```

We do not really want this message to appear at the end of the output from our program (it certainly has nothing to do with Nancy's birthday). If you thought that the GOTO instruction was rather too good to be true,

you may feel that your worst fears are confirmed. But at least we should investigate a little further to find the cause of this unwanted message.

When the computer runs the previous program line 20 causes it to read an item from the DATA statement, line 30 then causes it to print what it has just read, and line 40 causes the computer to go back to line 20 to read another item. Even after the last item of data has been read, line 40 still sends the computer back to line 20 to read another one. Since there are no more items of data to read, the computer cannot do what we have told it to do, and it automatically sends us a message to indicate its protest. The message 'DATA exhausted in 20' actually means 'there were no more data items to be found when the READ instruction with line number 20 was being carried out'.

This occurrence is typical of what can happen when a GOTO instruction is used by itself. The error has caused the computer to stop in this instance, but if we enter the program

```
10 PRINT "THE COMPUTER WILL NOT STOP."
20 GOTO 10
```

then running this program will cause the computer to keep displaying the sentence from line 10 for ever. To stop it, we must take some action ourselves. There are two possibilities: we can either press ESC twice or press SHIFT, CTRL and ESC together (which will wipe the memory clean!).

SUMMARY BOX 18

Changing the order in which instructions are obeyed.

GOTO line number

1. **The effect of this instruction is to cause the computer to go to the instruction with the line number given by 'line number' and to obey it next. The computer then carries on with the program by obeying the instruction following it.**

2. **When used to jump back to an instruction with a lower line number, it must be used with care as it creates a loop from which there is no escape.**

Having seen that the GOTO instruction when used by itself is potentially quite dangerous, we should mention that BASIC provides other instructions with which it can be used quite sensibly. We shall return to this shortly.

We started with a program to send a birthday greeting to a particular person, and then made the program more useful so that it could send a birthday greeting to any of our friends. It would be even more useful if it could be used to send any greeting to any person. We shall now develop the previous program to make it do this. First, the program must ask for the name of the person to whom the greeting is to be sent, and then it must ask for the greeting itself. After that it must read the other parts of the message and display them. But we can make the program use the same simple repetitive format if we place dummy items in the data such as NAME to represent any name and GREETING to represent any greeting. As long as the computer can replace NAME when it reads it by DEAR followed by the name we have given it in response to an INPUT instruction and, similarly, can replace GREETING by the greeting we have given it, the task we have set the computer can be carried out.

Before we can tell the computer how to do this, we must introduce a further BASIC instruction. We need to

tell the computer something like 'if you have read 'NAME' then replace it by 'DEAR' followed by the name we just gave you'. And for purposes such as this, BASIC provides us with an instruction involving the words IF and THEN. This instruction allows us to describe to the computer how to make decisions in much the same way as we should describe decision-making in ordinary English. The instruction takes the form of the word 'IF' followed by a test, the word 'THEN' and finally another BASIC command. The test can be a comparison of two values to see if they are equal or to see if they differ. (Other comparisons can also be made, and they are given in the Summary Box.) The command following 'THEN' can be almost any BASIC command. When an instruction of this type is carried out, the test is made, and if it is satisfied the command following 'THEN' is obeyed, otherwise nothing more is done and the computer moves on to the next instruction. In this way, we can let the computer decide whether or not to carry out a command by looking at the result of a test.

The instruction that tells the computer to display 'THREE' only if the value stored under X is 3 is:

```
IF X = 3 THEN PRINT "THREE"
```

The instruction that recognises when 'NAME' is stored under A$ and replaces it by 'DEAR' followed by the name stored under N$ is:

```
IF A$ = "NAME" THEN LET A$ = "DEAR " + N$
```

By using this conditional, IF-THEN, instruction, we can write our program to send any of our friends any greeting as:

```
10 INPUT "FRIEND'S NAME"; N$
20 INPUT "GREETING"; G$
```

```
30 CLS
40 READ A$
50 IF A$ = "NAME" THEN LET A$ = "DEAR " + N$
60 IF A$ = "GREETING" THEN LET A$ = G$
70 PRINT A$
80 GOTO 40
90 DATA "NAME", "GREETING", "LOVE",
    "GARRY"
```

When this program is run, its output is similar to this:

```
FRIEND'S NAME?  JOANNE
GREETING?  HAPPY NEW YEAR
```

The screen then clears before displaying

```
DEAR JOANNE
HAPPY NEW YEAR
LOVE
GARRY
DATA exhausted in 40
```

There's that exhausted DATA again! However, we now have the ability to overcome the problem using the IF . . . THEN command. We shall put in an extra line:

```
75 IF AS = "GARRY" THEN GOTO 75
```

Note that this time we have purposely made the computer go on repeating line 75 forever so as not to spoil our display.

SUMMARY BOX 19

Making decisions

IF test THEN command

1. The 'test' can compare two values for equality (=) for non-equality (<>). It can also see if one value is greater than (>) or less than (<) another.
2. The 'command' can be any BASIC command or sequence of commands.
3. When executed, the 'command' is obeyed only if the 'test' is satisfied.

Saving and loading programs with a cassette player

We now turn to the use of the cassette player with the Amstrad. Its main uses are to copy the program that is stored in the computer onto a cassette, so that the program is saved in a permanent form, and to put a program that has previously been saved on cassette into the computer. The reasons for doing this are that when the computer is unplugged everything in its memory is lost, including any program stored there. If you have previously keyed in a long program, it would be discouraging to know that the next time you needed the same program you would have to type it in all over again. By saving it on a cassette, you can load it again directly from there. Also, if you buy a program a copy of it must be given to you in some form. You could be given a listing of it on paper but, again, you would have to type what is, in all probability, a lengthy program. If it is supplied on a cassette it can be loaded from the cassette player. A disk drive can be used with its disks to load and save programs in just the same way, and it is much quicker. But the Amstrad has a built in cassette player while disk drives are rather expensive.

For recording programs it is better to use a short C10 or C15 cassette than the longer C60 and C90 tapes. The magnetic tape itself is thicker and consequently less liable to stretch than it is on a longer tape, making it less likely for the recording of a program to be corrupted. It

is also quicker to locate a program on a shorter tape, and it results in the waste of much less time should the computer not find the program you want on a cassette for any reason. The Amstrad's commands for saving and loading a program on a cassette are SAVE and LOAD.

The procedure for saving the program that is stored in the computer by copying it on a cassette is as follows:

* Wind the tape forward a short way so that it is well past the plastic reader.

* Type the command.

SAVE "PROGRAM"

'PROGRAM' is the name that the program will be saved under when you give this command, but you may choose any name you like.

* Press the RECORD and PLAY buttons on the cassette recorder and then the ENTER key on the Amstrad.

* The message Saving "PROGRAM" block 1 will appear above the display area. The READY message will appear with the cursor when the program has been saved, and you should then stop the cassette player.

By following this procedure you make a copy of the program that is stored in the computer on a cassette. The program itself remains in the computer.

The procedure for loading a program that has previously been recorded on a cassette into the computer is as follows:

* Place the cassette containing the program in the cassette player, and rewind the tape.

* Type

LOAD "PROGRAM"

If the program was saved under a name other than PROGRAM, then you must use that name. If you just want to load the first program on a cassette (and this is a good reason for recording only one program per side on a cassette), you can type

LOAD ""

* Press the ENTER key on the Amstrad and then press the PLAY button on the cassette player.

* The message Loading "PROGRAM" Block 1 will appear while the program is being loaded. The READY message and cursor will reappear when the program is loaded.

SUMMARY BOX 20

Loading and saving programs

LOAD "name"
SAVE "name"

1. **The effect of LOAD is to copy the program called "name" from a cassette tape into the computer.**
2. **The effect of SAVE is to copy the program in the computer onto cassette tape under the name "name".**

Summary

A BASIC program is a sequence of instructions, and an instruction is a numbered command, with the number showing the position of the instruction in the sequence and the order in which it must be stored and executed. We have written our first programs by noting that a program for a particular task can be constructed

by placing numbers in front of each of a set of commands that we know can cause a task to be accomplished. This must be done in such a way that they remain in the same order as before. The means provided by the Amstrad for editing text are described. They are particularly useful when editing or correcting a program.

By means of an extended example based on writing a program to produce a letter, we see how extra facilities are needed to make the program more generally useful and that, in every case, BASIC provides the facilities we need. In this natural way, BASIC's facilities for handling data, for repetition and for making decisions are introduced. After this, we have met many, although by no means all, of the instructions of BASIC.

Finally, we look at how the cassette player can be used in conjunction with the Amstrad. It allows us to store the program that is stored in the computer so that we do not lose it when the computer is unplugged, as we would otherwise. It also allows us to load programs that are recorded on cassette into the computer so that we have a convenient way to load the programs that we have previously saved and that we have bought.

Chapter 4

Graphics and sound

The Amstrad is well-equipped for producing both colour graphics and sound. We have already seen a little of its ability to produce highly detailed colour graphics, and it can produce a wide range of sounds, musical and otherwise, through its internal loud-speakers. The Amstrad's BASIC posesses quite a number of BASIC instructions for graphics and for sound production. We shall meet some of them and see how they can be used in this chapter.

GRAPHICS

We have alrealy met some of the commands that are used in connection with high-resolution graphics. PEN and PAPER set the foreground and background colours of a graphics display. PLOT and DRAW are used, respectively, to position the graphics cursor and to draw a line from the graphics cursor to a point of specified distance away.

We shall start by writing a graphics program that can be adapted easily to draw any shape because it reads a description of the shape from a DATA statement. By changing the DATA statement only, the program can draw any shape at all. The DATA statement gives the number of lines that have to be drawn to make the shape, and then the same number of pairs of numbers each showing where each line must be drawn to. The program is:

```
10 CLS
20 INK 1,26
30 INK 0,3
40 LET C = 0
50 PLOT 150,200
60 READ N
70 READ COL, ROW
80 DRAW COL, ROW
100 LET C = C + 1
110 IF C< N GOTO 70
120 DATA 4,300,200,300,50,150,50,150,200
```

Under the name C a count is kept of the number of lines that have been drawn so far, and the program plots lines repeatedly until the correct number have been drawn. As it is presented, the program causes a square to be drawn.

This program shows that it is sometimes useful to have a form of repetition using GOTO for situations in which the number of repetitions required is known in advance. BASIC provides us with this in the form of the pair of instructions FOR and NEXT. The advantage of using this pair of instructions in the appropriate situation is that it provides a counter that is automatically incremented, thereby saving us the trouble of writing the instructions for it ourselves. When this form of repetition is used, the instructions to be repeated are placed between a FOR and a NEXT instruction, and FOR must be followed by a name under which a counter can be stored together with the starting and finishing values for the count. To illustrate this, the previous program can be rewritten using FOR and NEXT as:

```
10 CLS
20 INK 1,26
30 INK 0,3
```

```
40 PLOT 150,200
50 READ N
60 FOR C = 1 TO N
70 READ COL, ROW
80 DRAW COL, ROW
90 NEXT C
100 DATA 4,300,200,300,50,150,50,150,200
```

This program has two instructions fewer than the previous one because the instructions for maintaining the counter are no longer needed.

SUMMARY BOX 21

Fixed numbers of repetitions

```
FOR name = start TO finish
        block of instructions
NEXT name
```

1 The effect of this instruction pair is to cause 'block of instructions' to be executed a fixed number of times.

2 A counter is stored under 'name'. It starts with the value 'start' and is incremented each time the block of instructions is obeyed. The final repetition is done with the value 'finish' stored under 'name'.

3 The total number of repetitions is 'finish' — 'start' + 1.

So far with the PLOT and DRAW commands we have used only two numbers to follow them. We can use three numbers, and in this case the third number will decide the ink-well colour we use for the line. If we choose the paper number we will not be able to see what we have drawn. Another way of looking at this is to say that if we draw a line using one colour and then draw the same

line using the paper colour the line will first be drawn and then it will disappear. One application of this 'unplotting' is in animation. We can make a shape appear to move across the screen by plotting it in one position, unplotting it and then moving the position across the screen before repeating the process. A program to do this with our square is:

```
10 CLS
20 INK 2,19
30 INK 0,3
40 LET COL = 100
50 PLOT COL,100
60 DRAW COL,150,2
70 DRAW COL + 50,150,2
80 DRAW COL + 50,100,2
90 DRAW COL,100,2
100 DRAW COL,150,0
110 DRAW COL + 50,150,0
120 DRAW COL + 50,100,0
130 DRAW COL,100,0
140 LET COL = COL + 1
150 IF COL< 550 GOTO 50
```

The movement can be speeded up by increasing the number of columns that the shape moves across the screen from the one set by line 140 to, say, three by changing line 140 to:

```
140 LET COL = COL + 3
```

The way that we have written the group of four DRAW instructions twice in succession in almost identical form seems rather unnecessary. Again, BASIC provides a means of avoiding this, and of passing work from the programmer to the computer. By providing the *subrou-*

tine facility, BASIC allows us to write a set of instructions as a self-contained unit which can then be called from a program as often as it is needed. The BASIC words that are used in this context are GOSUB followed by a line number to call the subroutine to cause the computer to return to the main program and resume at the instruction following the GOSUB. The previous program can now be rewritten, making use of a subroutine starting at line 200 to plot the shape, as:

```
10 CLS
20 INK 2,19
30 INK 0,3
40 LET COL = 100
50 PLOT COL,100
60 LET F = 2
70 GOSUB 200
80 LET F = 0
90 GOSUB 200
100 LET COL = COL + 1
110 IF COL< 550 GOTO 50
120 END
200 DRAW COL,150,F
210 DRAW COL + 50,150,F
220 DRAW COL + 50,100,F
230 DRAW COL,100,F
240 RETURN
```

We have introduced the END instruction in line 130. This simply marks the end of the program, and it is necessary here to prevent the computer from going on to execute the instructions in the subroutine when it has completed those in the program. Using a subroutine makes the program easier to recall than before, showing more clearly that the repetitive part of the program positions the cursor, plots a shape, unplots it and

then changes the position. The effect of the GOSUB and RETURN instructions in this program in communicating between the program and the subroutine is illustrated in Figure 4.1.

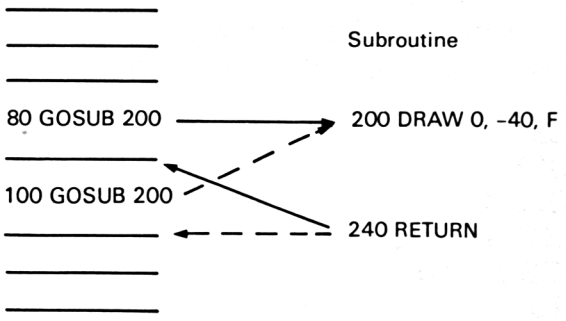


Figure 4.1 Communication between program and subroutines.

Having seen that a subroutine containing graphics commands can be used to draw a shape at a specified position, we can further illustrate the usefulness of the subroutine by creating patterns based on a single shape. We have chosen a hexagon as the shape because it can be used to build some interesting patterns. The following program calls the subroutine starting at line 500 which can draw a hexagon. It is used once to place a hexagon in the centre of the screen.

First we must learn another new instruction for drawing graphics. You may have tried to follow the shapes we have drawn and found it a bit difficult because we were relating everything back to the corner of the screen. We can make this a lot easier by using the command DRAWR instead of DRAW. When we used DRAW we drew a line from our current cursor position

to the new position relative to the corner. With DRAWR we draw a line from our present position to a number of graphic columns and rows relative to our present position. This makes it much easier to visualise what is happening, and thus easier to make up pictures. If we want to go up or right we can use positive numbers and if we want to go left or down we can use negative numbers.

```
10 CLS
20 INK 1,19
30 INK 0,3
40 PLOT 320,200
50 GOSUB 500
60 END
500 DRAWR 0,-30,1
510 DRAWR 27,-15,1
520 DRAWR 27,15,1
530 DRAWR 0,30,1
540 DRAWR -27,15,1
550 DRAWR -27,-15,1
560 RETURN
```

Knowing that the subroutine works, we can now draw a row of hexagons by repeatedly drawing hexagons across the screen. This can be done by:

```
10 CLS
20 INK 1,19
30 INK 0,3
40 FOR COL = 1 TO 6
50 PLOT COL*54,100
60 GOSUB 500
70 NEXT COL
80 END
500 DRAWR 0,-30,1
```

```
510 DRAWR 27, -15,1
520 DRAWR 27,15,1
530 DRAWR 0,30,1
540 DRAWR -27,15,1
550 DRAWR -27, -15,1
560 RETURN
```

We can now fill the screen with hexagons to give the pattern shown in Figure 4.2 by repeatedly plotting rows of hexagons. The program for this is:

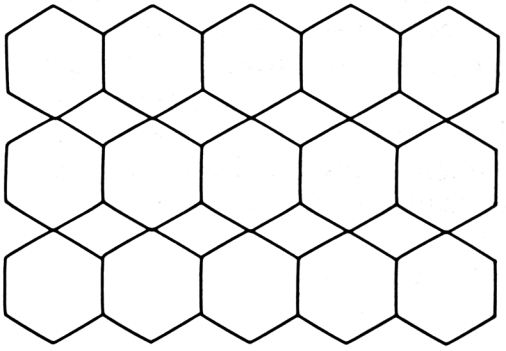


Figure 4.2 A pattern of hexagons.

