



Derek Ellershaw and Peter Schofield



# VERY BASIC BASIC

The first 15 hours on your

**AMSTRAD**





Derek Ellershaw and Peter Schofield

# **VERY BASIC BASIC**

**The first 15 hours on your**

**AMSTRAD**  
**(CPC 464)**



CENTURY PUBLISHING CO. LTD.

WITH

MELBOURNE HOUSE (PUBLISHERS) LTD.

Published in the United Kingdom by:  
Century Publishing Co. Ltd  
Portland House,  
12-13 Greek Street,  
London W1V 5LE  
in conjunction with  
Melbourne House (Publishers) Ltd.,  
Church Yard,  
Tring, Hertfordshire HP23 5LU  
ISBN 0 7126 0669 6

Copyright © 1985 by D. Ellershaw and P. Schofield

All rights reserved. This book is copyright. No part of this book may be copied or stored by any means whatsoever whether mechanical or electronic, except for private or study use as defined in the Copyright Act. All enquiries should be addressed to the publisher.

Reproduced, printed and bound in Great Britain by  
Hazell Watson & Viney Limited,  
Member of the BPCC Group,  
Aylesbury, Bucks

# CONTENTS

<b>SECTION 1</b>	<b>PRE-PROGRAMMING</b> .....	1
<b>CHAPTER 1</b>	INTRODUCTION .....	3
	Don't panic .....	3
	The first step .....	3
	How to use this book .....	4
	Before programming .....	4
	Basic techniques .....	5
	Useful aids .....	5
	More about programming .....	5
<b>CHAPTER 2</b>	KEYBOARD BASICS .....	7
	Before you begin .....	7
	The different modes .....	7
	Summary .....	9
	More keyboard hints .....	9
	Number keys .....	10
	Cursor keys .....	10
	More pre-programming hints .....	11
	Memory and screen clearing .....	12
	Command words .....	13
	The REM .....	13
<b>CHAPTER 3</b>	HOW TO CORRECT MISTAKES .....	15
	Correcting mistakes .....	15
	Correcting typing errors before pressing ENTER .....	15
	Correcting errors after pressing ENTER .....	16
	Correcting errors in a program which you are trying to RUN .....	17
	Using LIST .....	19
	Exercises .....	20
<b>CHAPTER 4</b>	USES OF THE PRINT COMMAND .....	21
	Print instruction .....	21
	Example .....	21

A better example .....	21
RENUM .....	22
Screen layout .....	24
LOCATE and position printing .....	24
Experiment yourself .....	25
The TAB command .....	25
Using speech marks .....	26
More use for TAB .....	26
Exercises .....	27

**SECTION 2      BASIC PROGRAMMING TECHNIQUES ..... 29**

**CHAPTER 5**

HOW TO USE INPUT .....	31
Getting information .....	31
Using INPUT .....	31
Let's go deeper .....	32
A working example .....	32
Another example .....	33
You can use the examples .....	34
Using INPUT for letters and words .....	34
A handy program for Xmas .....	35
The Xmas program explained .....	35

**CHAPTER 6**

NUMBERS AND YOUR COMPUTER .....	37
Using your computer as a calculator .....	37
Addition sign + .....	37
Subtraction - .....	38
Explanation .....	38
Multiplication sign .....	38
Division .....	39
Order of calculation .....	39
A program to print numbers .....	40
N is a numeric variable .....	40
Naming and giving a value to a numeric variable .....	41
What is allowed as a name .....	41
More uses for numeric variables .....	42
Explanation .....	43
Two new words — FOR and NEXT .....	43
To repeat a group of instructions .....	44
Consider this program .....	44
Explanation .....	44
The computer as a calculator .....	45
A tables program .....	45
A program for the ten times table .....	46



The RND function .....	46
RND is useful .....	47
A list of calculations is easily printed .....	48
Some useful formulae .....	48
Varying the level of difficulty .....	49
Different ways to use the computer .....	49
When the computer uses decimal .....	51
Raising a number to a power .....	51
Dealing with fractions .....	51
LET is optional .....	52

## CHAPTER 7

STRINGS AND THINGS .....	53
Introduction .....	53
How to recognise a string variable .....	53
An example .....	54
A use for the example .....	55
Adapting the example .....	55
Using the adaptation .....	56
Another use of a string variable .....	56
Another example — a quiz .....	56
String variables do vary .....	57
A true string variable explained .....	57
Using the quiz .....	57
A primitive word processor .....	57

## CHAPTER 8

SCREEN CLEARING AND TIMING .....	59
Introduction .....	59
To recap .....	59
Consider this program .....	59
Explanation .....	60
The untidy result .....	60
Another approach .....	60
A slower, more readable result .....	61

## CHAPTER 9

LOOPING AND BRANCHING .....	63
Using GOTO to repeat .....	63
Going round in a loop .....	63
GOTO branching .....	63
Other uses of GOTO and branching using IF-THEN .....	64
How branching and GOTO looping work .....	65
The END command .....	66
Over to you .....	66
Explanation .....	67
How branches and GOTO looping work .....	68

	Things to notice .....	68
	String variables .....	69
	More detail about END .....	69
	IF-THEN conditional statements .....	70
	ELSE — a better way .....	71
	Both = and <> can also be used with words .....	71
	Four more signs .....	71
	Example .....	71
	Explanation .....	72
	Two important words used with IF .....	73
	The meaning of AND .....	73
	An example of its use .....	73
	Logical operators can be used with words .....	74
	An example .....	74
	An example of the use of NOT .....	75
<b>CHAPTER 10</b>	SUBROUTINES .....	77
	Introduction .....	77
	When to use GOSUB .....	77
	For example .....	77
	Explanation .....	78
	An improved example .....	78
	Explanation .....	78
	Conclusion .....	79
<b>CHAPTER 11</b>	SUMMARY .....	81
<b>CHAPTER 12</b>	USING THE DATACORDER .....	83
	Amstrad advantage .....	83
	How to use pre-recorded tapes .....	83
	Saving programs on tape .....	83
<b>CHAPTER 13</b>	THE PRINTER .....	85
	Commercial advantage .....	85
	Consider before buying .....	85
	Commands for the printer .....	86
	Printing listings .....	86
<b>SECTION 3</b>	<b>INTRODUCTION TO COLOUR, SOUND AND GRAPHICS</b> .....	87
<b>CHAPTER 14</b>	MODES AND A FEW TIPS .....	89
	Three different modes .....	89
	An example using the three modes .....	89
	Screen clearing .....	90
	Something to notice .....	90

AUTO .....	91
Any sequence .....	91
Save typing time .....	92
Another time saver .....	92
Beware .....	93

## CHAPTER 15

COLOUR .....	95
Introduction .....	95
BORDER .....	95
PAPER — the screen background .....	96
More colours here .....	97
Flashing colours .....	97
PEN — the text colour .....	97
Experiment yourself .....	98
Example .....	98
INK .....	99
Example of the use of INK .....	99
Explanation .....	98
Computer language .....	100
Change PAPER colour .....	100
Change PEN colour .....	100
Switch-on state .....	101
Experiment .....	101