```
10 CLS
20 INK 1,19
30 INK 0,3
40 FOR ROW = 1 TO 6
50 FOR COL = 0 TO 11
60 PLOT COL*54, ROW*60
70 GOSUB 500
80 NEXT COL
90 NEXT ROW
100 END
500 DRAWR 0, -30,1
510 DRAWR 27, -15,1
520 DRAWR 27,15,1
```

```
530 DRAWR 0,30,1
540 DRAWR -27,15,1
550 DRAWR -27,-15,1
560 RETURN
```

In this pattern there are six rows each with 12 hexagons, and 72 hexagons are drawn, which means that the subroutine has been called 72 times. By calling the subroutine, we have saved ourselves an immence amount of work: first imagine writing out the six DRAW instructions for a hexagon 72 times! The pattern is a very interesting one, and can be seen as alternate rows of hexagons and diamonds or as rows of interlocking larger hexagons. The pattern can be amended quite easily to give the 'honeycomb' pattern shown in Figure 4.3. To create it, we basically need only to displace alternate rows by a small amount. The program for the 'honeycomb' pattern is:

```
10 CLS
20 INK 1,19
30 INK 0,3
40 FOR ROW = 1 TO 6
50 FOR COL = 0 TO 11
60 PLOT COL*54 + ROW*27, ROW*45
70 GOSUB 500
80 NEXT COL
90 NEXT ROW
100 END
500 DRAWR 0,-30,1
510 DRAWR 27,-15,1
520 DRAWR 27,15,1
530 DRAWR 0,30,1
540 DRAWR -27,15,1
550 DRAWR -27,-15,1
560 RETURN
```

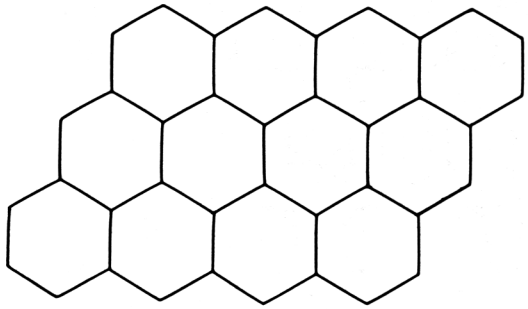


Figure 4.3 A 'honeycomb' pattern.

All these programs follow a very similar pattern, and although we have numbered them all neatly, they can all be obtained by starting from the first program for plotting a single hexagon and inserting or amending lines. But it is the use of the subroutine that makes the developments possible.

SUMMARY BOX 22

The Subroutine

GOSUB line number
RETURN

1 **The instruction GOSUB causes the computer to go to the subroutine starting at the line number 'line number'.**

2 **The subroutine must end with the instruction RETURN which causes the computer to return to the instruction following GOSUB and to obey that instruction next.**

Figure 4.4 shows some patterns that you might care to create on the Amstrad. You need to identify the pattern and its constituent shapes, but then the drawing programs will take forms very similar to the ones we have just written.

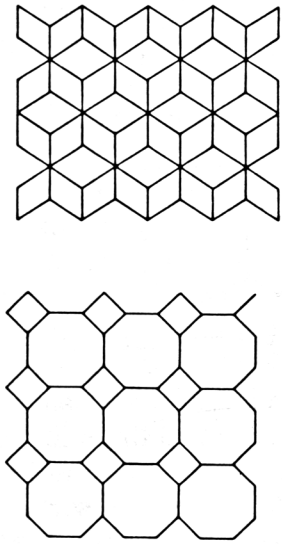


Figure 4.4 Some patterns to draw.

Still using the same few graphics commands, we can write an 'artist's program' which allows its user to create drawings on the screen. We shall write the program so that it responds to certian keys by drawing a short line in the direction suggested by the positions of the keys. The keys to be used are shown as they are situated on the keyboard in Figure 4.5 and the corresponding directions in which they cause a line to be drawn are shown in Figure 4.6. In this way, pressing the A key, for example, when the program is running will cause a line to be drawn up the screen, and lines can be drawn in directions at any multiple of 45 degrees from this.

We could use an INPUT instruction in the program to detect which key is being pressed, but this would not only temporarily halt the program but would also cause

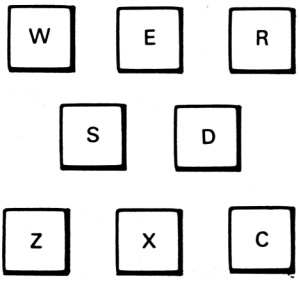


Figure 4.5 The keys used by the artist's drawing program.

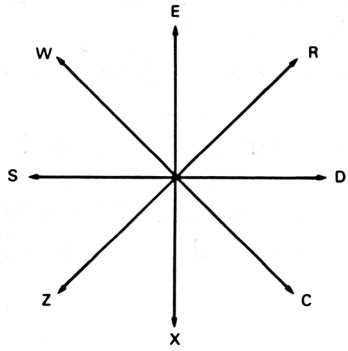


Figure 4.6 The directions associated with the keys in Figure 4.5.

a distraction by displaying its prompt. Instead, we shall introduce a new command, of the kind that is used in many computer games, which simply scans the keyboard to see if a key is being pressed at the instant that it is being executed. If a key is being pressed it reports which it is, and if no key is being pressed it reports *that*. The instruction is the INKEY$, and when executed it gives the single character that corresponds to the key that is being pressed or, if no key is being pressed, it gives nothing. It can be used as in

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

to store under the name A$ either the single character or no charcter.

SUMMARY BOX 23

Scanning the Keyboard

INKEY$

1 **INKEY$ gives the character that corresponds to the key that is being pressed when it is executed. If no key is being pressed, it gives nothing.**

A description of our program is given in the *flowchart* of Figure 4.7. Points at which a decision must be made are shown by diamonds: these will correspond to IF – THEN instructions in the program. The program based on this plan is:

```
10 CLS
20 INK 0,13
30 INK 1,19
40 PLOT 320,200
50 LET A$=INKEY$
60 IF A$="E" THEN DRAWR 0,1,1: GOTO 50
70 IF A$="R" THEN DRAWR 1,1,1: GOTO 50
80 IF A$="D" THEN DRAWR 1,0,1: GOTO 50
90 IF A$="C" THEN DRAWR 1,-1,1: GOTO 50
100 IF A$="X" THEN DRAWR 0,-1,1: GOTO 50
110 IF A$="Z" THEN DRAWR -1,-1,1: GOTO 50
120 IF A$="S" THEN DRAWR -1,0,1: GOTO 50
130 IF A$="W" THEN DRAWR -1,1,1: GOTO 50
140 GOTO 50
```

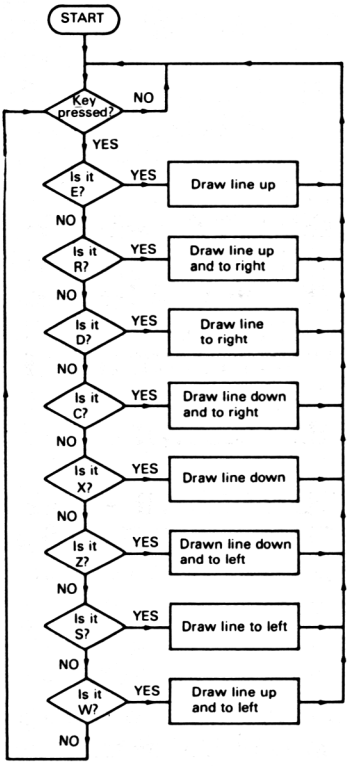


Figure 4.7 Flowchart for artist's drawing program.

This program allows any detailed shape to be drawn on the screen, but it produces a drawing as one long trace. This is because we have used 1 as the third number in all the DRAW instructions. But we can modify the program so that is can move the cursor without drawing if we amend the program slightly so that the third number can be changed. We choose the P key, only because it is at the opposite side of the keyboard to all the other control keys, to make the program switch between drawing lines and just moving the cursor when the other control keys are pressed. The amended program is:

```
10 CLS
20 INK 0,13
30 INK 1,19
40 PLOT 320,200
50 LET F% = 1
60 LET A$ = INKEY$
70 IF A$ = "P" THEN LET F% = 1 - F%: GOTO 60
80 IF A$ = "E" THEN DRAWR 0,1,F%: GOTO 60
90 IF A$ = "R" THEN DRAWR 1,1,F%: GOTO 60
100 IF A$ = "D" THEN DRAWR 1,0,F%: GOTO 60
110 IF A$ = "C" THEN DRAWR 1, - 1,F%: GOTO 60
120 IF A$ = "X" THEN DRAWR 0, - 1,F%: GOTO 60
130 IF A$ = "Z" THEN DRAWR - 1, - 1,F%:
    GOTO 60
140 IF A$ = "S" THEN DRAWR - 1,0,F%: GOTO 60
150 IF A$ = "W" THEN DRAWR - 1,1,F%:
    GOTO 60
160 GOTO 60
```

Line 70 causes the program to change its mode of operation when P is pressed by changing the value stored under F% from 1 to 0 or from 0 to 1. The way in which it does this is something of a programming 'trick'. When the value stored under F% is 1, the effect of LET F% = 1 - F% is to store 1 - 1, that is 0, under F. But if the value stored under F% is 0, the effect of this instruction is to store 1 - 0, that is 1, under F%. Since we begin by storing 1 under F% with line 50, each time P is pressed the value stored under F% swtiches between 1 and 0. In turn this affects all the DRAW commands, making them switch between plotting in the foreground colour (visibly) or in the background colour (invisibly). For this reason the program is also able to unplot, or erase, parts of the picture that have already been created.

It should also be noted that in both of the artist's drawing programs given above, we have placed two commands after the THEN in the IF – THEN instructions and that they are separated by a colon. Referring back to the Summary Box for making decisions will remind you that we may place a sequence of commands after 'THEN'. When we do this, the commands must be separated by a colon. In fact, we can place more than one command on any program line as long as a colon is used to separate them. But although this makes the program listing shorter it also makes it more difficult to read and is not recommended until a thorough familiarity with BASIC, and writing programs with it, has been acquired.

Other instructions for graphics

The Amstrad provides a number of instructions for graphics apart from those we have met already. They are given, with their purposes, in the following table. In this section, we shall examine a few other graphics.

The remaining graphics instructions

Instructions	Purpose of instruction
MOVER	Moves the graphics cursor to a new location relative to the previous position.
MOVE	Moves the graphics cursor to a new position relative to the origin.
ORIGIN	Moves the origin of the graphics window. Can also be used to set up a graphics window.
PLOTR	Similar to PLOT but works relative to the previous position of the graphics cursor.

SYMBOL, SYMBOL AFTER	Allows you to define your own characters rather than just using those defined when you switch on.
TAG, TAGOFF	Allows text to be written at the graphics cursor position. TAGOFF switches this function off.
TEST	Reports the colour of the screen at the current graphics cursor position.
WINDOW	Sets the size of the Text window.

We have already seen how to draw a square on the screen. The other common shape used is a circle. To draw a circle we can use quite simple maths.
The following program will draw a circle on the screen: –

```
10 CLS
20 FOR X = 1 TO 360
30 DEG
40 PLOT 320,200
50 PLOT 320 + 100*COS(X),200 + 100*SIN(X)
60 NEXT X
```

This circle has its centre at the middle of the screen (point 320,200) and a radius of 100. We have told the Amstrad that we are using degrees not radians.

If we wanted a spiral instead of a circle we should have to gradually reduce the radius. This can be done with the following program.

```
10 CLS
15 Y = 200
20 FOR X = 1 TO 360
25 IF X/10 = INT(X/10) THEN Y = Y – 1
```

```
30 DEG
40 PLOT 320,200
50 PLOT 320 + Y*COS(X),200 + y*SIN(X)
60 NEXT X
70 IF Y> 0 GOTO 20
```

This program has introduced a neat trick for making something happen every so often. The command INT only gives the whole number part. Thus INT(2.5) will give 2 while INT(3) will give three. So, in the program X/10 is only the same as INT(X/10) if X/10 is a whole number. This will only happen every ten numbers. The program has therefore told the computer to take one away from Y if X is divisible by ten.

TEST is used to examine a dot on the screen. It must be given the column and the row of the dot. It gives a number, which is the ink colour of the dot. It is very useful in games programs, for example to determine whether a missile strikes a target. We shall illustrate its use with a program that displays a random scattering of potential targets and a missile launcher, and then launches a missile when the F key is pressed. If the missile strikes a target then an explosion results, but if it passes harmlessly off the screen the program simply ends. A flowchart for this program is given in Figure 4.8.

The program generates the random positions for the targets by using RND. This generates at random a number from 0 up to, but not including, 1. Multiplying it by, say, 200 gives a random number from 0 up to, but not including, 200. For a row number, we need a whole number, that is an interger, since it does not make sense to talk about row ten-and-a-half. BASIC, as we have seen, provides us with INT for finding the whole number part of a number and INT (2.5), for example is 2. In this way, we can generate at random a whole number from 0

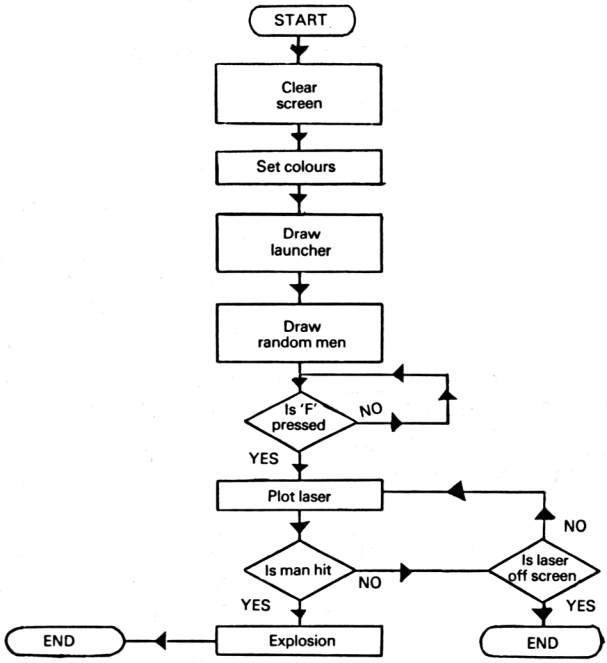


Figure 4.8 Flowchart for missile launcher program.

to 199 for use as a row number by INT(RND(1)*200). In similar fashion, if we only want row numbers from 50 to 189, we can generate them at random with INT(RND(1)*140) + 50. The program for this is:

```
10 CLS
20 INK 1,19
30 INK 0,3
40 PLOT 10,200
50 DRAWR 0, -10,1
60 DRAWR 10,5,1
70 DRAWR -10,5,1
80 FOR K = 1 TO 15
```