## CHAPTER 16

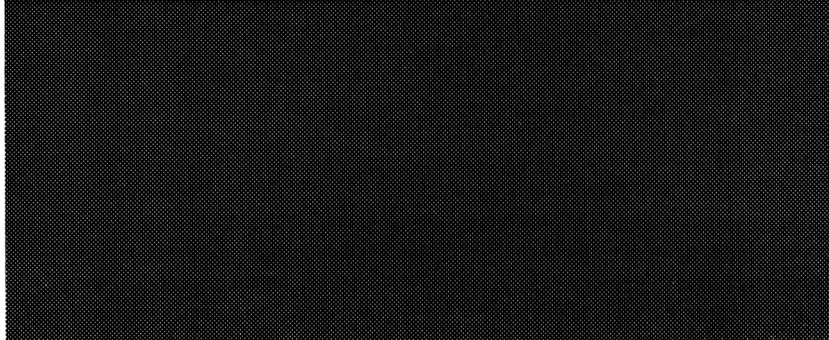
GRAPHICS .....	103
Amstrad character set .....	103
ASCII .....	103
Displaying the character set .....	105
Simple animation .....	104
It's up to you .....	104
Screen format .....	105
What is a pixel .....	105
Graphics cursor .....	105
Using PLOT and DRAW .....	106
Screen refinement using GOTO .....	106
Diagonal lines .....	107
Co-ordinates X and Y .....	107
Use for XPOS and YPOS .....	108
Don't forget to experiment .....	108
Round in circles .....	109
Radius circumference .....	109
Mathematical signs DEG, COS, SIN .....	109
Another way to DRAW circles .....	110
Colouring circles .....	110

	Create your own design .....	110
	The ORIGIN command .....	111
	A dash of colour .....	111
	Imagination .....	112
<b>CHAPTER 17</b>	<b>SOUND</b> .....	113
	High quality from the Amstrad .....	113
	Basic music .....	113
	Play a note .....	113
	Musical scale .....	114
	Sing along .....	115
	Sounds better .....	115
	Rules for the 'SOUND' command .....	116
	SOUND ENVELOPES .....	116
	Four rules for 'ENV' .....	117
	The ENT command .....	117
	'SOUND' effects .....	118
	Musical notes .....	118
<b>SECTION 4</b>	<b>MORE BASIC TECHNIQUES</b> .....	121
<b>CHAPTER 18</b>	<b>HOW TO HANDLE DATA</b> .....	123
	An example of READ and DATA explained .....	123
	How you go on .....	124
	Another example explained .....	124
	And it's over to you .....	125
	An example of arrays .....	125
	A different use for DATA .....	127
	Sorting letters .....	128
	Now you can experiment .....	129
<b>CHAPTER 19</b>	<b>HOW TO ADAPT PROGRAMS</b> .....	131
	Introduction .....	132
	A more complex program .....	132
	Explanation .....	133
	The program teaches as well as tests .....	133
	How to alter the program .....	134
	Another example .....	134
	How to adapt it .....	135
	Variations on a theme .....	136
	A test of logic .....	138



## **SECTION 1**

# **PRE-PROGRAMMING**



## CHAPTER 1

# INTRODUCTION

This introduction tells you how to use the book and how it is laid out.

By now you may have unpacked your Amstrad CPC 464, plugged everything in and perhaps have run a few games on it. You may even have tried some programs. If you are finding the manual difficult to follow, if you found that you understood the first few pages and little after that, then THIS IS THE BOOK FOR YOU. Don't feel ashamed! Don't burst into tears of frustration! Thousands before you have found computing difficult to understand. Very few people understand computers and programming from the very start. Many people feel that computing books go too far, too fast, too soon. This book is written for adults, teenagers, or any beginner. It avoids the use of jargon for its own sake and explains computing in English. New and technical terms are always fully explained.

This book does NOT aim to tell you everything about your Amstrad. It is very limited in its scope. It tries to guide you through the first hours and weeks of understanding, when confusion can often creep in. However, if you follow it carefully, it should provide a basis which will help you to understand the more complex books on the subject of programming.

**D**ON'T PANIC

**T**HE FIRST  
STEP

# HOW TO USE THIS BOOK

**(1)** Look carefully at the Sections and Chapters at the front of the book. They are set out very clearly because you will have to use them when you start to program. You will then be referred back or forward to another chapter if you want a more detailed explanation, but only do so if you really want an immediate explanation.

**(2)** You should ideally start at the beginning and work through the book unless the instructions indicate otherwise. The Chapters are arranged in a sequence. There are exercises in some Chapters. Make a real effort to attempt them to get the maximum benefit.

**(3)** The book is split into four main sections:

SECTION 1 PRE-PROGRAMMING

SECTION 2 BASIC PROGRAMMING  
TECHNIQUES

SECTION 3 INTRODUCTION TO COLOUR, SOUND  
AND GRAPHICS

SECTION 4 MORE BASIC TECHNIQUES

## BEFORE PROGRAMMING

### SECTION 1

This section outlines a number of things you need to know before you can begin programming. Yes, there are some simple programs but the main weight of the section is on helping you to be clear about using easy commands and knowing how to set out your programs.



## **B**ASIC TECHNIQUES



### SECTION 2

These chapters concentrate on the bare bones of programming techniques (for more information see Section 4). You are introduced to variables, looping, branching, counting, together with associated commands. We have aimed to make all the programs have a practical application, rather than make them theoretical exercises. We aim to help fathers and mothers, daughters and sons to discover how computers can help with education, leisure, home management and business.

### SECTION 3

These chapters are meant as an introduction only to some of the attractive features of your computer. Suggestions are made to incorporate these features in your programs.

### SECTION 4

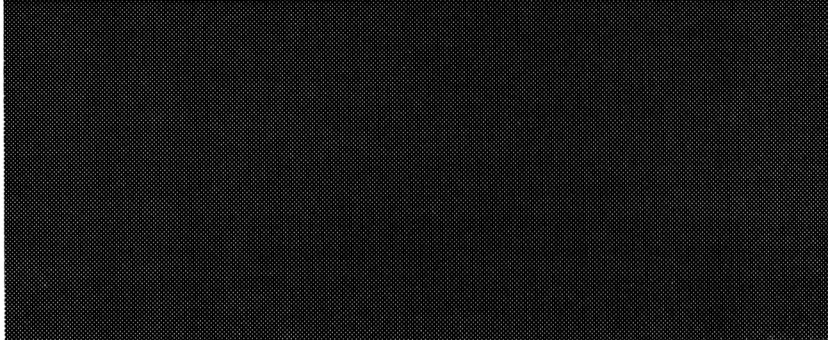
This section takes you a little further into programming and has some advice on program adaptation.

## **U**SEFUL AIDS



## **M**ORE ABOUT PROGRAMMING





## CHAPTER 2

# KEYBOARD BASICS

This chapter concentrates on clearing a number of obvious things out of the way so that you can begin programming.

It is possible to use the Amstrad CPC 464 with either an Amstrad colour or "Green screen" monitor, or a domestic T.V. You should therefore follow carefully the instructions in the User Instructions supplied with your computer to set up your equipment.