```
90 X = INT(RND(1)*440)+40
100 Y = INT(RND(1)*300)+50
110 TAG
115 PLOT X,Y
120 PRINT CHR$(250);
130 TAGOFF
135 NEXT K
140 ROW = 195
150 COL = 30
160 IF INKEY$="F" THEN GOTO 180
170 GOTO 160
180 IF TEST (COL+1, ROW) = 1 THEN GOTO 300
190 PLOT COL,ROW
210 COL = COL + 1
220 IF COL>480 THEN STOP
230 GOTO 180
300 INK 1,3,19 : INK 0,19,3
310 TAG
320 PRINT "X";
330 TAGOFF
340 END
```

The only new instruction in the program is STOP in 220, which when executed causes the computer to stop the execution of the program.

SOUND

The Amstrad can produce sounds that are very musical and flexible. Its sound generator has three channels, each of which can produce either a tone (a single note) or noise. Any channel or combination of channels can be activated. This means that it can play single notes, two-note and three-note chords,and combinations of notes and noise. The duration and volume of notes can be controlled, as can their envelope, that is, their attack — sustain — release pattern. With these facilities, the

Amstrad is a considerable music generator able to synthesise a wide range of sounds and to simulate various musical instruments and we will present a few programs using the SOUND instruction.

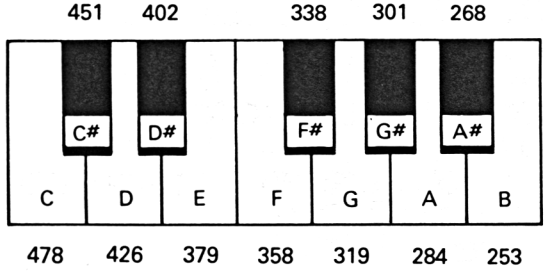


Figure 4.9 The notes of one octave on a piano keyboard and their values for use with SOUND.

When the SOUND command is used, it must be followed by between two and seven numbers. These give, in this order, Channel status, Tone period, Duration, Volume, Volume envelope, Tone envelope and Noise period. We shall only use 1 for the channel number.

Before going any further with generating sound on your Amstrad it is worth pausing to consider these seven numbers and the effects they can produce.

Channel status. The Amstrad has three different voices, so that it can play three notes at the same time. The first number after the SOUND command tells the computer which voice you are going to use. In addition, you can tell the Amstrad to use more than one voice, to hold, or to let two different voices rendezvous with each other. The numbers needed and their effects are shown in the following table.

Number	Effect
1	Send a sound to channel A
2	Send a sound to channel B
4	Send a sound to channel C
8	Rendezvous with channel A
16	Rendezvous with channel B
32	Rendezvous with channel C
64	Hold
128	Flush

To obtain an effect, just use the appropriate number. If the effect is to be a combination of the above, then add the numbers together. A couple of examples might be helpful.

4 = Send the following sound to channel C
3 = Send the following sound to channels A and B (1 + 2)
97 = Send the following sound to channel A, rendezvous with channel C and hold (1 + 32 + 64)

Tone period. The second number decides the frequency of the note. This number must be between 0 and 4095. For the technically minded, the freqency is given by:

Frequency = 125000/period

The above two numbers *must* follow any sound command. The remaining numbers can be left out, and if they are the default number, shown below, will be chosen by the Amstrad.

Duration. This tells the computer how long each note is to be, in hundredths of a second. If you do not put the number in the note will be 1/5 of a second long.

Volume. How loud is the note to be. Normally any number between 0 and 7, with 4 chosen as the default. The higher the number the louder the note.

Volume envelope. An envelope is something wrapped round a letter. In the same way this number wraps around your note. Notes usually change their volume as they play. This command allows you to have a note which starts off soft, gets louder, and then fades away. If you include a volume envelope number then the volume numbers can be between 0 and 15 with 12 as the default.

The envelope number must be between 0 and 15 and refers to an envelope you have made using the ENV command, explained below.

Tone envelope. Much the same as the volume envelope, but instead of changing the volume, this changes the tone as the note plays. This is what makes different instruments sound different. The number must be between 0 and 15 and refers to an envelope you have described using the command ENT, explained below.

Noise period. Specifies the noise to be added to the sound, unless you want a pure note. The number must be between 0 and 15, with the lowest number being no sound, and also being the default number.

ENV. This command tells the Amstrad to make an envelope for your volume, part of the sound command. It takes the following form

ENV N,P1,Q1,R1,P2,Q2,R2,P3,Q3,R3,P4,Q4,R4,P5, Q5,R5

The first number between 1 and 15 is the envelope number and is the same number as in your Volume Envelope section of the sound command.
There then follow up to 5 sections, as shown. You do not have to use all the sections but each has the following form.

P1 As you start the note you must go in steps, from quiet to loud and back again. This is just like going up the stairs. This first number tells the Amstrad which step you are on. Number from 0 to 127.

Q1 How big is each step? It could jump straight to the top (to loud) or be a very small step, followed by a larger one. Step sizes can be from −128 to +127.

R1 How long do you want to wait on each step? A number from 0 to 255.

ENT. This command is very like the ENT command, but for tone not volume. The command takes the form:

ENT S,T1,V1,W1,T2,V2,W2,T3,V3,W3,T4,V4,W4, T5,V5,W5

The sections are also similar to the ENV command as follows:

S	**Envelope number from 1 to 15**
T	**Step number from 0 to 239**
V	**Step size − from −128 to +127**
W	**Pause time from 0 to 255**

We will not be going deeply into the envelope commands, but if you wish to make your Amstrad sound like different instruments, it will be important to experiment with different envelopes. Try some at random, and every time you obtain a sound which you want to keep, make a note of all the envelope numbers you have used. One of the best places to learn good envelope commands from is games printed out in magazines. It is good practice to build up your own library of envelopes.

Now that we have seen the basic parts of the sound command, we will try and write some music to play on the Amstrad.

The frequency of middle C is 261.626. We can produce a note with a frequency of almost this by using a period of 478. The difference is 0.046 per cent and I doubt if many people could detect that difference!

The scale of middle C is given in the following table

C	478
D	426
E	379
F	358
G	319
A	284
B	253
C	239

The programs presented below take, perhaps surprisingly, the same forms as the graphics programs given earlier. First we can write a program to play a short sequence of notes, all of which are in the same octave giving:

```
10 READ M
20 FOR K = 1 TO M
30 READ N
40 SOUND 1,N,40
50 NEXT K
60 DATA 8,478,426,379,358,319,284,253,239
```

To change the tune we need only change the data at line 60.

```
60 DATA 11,319,319,319,284,253,284,
        319,253,284,284,319
```

However, all the notes are of the same length, and we should add a command to tell the Amstrad to alter the length. This can be added to the data command. In addition, you will have noticed that the first three and some later notes are slurred. We can overcome this problem by adding the instruction 'for one hundredth of a second play nothing'!

The finished program is:

```
10 READ M
20 FOR K= 1 TO M
30 READ N,D
40 SOUND 1,N,D
50 NEXT K
60 DATA 14,319,50,0,1,319,50,0,1,319,50,284,50,
      253,100,284,100,319,50,253,50,284,50,0,1,
      284,50,319,200
```

In English, the data line reads:

```
Play note 319 for 50/100 seconds
Play note 0   for 1/100  seconds
Play note 319 for 50/100 seconds
Play note 0   for 1/100  seconds, etc.
```

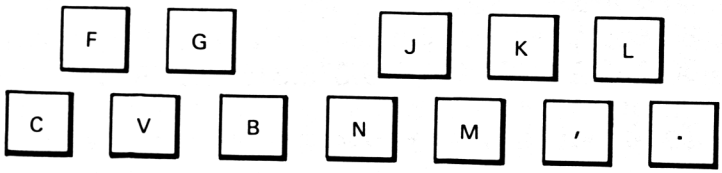


Figure 4.10 The keys on the Amstrad's keyboard used by the music-playing program.

We can convert a part of the Amstrad's keyboard to a stand-in for a piano keyboard so that we can play tunes on the computer by entering an appropriate program. We have chosen keys from the bottom two rows of the keyboard as shown in Figure 4.10 to play the notes of one octave. Referring to Figure 4.9 will show that the keys used start at C and have positions that correspond to those of the keys on a piano. The program will have the same 'shape' as the artist's drawing program, the flowchart for which is shown in Figure 4.7. The program is:

```
10 LET A$ = INKEY$
20 if A$ = "C" THEN SOUND 1,478
30 if A$ = "F" THEN SOUND 1,451
40 if A$ = "V" THEN SOUND 1,426
50 if A$ = "G" THEN SOUND 1,402
60 if A$ = "B" THEN SOUND 1,379
70 if A$ = "N" THEN SOUND 1,358
80 if A$ = "J" THEN SOUND 1,338
90 if A$ = "M" THEN SOUND 1,319
100 if A$ = "K" THEN SOUND 1,301
110 if A$ = "," THEN SOUND 1,284
```

```
120 if A$ = "L" THEN SOUND 1,268
130 if A$ = "." THEN SOUND 1,253
140 GOTO 10
```

For the first time it is important whether you use capital letters or lower case. The computer treats them differently, and will not accept that 'A' is the same as 'a'. If you type the above program as shown then make sure when you RUN it that you press CAPS LOCK.

Summary

The Amstrad has considerable facilities for graphics and sound. These are explained and the Amstrad's capabilities in these areas are explored in writing programs that incorporate them. While these programs are being developed, the opportunity is taken to introduce more of the features of BASIC as they are needed. In particular, the subroutine is introduced. This is something that is particularly important when writing lengthy programs as it allows them to be divided into parts which can be written separately and which can impose some structure on the programs. With the graphics instructions that have been introduced for creating shapes it is possible to create displays to almost any specification.

The instructions for creating sounds are also explained. There are fewer of them than for graphics, and some knowledge of music theory is necessary to appreciate to the full the ways in which they can be used. Programs that use these instructions are presented for playing music. Interestingly, they call on program structures which are the same as those used by the graphics programs.

Chapter 5

Applications for the Amstrad

There are plenty of programs available for the Amstrad that are ready to run and to make the computer do something useful for you. The most notable supplier of programs for the Amstrad is Amsoft. A lot of the programs that can be bought are for games, but in this chapter we are more concerned with programs that are for profit than those that are for pleasure. Amsoft supplies a program for word processing called Amsword, and a spreadsheet program called Amscalc. With a word processing program the Amstrad can be used to produce documents, a database program allows it to store information so that any item or items can be retrieved with ease, even when there is a great deal stored, and a spreadsheet is for planning or, indeed, any application where information is presented in tabular form. In this chapter we shall look at these three applications in general and examine the uses to which they can be put.

Word processing

When 'Amsword' is loaded into the computer from its cassette, the Amstrad is converted to a word processor.

At this stage the word processor is ready to use, and the user familiar with word processing will probably have little difficulty in proceeding to explore Amsword's capabilities. But we shall take time out to explain what word processing is all about. We shall also explain

many of the special terms that are used in word processing. This will enable those who are either new to word processing or with a limited experience of it to come to Amsword with a general appreciation of what it can do, and with a knowledge to the relevant vocabulary, to guide their expectations of how to use Amsword to advantage. It is worth reiterating the point that the act of loading a word processing program has converted the Amstrad from a general-purpose computer to a specialist word processor.

Word processing in general

Word processing is making use of the capabilities of a computer to store, process and print words. The computer has to do this in a way that satisfies any writer in applications ranging from writing a letter to creating a fair sized document to writing a book. Although the computer itself will still be performing much the same basic operations as when it handles numbers or creates graphics, loading a word processing program into it gives it the ability to recognise and handle words, sentences, paragraphs and pages in any text that it is given. It can then handle any of these as distinct units.

The computer does not understand the text that it is given in the way that a person would, and it would be overestimating the ability of a word processing program to think that it gave the computer this power. Such a misunderstanding could also lead to expecting more from a word processing program than it could ever deliver. A word processor simply provides the computer with some rules that allow it to recognise words, sentences and the other grammatical units. One rule, as an example, is that a number of consecutive letters followed by a space constitute a word, with the space indicating the end of the word. A second rule is that several words followed by a full stop make up a sen-

tence, although the word processor must also be aware that a question mark and an exclamation mark can terminate a sentence as well as a full stop.

A word processing program allows its users to type at the keyboard of a computer in just the same way as a typist does at a typewriter. The text that is typed is automatically stored in the computer's memory at the same time as it is displayed on the computer's screen. Because the text has been stored it is possible for the computer to process or manipulate it in any of a variety of ways. The particular ways that are available depend on the word processor that is used, but there are certain operations that are common to almost all of them.

A word processor automatically arranges for the text that is typed to be neatly laid out, both on its screen and when it is printed. The precise form of the layout can be prescribed by the user. Among the factors affecting the layout that the user can select, or change, are the length of the lines of text, the positions of the right and left margins, and the presentation at the beginning of a new paragraph. Many word processors display the text on their screens in the prescribed way as the text is typed. Later, when it is printed, using a printer attached to the computer, it will appear in exactly the same form. This means that the document being produced can be checked to ensure that it is perfect while it is on the computer's screen and before it is committed to paper. It can then be printed for the first time with the confidence of knowing that it is correct. To print the document, all that is necessary is to give the appropriate command to the word processor.

With other word processors, the text is not displayed exactly as it will be printed. This may be because the number of characters that can be displayed on one line of the screen is less that the number that is printed on a line on paper. Special characters, such as those to mark

the end of a paragraph, may be shown on the screen as special symbols although they will not be printed on paper when the document is printed: they will give a particular effect — in this case to cause a new paragraph to begin. In the end it does not really matter how a word processor displays its text, because the user will adapt to the way that it works and learn to take advantage of all its facilites.

If the user finds mistakes in the text when reading it on the screen, it is obvious that he or she must be able to correct them. Word processors allow corrections to be made to the text that has been entered by deleting, replacing or inserting letters so that simple errors such as spelling mistakes can be corrected. But they also allow whole stocks of text, such as a paragraph, to be inserted, deleted or even to be moved from one place in a document to another.

It is evident that text must always be typed, to enter it, before it can be handled by a word processor. The better the user is at typing, the faster documents can be entered and the more use will be made of the word processor. These seemingly obvious points are made to bring out the importance of the Amstrad having a 'proper' keyboard. Without it, touch typing would be impossible. As a second matter, although we are accustomed to thinking of documents as being printed on paper, when using a word processor there may be no need to use paper at all. Because the Amstrad can communicate via its EXPANSION socket a document can be passed from one computer to another in electronic form. Documents can equally well be exchanged after recording them on cassettes. Either way, documents can be passed to another computer for storage and display so that their contents can be communicated and read without committing them to paper at all.

By these means, the Amstrad together with Amsword provides the ability to produce perfectly correct documents and the means to communicate them, perhaps as electronic mail, without the need for paper.

Who needs a word processor?

Word processing makes the production of documents easier in various ways. It is interesting, and informative, to consider who can benefit by using a word processor and to examine the ways in which they can use a word processor to advantage.

The rapid and simple production of perfect documents is one major advantage of word processing that benefits all its users. All the corrections and amendments to a document that are needed can be made before it is ever printed. Word processors can also produce refinements that enhance the appearance of a document such as underlining words, placing headings centrally on a line, and placing the columns of a table in neat alignment. A letter can be arranged after it has been typed at the word processor, for example, to make it fit onto one page rather than running on to a second page that contains just one line to be followed by a signature. The word processor can make the rearrangement automatically if it is told to make the length of the lines in the document a little greater than they were previously. Anyone writing letters can benefit from this, but there are further benefits that can help many other groups of users, including students writing essays, authors writing books and businessmen producing reports and publicity releases.

A second very real benefit can be obtained by anyone producing quantities of letters that may all be slightly different, but which consist in the main of standard paragraphs drawn from a fixed repertoire. With the standard paragraphs stored by the word processor, any of

these letters can be produced simply by calling up the necessary paragraphs in the correct order and making a few insertions, such as the recipient's name, to personalise it. There is no longer any need to type each letter individually. The insertions can be made using the editing facilities of the word processor. Documents of this kind have to be produced routinely by, for example, lawyers, estate agents and insurance salesman.

The authors of magazine articles and books, and students who have to write essays, are among those familiar with the task of creating a polished text after producing several draft versions. This process involves writing a first draft, crossing out parts of it, inserting others perhaps by cutting and pasting bits of paper, and then rewriting the whole if it becomes too untidy or unreadable. This gives a second draft on which the whole process may be repeated to give a further draft, and so on. This can be very time-consuming, and if each draft is typed afresh, there is a danger of introducing errors into the parts that are already satisfactory. With a word processor, amending, revising and polishing a draft are all much easier. There is never any need to retype an entire document as the word processor's editing facilities are sufficient to allow one version to be converted to the next. The new version is always displayed at once. In this way, the current version of the document is always available and perfectly legible, but previous versions can be saved in case the editing is not done properly or its results are not satisfactory.

It will take anyone who is accustomed to working with hand-written drafts a little time to become accustomed to using a word processor instead. But the almost universal experience of those who have made the transition is that to use a word processor is a much more rapid and convenient way of creating a finished document.

A word processing session

We now describe a typical word processing session with a view to illustrating how it proceeds in practice. At the same time, we shall introduce some of the terminology of word processing.

On the assumption that you have some text to turn into a document, once your word processor is ready to accept text you can sit down and start to type it. The words that you type will appear on the screen as you type them. They are also being stored in the computer's memory. Nothing unexpected happens until a line on the screen is filled. Then the first word that makes the line too long to fit on the screen is automatically placed at the beginning of the next line. This is known as *word wrap*. It is worth stressing that there is no need to press ENTER at the end of each line as the equivalent of a carriage return on a typewriter: in fact, it is wrong to do so. The word processor manages the creation of the lines itself. At the same time as the new line is being started, some word processors can adjust the previous line so that its final word finishes exactly at the end of the line. This is done by inserting extra spaces to push the end of the line across to the required position, and it gives the document a neat vertical margin at the right of the page as well as at the left. The process is known as *justification*.