Once you have switched on you will see the message:

Amstrad 64K Microcomputer (v1) c 1984  
Amstrad Consumer Electronics Plc and  
Locomotive Software Ltd.

BASIC 1. 0  
Ready



This ■ is called a CURSOR. You are now ready to begin computing.

The computer is now READY to accept instructions from you — the CURSOR acts as a prompt — in other words it is encouraging you to type some instruction. Whenever you see the word READY followed by the cursor you know that the computer is waiting for you.

**B**EFORE YOU  
**BEGIN**

**T**HE  
**D**IFFERENT  
**M**ODES

Press the number keys on the top row of the keyboard and then try some letters of the alphabet. Now press the keys with the symbols ; and :. This shows that the computer is automatically locked into the small (lower case) letters. Now find the CAPS LOCK key. It is one of the green keys on the left hand side of the keyboard. Press it. Try the number keys again — still numbers. Try the symbols ; and : — still displaying the lower symbol. Now press some letter keys!!! When the CAPS LOCK key is pressed once all letters are displayed as capitals (upper case) but it is still possible to get the numbers and the lower symbols. To release the CAPS LOCK key just press it again. Now find one of the green SHIFT keys — there are two, one on the left and one on the right hand side of the main keyboard. Whilst holding down a SHIFT key press the number keys. This time the symbols above the numbers are displayed. Try some of the other symbols keys whilst holding SHIFT.

There is another key that can be used whilst typing. It is the green CTRL key which is next to the space bar on the bottom row. This key is the ConTRoL key and has a few functions. Return the keyboard to the lower case (small letters) mode. Now by holding the CTRL key down press the CAPS LOCK key once. Type some letters again — now try the number keys. This time you can get capital letters and the upper symbols — this can be useful when typing in some programs. To return to lower case letters and numbers hold down the CTRL key and press CAPS LOCK once. To return to capital letters you will have to press CAPS LOCK again.

Our screen is beginning to look a little cluttered — let's see if we can clear the screen so that we can start afresh. On the left hand side of the top row you will see a red key ESC. This is the ESCape key. Hold down the CTRL key and a SHIFT key and then press ESC. The computer is now Ready to start again!



To summarise so far:

- (a) We refer to the rectangle underneath Ready as a CURSOR.
- (b) The computer is set initially to lower case (small) letters.
- (c) With the CAPS LOCK key you can get Capital letters whilst retaining the numbers.
- (d) With SHIFT you can get Capital letters and the Upper symbols.
- (e) With CTRL and CAPS LOCK you can also get capital letters and the Upper symbols.
- (f) CTRL, SHIFT and ESC clears the screen and returns the computer to its initial state.

Practice using the keyboard — try the tape provided with your computer — it has a good keyboard practice exercise.

## MORE KEYBOARD HINTS



You will have noticed other keys with instructions written on them. Let us have a look at these. First of all clear the screen (CTRL, SHIFT and ESC). Type several letters and then press the key marked ENTER. You will see the following:

Syntax Error

followed by Ready and the CURSOR. The words "Syntax Error" mean that the computer does not understand the message you have typed. This is to be expected because we are only "playing" with the keyboard and not actually giving instructions. We can ignore the error message for now (it is explained in more detail later) and continue playing because we have the word Ready and the Cursor tells us that the computer is awaiting further instructions.

The ENTER key (sometimes called the RETURN key on other computers) is the most important key of all because it is the key that is used to ENTER the information typed into the computer. After typing any instruction line of a program the ENTER key must be pressed. You will soon become used to this.

## **N**UMBER KEYS



There are in fact two ENTER keys on the Amstrad — you will find the other next to the cluster of number keys to the left of the Datacoder. This ENTER key can also be used — it's obviously very handy if you have to type in a lot of numbers when the number cluster can be used more easily than the top row.

## **C**URSOR KEYS



Above the cluster of numbers you will find another cluster of keys. We will ignore the green key in the middle (COPY) for the time being as this is covered in the next chapter.

The four keys with arrows on control the position of the cursor. Clear the screen again (CTRL, SHIFT and ESC) and press the key with the down arrow . You will see the cursor move down the screen. Try the other arrow keys. If you hold one of these keys down you will see that the arrow moves rapidly in the direction indicated. This is known as AUTO-REPEAT and you will find that this facility is very useful in correcting typing errors (next chapter).

Experiment with the cursor control keys to familiarise yourself with the movements of each key.

We will explain the functions of the other command keys as we come across them in later chapters.

The following is useful as a reference list. On their own the points made may not seem to mean much at this stage but you may need to refer back to this list as you work through the next few pages until you become familiar with the workings of the machine.

Type this (clear the screen first CTRL, SHIFT, ESC)

Note ▲ means leave a space.

10 print ▲ 3

You will notice that the computer doesn't leave a space unless you do. Therefore, to leave a space, press the long space bar below the keys.

**(a)** When you first type a program line and it appears on the screen the information will not be retained in the computer's memory until you press ENTER. You must press ENTER to get each program line into the computer's memory.

Now press ENTER. You will notice that the cursor appears on the next line, it is telling you that it is expecting a further instruction. The computer is storing the previous instruction.

**(b)** You cannot get the computer to do anything with your information unless you type the word RUN. Until then it simply stores your instructions. Type RUN and then press ENTER.

3 appears beneath the word RUN because your instruction was: PRINT 3. PRINT is a command word. We will come across many more as we progress.

Congratulations! You have just RUN your first program! Note that we put 10 at the start of the line. All programs have their lines numbered e.g. 10, 20, 30, etc.

They are important for the computer to know the order in which it should carry out instructions.

**(c)** We number lines 10, 20, 30 rather than 1,2,3 in case we wish to insert additional lines. We can then use 11, 12, 13 etc. as line numbers. If we did not use this technique we would have to do a lot of retyping!

**(d)** Be careful not to confuse the number 0 with the letter O. They are not interchangeable. Similarly you must not confuse the number one (1) with the letter I on the keyboard nor the number one (1) with the small L (l).

**(e)** In this book, the symbol ▲ means make sure you leave a space. Spaces are important on the Amstrad — you will see why as we progress. To leave a space press the space bar once, or twice if ▲▲ appears — even if you do not see the sign it may be necessary to leave a space.

## **MEMORY AND SCREEN CLEARING**



You already know that you can clear the screen and memory by pressing the CTRL, SHIFT and ESC keys. This is fine but you may wish to start your programs on a screen which does not display the Amstrad 64K Microcomputer etc. message.

Type the letters  
CLS  
and press ENTER

That's better. CLS is another command word which means clear the screen — we suggest that whenever you start a new program, the screen is cleared completely in this way.

It is also possible to use the command word NEW or new (the Amstrad recognises both upper and lower case command words — more about this shortly) to clear memory but this does not clear the screen. Similarly the command word CLS or cls only clears the screen. It does not clear the memory. To clear both the screen and the memory you have to use both commands.

Type the following, pressing ENTER after each line:

10 print ▲ 3

20 print ▲ 4

Type RUN and press ENTER.

Now type CLS and press ENTER — the screen should be clear — if it's not start again. If it is clear type RUN and press ENTER. The screen should now be displaying the numbers 3 and 4 which proves that your program is still there.