Continuing to type gives successive lines, all of which are treated in the same way. When the end of a paragraph is reached, ENTER should be pressed at the end of its final sentence. This causes the word processor automatically to create the gap between paragraphs and to indent the beginning of the next paragraph. This is part of the *formatting* of every document that is carried out by a word processor, as are the positioning of the right and left margins and the line spacing. At all times the document is displayed on the screen (although

not necessarily exactly in the form which it will be printed, as we have already explained) so that the user can see what he has typed.

The word processor provides a particular format for the documents it produces by default. If the user prefers a different format, any aspect of it can be changed after giving the appropriate *command* to the word processor.

When sufficient text has been entered, the word processor may indicate that the end of a page has been reached and start on a new one. The pages of the final document can be numbered: they can also be given a *header* , that is, a line of text to be placed at the top of each page as in the running title seen at the top of the page in many book. Sometimes it is also possible to give each page a *footer* , which is the same as a header except that it appears at the bottom of each page.

If words can be underlined or emboldened, to emphasise them, this is usually done in one of two ways. Either a special symbol is typed before and after the words in question, or a special mode is entered before the words are typed and is left afterwards.

On reaching the end of a document, or even after typing a certain amount of it, you will want to go back and check it to ensure that everything has been typed correctly, to see that there are no spelling mistakes, and that everything is to your liking. The process of correcting a document is known as *editing*, and before you can do it you must give the editing command. Editing is one area where word processing really comes into its own and shows to considerable advantage over using a typewriter.

If your document can be displayed on the screen in its entirety during editing, then you first move the cursor to the position on the screen where a change is needed by using the cursor movement keys. Then letters can be

deleted, inserted or replaced at this position by pressing the key that initiates the necessary action. Whole sentences and paragraphs can be deleted or moved just as easily. A display at the top of the screen provides a reminder of the letters that can be used during editing for all the different purposes. As soon as any editing operation is completed, the word processor re- formats the text to take account of the changes that have been made.

If the document is too large to fit on the screen, the screen acts as a window through which a part of the document can be seen. Pressing the appropriate keys causes the screen window to move up and down the document or, looking at it in another way, causes the document to *scroll* up and down beneath the screen. Since a part of a document can only be edited if it is being displayed, a large document is edited by first bringing the part to be changed into the display area and then proceding in the way just described. The position of the cursor along a line is usually indicated or a *ruler* line, which is a line that appears above the displayed text that also shows that character positions on a line, the positions of the margins and the tab stops.

When you have finished editing a document, you will want to save it or to print it, or even to do both. Either is done by giving the appropriate command. This is done by returning to the main command *menu*, or list of commands, and issuing one of the commands displayed there. A document can be saved on cassette or disc depending on whether you have a cassette player or disc drive attached to the computer for storage purposes. The document can be printed as long as a printer is attached to the computer.

There are several other commands that can be given to Amsword or, indeed, to any other word processor. You will probably find with any word processor that you

do not need all the commands that are provided. The ones mentioned already and a few others that satisfy your particular needs will usually be sufficient. But it is a good idea to be aware of all the commands that your word processor possesses, for if you are not you may not know that your word processor is perfectly capable of doing something that you need, especially when a new requirement emerges.

A glossary of word processing terms

This section provides a brief summary of the key terms encountered in word processing. Most of the terms were introduced in the previous section, but some others have been added for completeness. Not all the terms relate to features provided by Amsword, but then Amsword is not the only word processor available for the Amstrad, and it may be that you need facilities that Amsword does not have.

Back-up. A spare copy of a document that is recorded in case anything untoward should happen to the current document or if the results of editing the document prove unsatisfactory.

Centring. Automatically placing a heading or a line of text symmetrically in the centre of a line.

Command. Issued when in a special command mode, it tells the word processor to carry out one of the functions of which it is capable.

Deleting. Removing letters or words from the text.

File. The form in which a document is stored on a cassette or disc.

Footer. A fixed line of text appearing at the bottom of each page of a document.

Format. The way in which the text of a document is to be arranged by the word processor.

Header. A fixed line of text appearing at the top of each page of a document.

Help facility. Provides information on how to use the word processor and its commands so that assistance is always immediately available and there is no need to refer to a manual.

Insertion. The ability to insert text at any point in a document.

Justification. Justified text is arranged evenly at both the left and the right margins.

Line spacing. The vertical spacing between lines. Typing is usually double spaced.

Marker. A mark placed in the text which will not show when the document is printed and which serves to identify a block of text so that it can be moved, for example.

Menu. A list of items, such as commands, from which one may be selected.

Page break. Indicates to a printer that it must start to print on a new sheet of paper.

Ruler. A line usually placed above the displayed text on which margin and tab settings are shown, and with the aid of which they can be changed. It also shows the position of the cursor along a line.

Search and replace. A useful editing facility with the aid of which the word processor can search for occurrences of a specified word or phrase and, optionally replace them by another one.

Tabulation. Moving along a line to a fixed position (the tab stop). It is useful for setting out tables or in any work where vertical alignment is needed.

Word wrap. Placing words that will not fit at the end of one line at the beginning of the next one.

Summary

This section provides a general introduction to word processing. Although there is a good deal more to word processing than has been covered here, it should pro-

vide sufficient information for the reader to acquire an appreciation of the capabilities of word processing, and to proceed with confidence. A really detailed account of Amsword is, of course, provided with the program.

Databases

A database is an organised and integrated collection of data. It is also rather more than this, for a collection of data has no value unless we can make some use of it. This means that the data in a database must be organised in such a way as to allow us to do the things that we want with it. Operations typical of those that we might want to carry out with a collection of data are to select particular items of data from it according to some criterion, to search it for all the items that meet a given condition, to update items, and to sort the items into some special order. If a database if organised in this way, then from a collection of raw data we can extract useful, and even valuable information.

The data in a database is organised, essentially, by ensuring that all the items are stored in the same structured fashion. If possible, this should take advantage of any relationships between different items of data. A database becomes integrated by ensuring that all items are stored in the same way and also by avoiding any duplication of items. This allows access to any item of data in the database in a natural way by using its relations to other items.

Anyone can benefit from a database program, for we all have the need to store information and to retrieve it, whether in keeping records, running a business or looking after our finances. Typical personal uses are to keep the details of a record collection, a stamp collection or a collection of programs for the computer. Keeping records of all the financial transactions relating to one's annual tax return would also be possible. For educational purposes it could be used to keep an account of all the work that is due to be handed in, and of all the

topics taught on a particular syllabus. In a small business, or for that matter a large one, it could be used for stock control by recording the quantities of each item held in stock and updating them as necessary.

The ways in which the items in the database may need to be accessed will be broadly the same in all these cases, although in some instances particular operations may be needed more frequently than others. When using a database to hold the details of a record collection, a typical requirement might be to display all the details of a particular record, or to find the shelf location of a particular record. When used for stock control, the updating of the quantities held will be a common requirement, but if the levels at which goods should be re-orderd are also held in the database, then re-ordering can take place without fail if a stock level can be compared with the re-order level. A businessman who keeps the names and addresses of his customers in a database can print this information on envelopes to address them automatically. If other information is recorded about each customer, then it is posssible to select a particular group. By including their ages, all those under 30 can be selected for a special offer. By including the balance of their account it could be ensured that no more goods were supplied to any customers in the red.

From this discussion we can see that a database program has two distinct parts. The first must allow the data to be entered. The second must allow the user to examine and retrieve the data in any way that may be required. In the entry phase the database program should allow its users to structure their data in a way that is suited to the application. For the second phase the program should supply the facilities to enable the user to carry out all the necessary operations on the stored data. These operations include selection, searching

and sorting as we have seen and commands for these activities should be provided.

As the final point in this section, we can illustrate how a database gives considerable advantages over a collection of information recorded in a conventional fashion. Almost every house in the country contains a database, although it is printed on paper, in the form of the details of the week's television and radio programmes. This is a database in the sense that it is an organised and integrated collection of data about radio and television programmes. From the printed record it is easy to find the programme that will be broadcast at a particular time, not quite so easy to find at what time a given programme will start, and less easy still to find all the programmes of a particular kind that will be broadcast during the week and when they begin. But if the programme information were all stored in a database program, rather than being printed on paper, each of these selections would be equally easy to find. By giving the appropriate command to the database program it would retrieve the required information straight away.

Filing cards and a database

We begin by examining the way that information is recorded on filing cards and then recovered from a card box full of filing cards. This is because the way that a database program is used is entirely analagous to using a filing card system. Writing the information on the cards corresponds to storing the data in the database program. Retrieving information by examining the cards corresponds to recovering information from the database program. The idea is that by using the database program on a computer, the recovery of the information is made easier and quicker, particularly if there is a great deal of it.

To deal with a concrete example, suppose that we decide to create a file containing the week's television programmes, with the details of each individual programme recorded on a file card. We must first decide how to record the information about a programme on a file card. The details that we are likely to want are its title, its type, the day it is shown, the starting time and the channel on which it is being shown. We can now design a file card. When we have filled in a card for every programme, we shall have a box full of cards that records the week's programmes.

The information to be stored has been structured by deciding carefully which details to record. It is clear that other items can be placed on each card just as easily as the ones we have chosen, and that an item on each card can be crossed out if we are no longer interested in it. But the time spent on getting the design for the cards right in the first place is well spent. It is also obvious that a file card on which the details of a stamp in a stamp collection were recorded would not be stored in the same box as the cards containing television programme details. A separate box would be kept for these so that it could be filled with similar cards for each stamp and eventually record the entire stamp collection.

When a card for every television programme has been filled in and our box for them is full we have completed the data entry stage and completed our database. Now it can be examined to recover any information we want about the week's television programmes. We can find what is being shown at 8 o'clock on Tuesday evening by flicking through all the cards and noting those that give a match to our requirements under DAY and TIME. We shall find more than one card like this because there is more than one channel broadcasting programmes. We can find the time at which the news is shown by flicking through the cards to find

'news' under TYPE and then reading the entry under TIME. We can find what is on now by looking for an entry under TIME that matches the time now and reading the entry under TITLE. We can find how many sports programmes there are by counting the cards with 'sport' under TYPE. We could even decide to sort the cards into an order that suited us, perhaps with the programme titles in alphabetical order.

These operations, and particularly the sorting, will take a considerable length of time. At the least, we must rifle through every card, while re-ordering a large pack of cards to sort them into a new order could be a very lengthy affair. Although a computer database will operate in essentially the same way when it carries out the same operations, the computer's speed of operation ensures that everything is done much more quickly. In addition, once the data is entered, it is the computer that does all the work rather than us.

When using a database program, it first allows us to design, as the equivalent of a file card, a *record*. After this, the details of each record can be entered as the equivalent of filling in file cards. Each item of information within a record is known as a *field*. It corresponds to a single entry on a file card. When a record is entered, it is stored in the computer's memory. A collection of records stored in this way is called a *file*. A file is equivalent to a box of completed cards. A file can be stored permanently on cassette or disc. A database program can be used to create any number of files, so that we could create one file for television programmes, another for stamps and as many others as we needed.

Summary

This section has introduced the idea of a database and described the uses to which it can be put. All in all, this

section provides a general introduction to databases which shows in general terms how an individual's needs for storing and retrieving information can be met.

Spreadsheets and 'Amscalc'

By running a spreadsheet program, a computer provides its users with the electronic equivalent of pencil, sheets of paper and a calculator. A spreadsheet can be used for preparing tables of numbers, and for performing calculations on them, so that the effects of changing one value or another in the tables can be investigated. In this sense, a spreadsheet is an 'electronic worksheet'. The advantages it brings when compared to pencil and paper methods are speed, convenience and the ability to handle large amounts of data with ease. By allowing the rapid and direct investigation of many alternatives that can occur in a given situation, particularly in a complex one, the spreadsheet becomes an ideal tool for planning and forecasting.

The user of a spreadsheet program is initially presented with a blank sheet. The sheet provides a number of positions at which entries can be made, and these are arranged in rows and columns. Each position in this tabular arrangement is known as a 'cell', and our entry is made in a cell by moving the cursor onto that cell, typing the entry and pressing ENTER to show that it is complete. The entries can be numbers or text, and there is one other possibility, as we shall see. But by placing numbers or text in the appropriate cells a table such as the following can be prepared. Text provides headings and labels for the table and numbers the entries in the table itself.

A small table

Quarterly profits

	Sales	Costs	Profit
First quarter	1500	1000	500
Second quarter	1600	800	800
Third quarter	1400	800	600
Fourth quarter	2000	1100	900

Although a spreadsheet allows tables like this to be prepared rapidly and conveniently, this only begins to hint at its capabilities. What makes it much more useful is that a formula can be associated with any cell. When this is done the spreadsheet does not display the formula: it calculates a value from the formula and displays that. This ability helps even in preparing a table as small as ours, for the entries in the third column are the differences of those in columns one and two. So in this case the table could also be prepared by entering the formula 'cell in column 1 — cell in column 2' each time the cursor is positioned on the cell in the third column. We could also display the annual profit in a cell somewhere by placing the formula for the sum of the entries in column three in that cell.

When our table is prepared by placing numbers in the third column, we have a once-and-for-all table that serves to display the figures but can do nothing more. But if the entries in column three are formulae we have a much more useful table for when the number in one of the cells in columns one and two is changed, the entry in column three having this cell in its formula changes correspondingly. So will the total profit cell. Whatever figures are placed in the table for sales and costs, the

spreadsheet will automatically calculate and update the display of the quarterly profit, and of the total profit.

All spreadsheets have this property of re-calculating and updating the values of formulae whenever a change is made to a cell that is involved in a formula. This means not only that calculations are left to the spreadsheet, which can do them more reliably than us, but also that the spreadsheet can rapidly display any and all changes in a complex situation, whereas we might neglect some of them.

When our table is prepared by using formulae, it can be used to present the quarterly figures of any company in any year. All that is necessary is to enter the necessary figures in columns one and two. In this way it is a general model for a quarterly table. It can also be used to show the effects that result from having different figures in the columns where data must be entered. This illustrates, in a small way, that a spreadsheet is ideal for investigating the effects of changing data values. It will provide immediate answers to 'what if ?' questions about the data that are of the kind that need to be answered when planning ahead to prepare budgets, make forecasts and plan investment.

In fact, by using a spreadsheet, models of a wide range of activities can be created, including a company's operations, a proposed enterprise and an investment portfolio. The ways in which future activities may proceed can then be examined quantitatively, and coherent plans can be prepared based on an analysis of the possible occurrences. Creating a model requires a knowledge of the initial data items that are needed, but its essence lies in deriving the formulae to represent the situation. As an illustration, when modelling a business we might expect the fixed costs of the business and its income from sales to be provided as data items and, correspondingly, to be entered as data items on a spread-

sheet. The overall costs of the business may be given by the costs associated with making the items that have been sold plus the fixed costs. The overall costs can then be calculated from a formula, and can be entered on the spreadsheet in that way. The formula would give the overall costs as some fraction of the income from sales plus the fixed costs. The fixed costs, despite being calculated from a formula, can in turn be incorporated in a further formula for the profit.

When modelling any complex situation, the spreadsheet will have to support a very large table. It is impossible to display a large sheet on the screen in its entirety and in this circumstance the screen acts as a 'window' through which a part of the sheet can be seen. By moving the window, any part of the sheet can be seen.

An example of spreadsheet usage

In this section, we give an example to show how a spreadsheet can be used. It is quite a small scale example because it is not practical to develop a large one in a book. A spreadsheet really comes into its own with a large sheet, but a small example can illustrate the basic ideas behind constructing and using one.

The example deals with a personal investment portfolio. Anyone who has shares in, say, three companies can create the following work sheet to show their holding in each company, its current price and the current value of the holding. The holding is a number of shares, the price is expressed in pence and the value of the holding is in pounds.

This table can be entered as numbers and text only, but since column three depends on columns one and two it is preferable to enter formulae in column three. There is no alternative to entering the numbers in columns one and two, for the holdings and the prices represent the basic data.

Company	Holding	Price	Value
Sainsbury	200	276	552
ICI	500	602	3010
Amersham	250	268	670
		Total	4232

When a spreadsheet program presents its initial blank sheet, the columns are normally labelled with numbers and the rows with letters. The rows are labelled, from the top downwards, by AA to AZ, then BA to BZ and so on for as long as necessary towards ZZ. Similarly, the columns are labelled from left to right by 1, 2, 3 and so on. The user can fix the size of the sheet to be used by giving the number of rows and columns for it. Any cell can be identified by giving its row letters and column number. The cell at the top left is cell AA1, and the cell to its right is AA2. We can position our table in the following way:

	1	2	3	4
AA	**Company**	**Holding**	**Price**	**Value**
AB	Sainsbury	200	276	552
AC	ICI	500	602	3010
AD	Amersham	250	268	670
AE				
AF			Total	4232

To do this, we position the cursor on each cell in turn and type the appropriate text, number or formula.

Since the value of a holding is obtained by multiplying the number of shares held by the price per share, to give it in pence, and then dividing by 100, to convert it to pounds, we have the formula we need for the column headed 'Value'. In cell AB4 we must put the formula AB2 * AB3/100, in AC4 the formula AC2 * AC3/100 and in AD4 the formula AD2 * AD3/100. Since these formulae are all very similar, and this occurrence is typical in spreadsheet usage, there is usually a facility for copying them, to save typing each individually. The total is obtained by placing the formula AB4 + AC4 + AD4 in cell AF4.

This worksheet can be extended to allow us to keep track of the gains and losses on our share holding. By adding in column 5 a new set of entries headed 'Cost' we can show the original cost of each share (in pence). This will have to be entered as data. But then we can use a formula to show the gain in column 6. The formulae we need are, for AB6, AB4 – (AB2 * AB5)/100, for AC6, AC4 – (AC2 * AC5)/100, and for AD6, AD4 – (AD2 * AD5)/100. Again, the formulae show a pattern. We can show the total gain in AF6 with the formula AB6 + AC6 + AD6. The resulting table is:

	1	2	3	4	5	6
AA	**Company**	**Hold-ing**	**Price**	**Value**	**Cost**	**Gain**
AB	Sainsbury	200	276	552	251	50
AC	ICI	500	602	3010	599	15
AD	Amersham	250	268	670	242	65
AE						
AF			Total	4232		130

Now, as the share price changes or as our holding in a company changes, all we have to do is change the relevant data, and the spreadsheet automatically recalculates the other values in the table that alter in consequence. Of course, a completed sheet can be saved to be reloaded, into the computer later, saving us the trouble of entering it all over again.

Summary

This section provides an introduction to spreadsheets and their applications in general. A spreadsheet is an 'electronic worksheet' that can be used to display tables, and to investigate the consequences of changing any entries in the table. By allowing the investigation of various alternatives in a given situation, it is an ideal tool for planning and forecasting. Although Amscalc does not possess all the refinements of other spreadsheets it does provide the basic spreadsheet functions.

INDEX

USING YOUR AMSTRAD CPC464
by Garry Marshall

The aim of this book is to introduce the newcomer to the use of the Amstrad CPC464 computer. It is divided logically into three main areas: first, an introduction to the hardware; second an introduction to BASIC; and third an introduction to the software. The reader can learn how to use the Amstrad CPC464, quickly and easily and at the same time discover its unique capabilities.

Features of the hardware
Inside the computer
Expanding the computer
Features of Amstrad's BASIC
Graphics
Sound
Colour
Word processing on the Amstrad
Databases on the Amstrad
Spreadsheets on the Amstrad

£3.95

N A Newtech Production

ISBN 0-09-938800-6

00395

9 780099 388005

HCP

Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/