Type:

NEW

and now press ENTER.

Now type RUN and press ENTER. This time the program has gone. Type CLS, press ENTER — the screen and memory are now both clear.

You will see that all command words in this book are shown in capital letters. You do not have to type them in capitals because the Amstrad recognises command words typed in lower case.

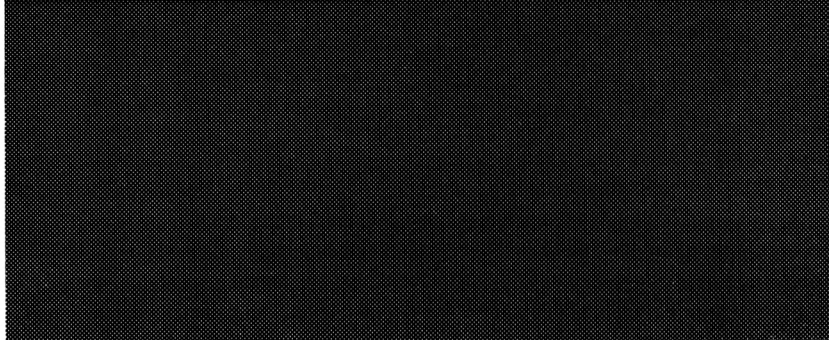
Finally in this section we would like to explain the command REM. It is important to remember that REM does not have a computer function nor does it affect the running of a program. Its purpose is to act as a REMinder or REMark for your benefit when writing or identifying your own programs.

## COMMAND WORDS



## THE REM





## CHAPTER 3

# HOW TO CORRECT MISTAKES

When you begin to write programs you will make mistakes; you may not spot these until you try to RUN the programs. Also, you will make simple typing errors. This short chapter shows how to correct these and other mistakes.

## CORRECTING MISTAKES

**(a)** Consider the following first line in a program. You are ready to press ENTER and you find a typing error. Type this line with the error, but don't press ENTER.

```
10 REM : This is tz show you how to  
correct mistakes.
```

**(b)** Find the green DEL key which is on the top row above the large ENTER key. Press it to see what happens — it rubs out or DEletes one letter at a time. Delete all the letters up to the mistake and then type the line again exactly as before — that is without the mistake. The DEL key used in this way is particularly effective if you realise that you have just made a typing error and want to correct it immediately. It is less useful if you have typed a long line before realising your mistake.

## CORRECTING TYPING ERRORS BEFORE PRESSING ENTER

## **CORRECTING ERRORS AFTER PRESSING ENTER**



**(c)** You should have the incorrect line displayed on the screen. If not, type it in again — don't press ENTER. This time we are going to use the CLR key which is next to the DEL key. First of all, by using the cursor control key, the one with the arrow → on it, place the cursor over the mistake. Now press the CLR key and the z will disappear. Type the correct letter (o) and your mistake is corrected. If you now move the cursor to the end of the line, you will see that the computer has also remembered to leave the space between the corrected letter o and the s of the next word.

**(d)** If you hold down the DEL key you will see that it also has the facility to AUTO-REPEAT — it keeps going backwards eating up your typing until you take your finger off the key. CLR (which stands for CLear) also works in a similar way except it appears to swallow up everything in front of it. Try it!

There will of course be many times when you have ENTERed a program line before noticing errors. Type and ENTER the following:

```
10 REM : Moor mistakes
20 PRINT ▲ " I luv"
30 PRINT ▲ "Compoeters"
```

To correct these mistakes we can:

**(e)** Retype a new line completely using the same line number. The computer will replace the old line with the new one. To see what we mean type and ENTER :

```
10 REM : More mistakes
```

Now to prove that this new line 10 has actually taken the place of the old one, type LIST and press ENTER. The LIST command does exactly as it says. It LISTS, your program (more about LIST shortly).



You can see that the corrected line 10 has taken the place of the old one.

(f) Another way is to use the command word EDIT. We are now concentrating on the mistake in line 20 so type:

EDIT▲20

Press ENTER and you will see that line 20 is displayed with the cursor covering the 2. You can now, by using the cursor control keys and either DEL or CLR, correct the line in the way described in paragraphs (b) and (c).

When you have corrected the mistake (I love) press ENTER. Type LIST again — press ENTER and you will see that we have now corrected 2 of the 3 lines. We could also correct line 30 by using EDIT 30 — this would bring line 30 out of the listing — but let's try another way.

(g) The green COPY key which is situated in the middle of the cursor control keys is another useful editing key. Hold down the SHIFT key and press the ↑ cursor key. You will see another cursor appear — this is the copy cursor. Position this cursor over the 3 in line 30. Now, whilst still holding down SHIFT, press the COPY key until it is over the second o in computers. The COPY cursor has copied the line exactly; now type u — hold down shift and with the cursor control key → position the copy cursor over the t — press the COPY key (don't forget to keep holding down the SHIFT key) until the line is complete. Press ENTER — type LIST — ENTER and the lines are correct.

## **CORRECTING ERRORS IN A PROGRAM WHICH YOU ARE TRYING TO RUN**

The mistakes which we have shown are spelling errors and would not have affected the running of the program but some errors may be programming errors — errors which stop the

program from actually running. Type these lines exactly as they appear — don't forget ENTER after each line:

```
10 PRINT ▲ "Hello"  
20 PRINT ▲ "I am"  
30 PRINT ▲ Your"  
40 "AMSTRAD"
```

Now type RUN and press ENTER.  
You were expecting the words

```
Hello  
I am  
Your  
AMSTRAD
```

to be displayed but in fact it hasn't quite worked out that way. Lines 10 and 20 have worked all right but line 30 has displayed 0 and we have been told that there is a Syntax error in 40.

The problem in line 30 has been caused because we have forgotten to put the "(speech marks) after the instruction PRINT (see next chapter for more detail). The Syntax error in line 40 is caused because we have forgotten the command word PRINT.

Let's correct these mistakes. Press ENTER, type LIST and press ENTER again.

Hold down SHIFT and press the cursor control key until the COPY cursor is covering the 3 in line 30. Now, using the COPY key, and holding down SHIFT, position the COPY cursor over the y. Now insert the speech marks and then COPY the rest of the line. When you have done this press ENTER. That's one line corrected. Now let's correct line 40.

This time type EDIT ▲ 40 and press ENTER. Now using the cursor control key position the cursor between 40 and AMSTRAD. Type PRINT and press ENTER. Your program should now look like this:

```
10 PRINT ▲ "Hello"  
20 PRINT ▲ "I am"  
30 PRINT ▲ "Your"  
40 PRINT ▲ "AMSTRAD"
```

RUN it to see that everything is all right.

Let's have a closer look at the command word LIST. Clear screen and memory.

## USING LIST

Type and ENTER the following lines (it is not a program so don't try to RUN it.):

```
10 A
20 B
30 C
40 D
50 E
60 F
70 G
```

You already know what will happen when you type and ENTER the command LIST. Try it. The computer has LISTed all the program lines. Now type:

```
LIST ▲ 30
```

Press ENTER. This time only line 30 has been displayed. Try:

```
LIST ▲ 30 - (minus sign on top row)
```

This time the computer has started at line 30 and listed all the lines from that point to the end of the program. Try:

```
LIST ▲ - 30
```

This time the listing has stopped at the first line and stopped at line 30. Now try:

```
LIST ▲ 30 - 60
```

Starts at line 30 — finishes at line 60.

This command is very useful as you go further and become involved with long programs. It enables you to list any line or block of lines for checking. It is also useful to know that you can interrupt a listing by pressing the ESC key — try it by typing LIST, pressing ENTER and then pressing ESC. Pressing ESC will stop the listing and once stopped can be continued by pressing the space bar.

Experiment with the commands that you have come across in this chapter.

## **E**XERCISES

(1) Type in the following lines as they appear and then correct each one before pressing ENTER. Remember to clear screen and memory before starting any new program:

```
10 REM : This is an inkorrect line
20 PRINT ▲ "I ave a computer"
30 PRINT ▲ "It is gud fun"
40 PRINT ▲ Computer
```

(2) Type the following, pressing ENTER after each line. Then using the COPY method, correct each line and then RUN the program:

```
10 REM : There is a mistake on every lane
20 "Correct each after each line has been ENTERed".
30 PRINT ▲ Hello again
40 PRINT ▲ "3+3=7"
50 PRINT ▲ This is the end
```

(3) Type the following, pressing ENTER after each line. Then using the EDIT method correct each line and then RUN the program:

```
10 REM : Wot more mistakes
20 PRINT ▲ I'm getting fed up of this
30 PRINT ▲ "Only one more lane"
40 PRINT ▲ Goodnight.
```

(4) Using the LIST command on the previous exercise, list

- (a) Line 20 to the end
- (b) Lines 20 and 30

## CHAPTER 4

# Uses of the PRINT command

## **P** RINT INSTRUCTION

The PRINT instruction is one of the most frequently used commands in programming. It is also one of the easiest to understand! You have already used it in Chapters 2 and 3.

Type this — don't forget ENTER after each line:

```
10 PRINT ▲ "John"  
20 PRINT ▲ "Smith"  
30 PRINT ▲ "35"  
40 PRINT ▲ "Acacia Ave"  
50 PRINT ▲ "ANYTOWN"
```

Now type RUN and ENTER. This simply shows how to print a list of words or phrases one below the other.

Note that each phrase is within speech marks.

## **E** XAMPLE

## **A** BETTER EXAMPLE

The following version looks neater:

```
10 PRINT ▲ "John Smith"  
20 PRINT ▲ "35, Acacia Ave"  
30 PRINT ▲ "ANYTOWN"
```

Look at these additional lines:

```
15 PRINT  
25 PRINT
```

Type them in and RUN the program again. The addition of lines 15 and 25 leaves a space between each line of the address — lines 15 and 25 don't PRINT anything, they are empty PRINT lines for display purposes. If you have a printer you could print out your own address labels from this program.

If you now LIST the program you will see that the computer has automatically put lines 15 and 25 in their correct positions. Here's another little tip — if you wish to keep your program lines numbered in tens for neatness type:

## RENUM

Press ENTER — now LIST the program again. Very handy! You could RENUMber them within any range you like — try:

```
RENUM ▲ 100, 100
```

Whilst we're on about tips and hints here's another. You will have noticed that we show our program lines:

```
10 PRINT ▲ "Hello"
```

We said earlier that a space must be left after a command word. This is not strictly true because it is possible to use a punctuation mark, e.g. ? , ; : " instead of a space. Type and RUN — (don't leave a space after PRINT this time):

```
10 PRINT"Hello"
```

Where were we?

Let's see how different layouts can be achieved simply by using commas and semi-colons and spaces. Type and RUN (where ▲ appears press the space bar):

```
10 PRINT ▲ 123
20 PRINT;123
30 PRINT ▲ 1 ▲ 2 ▲ 3
40 PRINT;▲ 123
50 PRINT; ; 1;2;3
60 PRINT; 1,2,3
70 PRINT ;1,2;3
80 PRINT"123"
```

Type RUN and press ENTER.

Interesting isn't it?

**(a)** Lines 10, 20, 30 and 40 all print the numbers together. Printing starts one space in from the left hand margin. Typing spaces in lines 10, 30 and 40 has no effect on the display.

**(b)** The semi-colons in lines 20 and 40 do not appear to have an effect, but they do ensure that printing starts at the first available position.

**(c)** The semi-colons in lines 50 and 70 however do have an effect. They ensure that, when handling numbers, sufficient space is left to separate them. Imagine the problem if spaces were not left!

**(d)** The commas in lines 60 and 70 tell the computer to leave 13 spaces (columns) after the first character printed before printing again. The screen is therefore effectively divided into 3 parts (more about this shortly).

**(e)** Line 80 is interesting. The speech marks make the computer print right up against the left hand margin.

**(f)** It has been explained that spaces (▲) used on their own have no effect on the display. Add this line

```
90 PRINT "1▲▲2▲▲3"
```

and RUN it.

What has happened this time? That's right, the spaces (▲) have been printed this time because they are within speech marks.

**(g)** Now add this line and RUN it:

```
100 PRINT "1,2,3,4"
```

Commas and semi-colons within speech marks have no effect on the display!

**(h)** That's enough to start with! Experiment yourself with different spacings. You can see how important being able to print in columns can be.

Type and RUN the following:

```

10 PRINT "What is 3+2?"
20 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T
30 PRINT "The answer is...."; 3+2

```

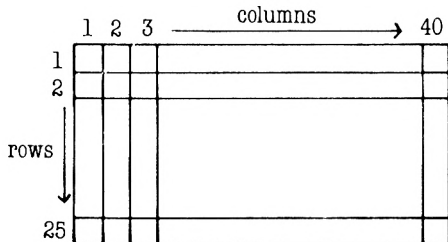
LINE 10 — PRINTS the question.

LINE 20 — waits for a short time (see Chapter 8 for explanation).

LINE 30 — PRINTS "The answer is...." and completes the calculation. Note that when 3+2 is written within speech marks on line 10 no calculation takes place but when 3+2 is outside the speech marks the computer works out the calculation. You will see that the answer is printed one character position away from "The answer is...." (like lines 10, 20, 30 and 40 in the previous program).

## SCREEN LAYOUT

You will of course want to PRINT something in an exact place on the screen and to do so you must give the computer the correct instructions. The screen in MODE 1 is split into 40 vertical columns numbered from 1 to 40 and 25 horizontal rows, numbered from 1 to 25.



## LOCATE AND POSITION PRINTING

The command used to give the computer these instructions is LOCATE. Clear screen and memory and then with CLS clear the screen completely.

Type and RUN:



```
10 LOCATE ▲ 20,12
20 PRINT "A"
```

You will see the letter A appear in the middle of the screen. It has been LOCATED at position 20,12.

20 is the column number  
12 is the row number

Change line 10 to read:

```
10 LOCATE ▲ 20, 20
```

Clear the screen (CLS) and RUN it again.

Experiment with this feature. Remember the first number is the column number, the second is the row.

There is another command word which is useful when used with the PRINT command — TAB. Type and RUN the following line:

```
10 PRINT ▲ TAB (20) "Hello"
```

It has printed "Hello" starting 20 columns from the left hand margin. TAB therefore, followed by a number from 1 to 40, will PRINT whatever is asked starting at the stated column number.

Type and ENTER the following:

```
10 PRINT ▲ TAB (20) "Hello"
```

```
20 LOCATE ▲ 20,12
```

```
30 PRINT "there"
```

```
40 PRINT ▲ TAB (20) "again!"
```

Clear the screen (CLS) before RUNNING it.

You will see that the word "Hello" has been printed at column 20 because of the TAB command in line 10. The cursor is then positioned at column 20 row 12 by the instruction LOCATE 20,12. The word "there" is then printed at position 20,12. Finally the word "again!" is printed on the next line TABbed across to column 20.

## EXPERIMENT YOURSELF



## THE TAB COMMAND



# USING SPEECH MARKS

Try these:

(a) 10 LOCATE ▲ 30,20  
20 PRINT ▲ 35

Clear the screen and RUN it.

(b) 10 LOCATE ▲ 10,20  
20 PRINT "35"

Clear the screen and RUN it.

(c) 10 LOCATE ▲ 20,10  
20 PRINT ▲ 7 \* 5

Clear the screen and RUN it.

(d) 10 PRINT ▲ TAB (35) "7\*5"

Clear the screen and RUN it.

Notice that the speech marks in (b) have not affected the display. As in (a) 35 has been displayed on the screen. However, look what happened when in example (c) we asked it to work out  $7 * 5$ . When we left the speech marks off, the calculation was completed and the answer, 35, given. When the calculation was placed inside speech marks, the computer printed exactly what was inside them, i.e. "7\*5"

Try this.

(e) 10 PRINT ▲ TAB (15) "John"

Clear the screen and RUN it.

(f) 10 PRINT ▲ TAB (15) John

Note what happens when you RUN these two lines. (e) prints "John" at column 15 but (f) prints  $\emptyset$ . This is because there appeared to be nothing to print because there was no speech marks around John.

WHENEVER YOU WANT TO DISPLAY LETTERS OR A COMBINATION OF LETTERS WITH A PRINT STATEMENT, YOU WILL HAVE TO USE SPEECH MARKS.

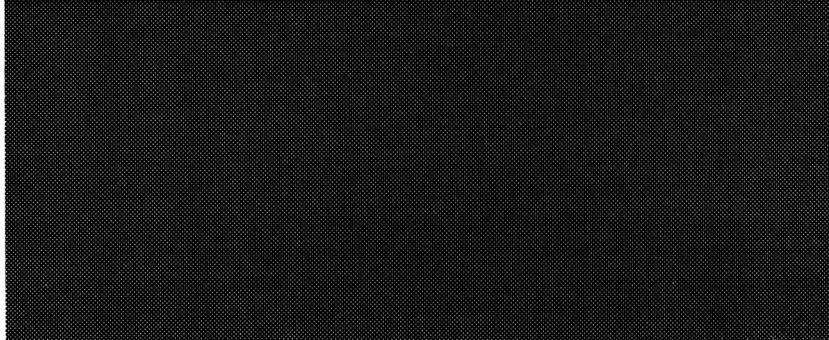
# MORE USE FOR TAB

One final thought about TAB. You can use more than one TAB command in a line; type and RUN:  
10 PRINT ▲ TAB (10) "Hi" TAB (30) "there"

## ***E*EXERCISES**

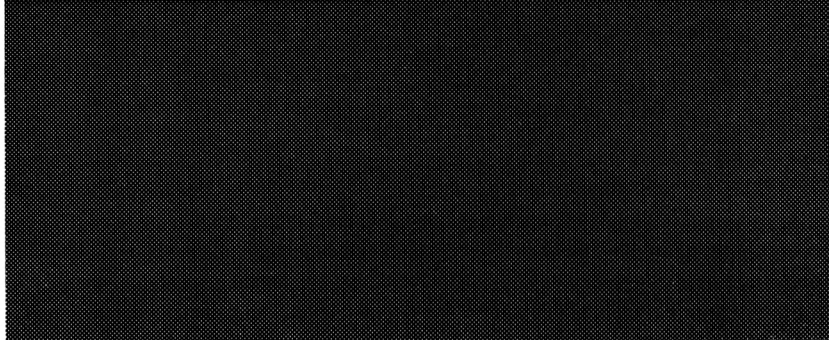


- (a) Write a program that PRINTS your name at three different positions on the screen.
- (b) Write a program using TAB, that prints the sum "6+6=" and on the same line but 15 columns further along prints "Answer 12".



## **SECTION 2**

# **BASIC PROGRAMMING TECHNIQUES**



## CHAPTER 5

# How to use input

## GETTING INFORMATION

Let us now look at the command word INPUT. This command is of great use as it enables you to INPUT or feed information into a program which is already running.

## U SING INPUT

Type this program into your computer:

```
10 REM : This is one way to print numbers.  
20 INPUT "What is your number "; N  
30 PRINT ▲ N  
40 GOTO ▲ 20
```

Type RUN and press ENTER.

You will see that the computer is asking you to give it a number. After the words "what is your number?" you will see the cursor. The computer is now waiting for you to INPUT any number of your choice — remember, it expects a number i.e. 1,2 etc. NOT one, two — it will not accept letters. Press 1 and you will see the number appear after the question mark. Now press ENTER. The number 1 has been printed on the screen and the question appears again.

The computer has followed the instructions in the program, i.e.

LINE 20 — it has asked you to INPUT a number.

LINE 30 — it has PRINTed that number.

LINE 40 — it has been sent back to line 20 — GOTO line 20 (GOTO is explained in more detail in Chapter 9).

## LET'S GO DEEPER

This program will continue for as long as the instructions are followed. To break into the program press ESC twice.

The simple program that we have just examined is not very useful in itself. It can, however, be adapted in many ways. Let us go a little deeper.

## A WORKING EXAMPLE EXPLAINED

Type, ENTER and RUN the following:

```
10 REM Mental Arithmetic Quiz
20 INPUT "what is your first number"; A
30 INPUT "what is your second number"; B
40 PRINT, " " A
50 PRINT, " " B "+"
60 PRINT, "—"
70 FOR▲T = 1 ▲ TO ▲ 6000 : NEXT▲T
80 PRINT, A+B" ▲ = ▲ ANSWER"
90 GOTO▲20
```

Remember the technique explained in the previous program. It also applies to this quiz.

The computer asks you for your first number — INPUT any number you like, but remember line 10 of the program! You don't want the computer to beat you yet! After INPUTing your first number you are asked for a second number. After you have pressed ENTER this time the computer will print the sum on the screen; you have approximately 10 seconds to answer before the computer answers for you. Did you get it right?

Let us study the program.

LINE 20 — asks you to INPUT a number

LINE 30 — asks you to INPUT another number

LINE 40 — PRINTS the first number

LINE 50 — PRINTS the second number

LINE 60 — draws a line.



LINE 70 — FOR T = 1 TO 6000 : NEXT T  
— acts as a stop watch — the time can be altered by making the number 6000 higher or lower as desired. (FOR-NEXT is explained in Chapter 6).

LINE 80 — PRINTS the answer

LINE 90 — GOTO 20 sends the program back to line 20 to start again.

Let us now look at another simple arithmetical program, the basis of which has many possibilities.

## ANOTHER EXAMPLE

Type and ENTER:

```
10 REM : ARITHMETIC
20 INPUT "what is the first number"; A
30 INPUT "what is the second number"; B
40 PRINT
50 PRINT ▲ A; "+" ; B; "="
60 PRINT
70 INPUT "My answer is ----"; C
80 PRINT
90 PRINT ▲ A; "+" ; B; "=" ; C
100 PRINT "Computer answer ----"; A+B
110 GOTO ▲ 20
```

RUN it. after INPUTting your two numbers as before, the sum is displayed in a slightly different form and you are now asked to INPUT the answer. Once you have answered the question the computer then displays the correct answer.

Program explanation.

LINE 20 — requests a number.

LINE 30 — requests second number.

LINE 40 — empty PRINT line (for display purposes).

LINE 50 — PRINTS the question.

LINE 60 — empty PRINT line.

LINE 70 — requests the answer.

LINE 80 — empty PRINT line.

LINE 90 — PRINTS the question again this time with your answer.

LINE 100 — PRINTS the correct answer.

LINE 110 — sends the computer back to line 20 to start again.

## YOU CAN USE THE EXAMPLES

The previous two programs are adaptable and can be incorporated in longer and more sophisticated educational programs. It is our aim to help and encourage you to develop your own.

## USING INPUT FOR LETTERS AND WORDS

We have seen how the computer accepts numbers. You will, of course, wish to INPUT letters, or more importantly, words. Clear memory and screen. You will remember the first simple program in this chapter which began:

```
10 REM : This is one way to print numbers.
```

Well we can change it slightly to meet our needs.

Type and ENTER.

```
10 REM : This is one way to print a line of words
```

```
20 INPUT "What word do you want printed":A$
```

```
30 PRINT▲▲A$
```

```
40 GOTO▲▲20
```

RUN it.

The words "what word do you want printed" followed by a question mark and the cursor will appear.

Type in your name and press ENTER. Now type in your full address, including the post code. You will see that the computer will accept numbers as well as letters, in fact it will accept whatever is INPUT except quotation marks. Break into this program as previously explained by using the ESC key.

## A HANDY PROGRAM FOR XMAS

Here is a program which, with the aid of a printer, will greatly assist with the chore of writing Christmas thank you letters. (You can, of course, display this on the screen, even without a printer.) Type and ENTER.

```
10 REM : Thank you letters
20 INPUT "Dear", A$
30 INPUT "Thank you for the"; B$
40 INPUT "It is very"; C$
50 INPUT "Best wishes"; D$
60 CLS
70 PRINT "Dear "; A$
80 PRINT
90 PRINT▲TAB (8) "Thank you for the "; B$
100 PRINT▲TAB (8) "It is very "; C$
110 PRINT▲TAB (8) "Wishing you a very happy New
    Year"
120 PRINT
130 PRINT▲TAB (12) "Best wishes, "; D$
```

RUN it.

You will see the word Dear appear on the screen followed by ? and the cursor. INPUT the name of the person to whom you are writing — don't forget to press ENTER. The first line of your letter will appear for you to complete, followed by the second and finally your name.

## THE XMAS PROGRAM EXPLAINED

The program works as follows:

```
LINE 20 — Asks you to INPUT the name
LINE 30 — asks for the description of the
           present
LINE 40 — asks your opinion of the present
LINE 50 — asks you to sign your name
LINE 60 — CLS — clears the screen
LINE 70 — prints the name
LINE 80 — empty print line
```

LINE 90 — prints the thank you line  
LINE 100 — prints your opinion  
LINE 110 — prints the greeting  
LINE 120 — empty print line  
LINE 130 — prints your signature.

You can of course, extend or adapt this type of program quite easily by adding more INPUT lines and corresponding PRINT lines. The layout can also be made more attractive by using LOCATE and TAB to greater effect.

## CHAPTER 6

# Numbers and your computer

## USING YOUR COMPUTER AS A CALCULATOR

Your computer can work as a calculator. Moreover, it will display calculations on the screen, show you if you have the correct answer and allow you to rectify any error.

The computer is quite capable of handling complex mathematical calculations but this chapter concentrates on handling some simpler functions.

Type and RUN:

```
10 PRINT ▲ 3+4
```

You will see 7 printed on the screen.

Type and RUN:

```
10 PRINT "3+4"
```

This time 3+4 is printed.

Type and RUN:

```
10 PRINT "3+4=" ";3+4
```

This time both the calculation and answer have been displayed.

(a) Therefore, if you simply want to see the result of adding two numbers, follow the first method.

(b) If you put the calculation inside speech marks, it will print whatever is within them and it will not complete the calculation.

## ADDITION SIGN +

(c) If you wish to print the calculation and its result then follow the last example.

## SUBTRACTION

Type and RUN the following:

```
10 PRINT ▲12-6
20 PRINT "12-6"
30 PRINT "12-6="; 12-6
```

As you see, subtraction follows the same rules. Type NEW to clear memory and then type:

```
10 PRINT ▲12-6+3+2
```

## EXPLANATION

RUN this and you will get the answer 11. This is because the computer follows the signs - + as they appear. You will see the importance of this in the paragraph about order of calculations.

Type this:

```
10 PRINT ▲8+7+6-3
```

RUN this and the answer 18 will be displayed for the same reason.

## MULTIPLICATION SIGN \*

The sign the computer uses for multiplication is \* not x. The rules are similar to the paragraph on Addition and Subtraction.

Type this:

```
10 PRINT ▲4*6
20 PRINT "4*6"
40 PRINT "4*6="; 4*6
```

Now RUN it.

## DIVISION /

The sign the computer recognises for division is / not the usual  $\div$ .

Type and RUN:

```
10 PRINT ▲ 10/2
20 PRINT "10/2"
30 PRINT "10/2= "; 10/2
```

## ORDER OF CALCULATION

Where a calculation involves mixing signs, the computer deals with them in a definite order. That is, it deals with \* and / before + and - NO MATTER WHAT ORDER THEY COME IN.

Consider the following:

```
10 PRINT ▲ 3+2+4*5
```

If you saw this calculation you might think the answer would be 45. The computer disagrees. RUN it. It has given the answer 25, because it works out  $4*5$  before performing the additions. If you want the computer to calculate in the order you have written it down, you have to use brackets, as follows:

```
10 PRINT (3+2+4)*5
```

The computer gives priority to the calculation within the brackets.

Let's move on a little.

# A PROGRAM TO PRINT NUMBERS

Type in the following:

```
10 REM : Program to print a list of numbers
20 LET ▲N = 1
30 PRINT ▲N
40 LET ▲N = N+1
50 PRINT ▲N
60 GOTO ▲40
```

Now RUN it.

You will see that the computer is printing a list of numbers in sequence down the left hand side of the screen — and it will continue to do so until it runs out of memory. Press ESC to stop the printing. Now press any key to continue. PRESS ESC twice when you have seen enough and read on.

LINE 10 — remember the computer ignores this line because it begins with REM.

LINE 20 — means LET N (the first number) = 1

LINE 30 — prints N — that is 1

LINE 40 — adds 1 to N to make it 2

LINE 50 — prints the new value of N — that is, 2

LINE 60 — sends the program back to line 40 which gives a new value to N which is one more than the old N — this time it will be 3.

The computer continues doing this and printing each number.

## N IS A "NUMERIC VARIABLE"

The N in the program is called a NUMERIC VARIABLE — because the value of N varies — first it is 1, then 2, then 3 etc. When we write LET N = 1, we talk about establishing a variable — saying what its value is.



The program actually works perfectly well without lines 20 and 30. Try it — delete the two lines 20 and 30 and RUN the program again.

## **N**AMING AND GIVING A VALUE TO A NUMERIC VARIABLE



This time the computer first puts its value to  $N$  —  $N = 0$ , immediately increases it by one, therefore the first number printed is 1. ( $N=20$   $8N+1$  or  $N=0+1$ ) we have included lines 20 and 30 because we feel that it is easier to understand the setting up of the variable.

## **W**HAT IS ALLOWED AS A NAME



Also you will have noticed that we have called the variable by a single letter. We can call a numeric variable by any letter of the alphabet from A-Z.

We can, if we wish, use more than one letter for our variable name — or a combination of letters, numbers or symbols.

There are some rules which must be obeyed when naming variables however:

- (a) Numeric variables must always begin with a letter.
- (b) Commas, spaces or speech marks are not allowed in variable names.
- (c) Words which the computer recognises as commands cannot be used on their own, but can be used within a phrase or longer word — for example:

```
10 LET ▲ PRINT = 6  
20 PRINT ▲ PRINT
```

is not allowed, but

```
10 LET ▲ SPRINT = 6
20 PRINT ▲ SPRINT
```

is allowed.

Here are a few examples; type and RUN them:

```
10 LET ▲ ABC = 3
20 PRINT ▲ ABC

10 LET ▲ AMSTRAD = 64
20 PRINT ▲ AMSTRAD

10 LET ▲ R2D2 = 1
20 PRINT ▲ R2D2

10 LET ▲ HELLO! = 6
20 PRINT ▲ HELLO!
```

We couldn't use:

```
LET ▲ 3D = 25 (it doesn't begin with a letter)
LET ▲ NEW = 60 (NEW is a command word)
LET ▲ HI, THERE = 5 (, commas are not
allowed)
```

## **MORE USES FOR NUMERIC VARIABLES**

We have demonstrated one use of a numeric variable in an earlier paragraph to print a list of numbers. It saves space in a program and in that particular one saves a long list of PRINT statements:

```
10 PRINT 1
20 PRINT 2
990 PRINT 99
```

Consider this program.

```
10 REM : controlling numbers
20 FOR ▲ N = 1 ▲ TO ▲ 15
30 PRINT ▲ N
40 NEXT ▲ N
```

Type and RUN it.

## ***E*EXPLANATION**

LINE 20 — sets the lower and upper limits of the list of numbers, that is, the lowest value of N will be 1 and the highest will be 15.

LINE 30 — prints the first number — 1.

LINE 40 — increments the value of the number by 1, that is 2.

LINE 30 — prints 2

LINE 40 — increments the value of the number by 1, that is 3, and so on.

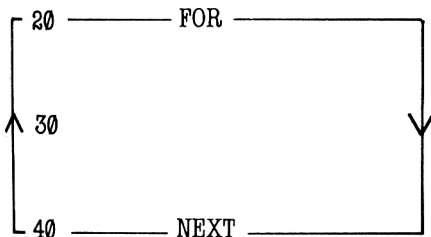
LINE 30 — prints the new number.

When the number reaches 15 the program has finished.

You will have noticed two new words FOR and NEXT. They are used together when you want to count in a program. The program LOOPS round lines 30 and 40, printing and incrementing the numbers until the upper limit set in line 20 is reached.

To summarise, the numeric variable is N, and N varies between 1 and 15.

This diagram may help you to understand FOR/NEXT loops.



There can be more than one line between lines 20 and 40.

## ***T*TWO NEW WORDS – FOR AND NEXT**

## **T**O REPEAT A GROUP OF INSTRUCTIONS

If you want to do something a certain number of times, you simply include the instructions as part of a FOR/NEXT loop.

## **C**ONSIDER THIS PROGRAM

Type and RUN:

10 REM : Program to demonstrate FOR/NEXT loop.

```
20 FOR▲N = 1 ▲ TO ▲ 10
30 PRINT "This is a FOR/NEXT loop"
40 PRINT "2x6=" ; 2*6
50 PRINT
60 NEXT▲N
70 PRINT
80 PRINT "The end"
```

## **E**XPLANATION

Lines 20 to 60 are the FOR/NEXT loop. The program runs through 10 times, hence FOR N = 1 TO 10.

Lines 30 and 50 are PRINT lines

Line 80 is executed after the FOR/NEXT loop has been completed.

When you use a FOR/NEXT loop in this way, it doesn't actually print the numbers 1,2,3 etc.

What would you add to the program if you wanted to actually number your PRINT statements? That's right, alter line 30 so that it reads:

```
30 PRINT▲N; "This is a FOR/NEXT loop"
```

If you really want this display to look better, change line 40 to read:

```
40 PRINT ▲ TAB (4) "2x6 = "; 2*6
```

and RUN the program again.

# THE COMPUTER AS A CALCULATOR

Remember that we showed you how to use the computer as a calculator. Instead of typing in each calculation, the program can be written as follows:

```
10 REM : Using Numeric Variables
20 LET ▲ N = 1
30 FOR ▲ C = 1 ▲ TO ▲ 9
40 PRINT "What is "; N; "+" ; C
50 PRINT
60 LET ▲ N = N + 1
70 NEXT ▲ C
```

Type and RUN it.

LINE 20 — sets variable N at 1

LINE 30 — sets the lower and upper limit of the FOR/NEXT loop.

LINE 40 — prints the sum as follows  
what is N (1) + C(1)

LINE 60 — adds 1 to the value of N  
(N=N+1)

LINE 70 — increments the value of C by 1.

If you do not quite understand this, delete line 20 and RUN the program again. You will see that the value of N is then set by the computer at 0 and therefore each increase in the value of N will always be 1 less than C. Remember that the value of C is set at 1 in line 30.

# A TABLES PROGRAM

## A PROGRAM FOR THE TEN TIMES TABLE

This program asks the ten times table:

```
10 REM : MULTIPLICATION
20 LET▲N = 10
30 FOR▲C = 1 TO 12
40 PRINT "What is"; C; "*" ; N
50 PRINT
60 NEXT▲C
```

Compare with the previous program.

LINE 20 — sets N at 10 because this is the ten times table.

LINE 30 — sets the limits of C from 1 to 12.

LINE 40 — the variables have changed places and the + sign becomes \*.

LINE 60 — has been omitted because the value of N must remain at 10.

LINE 70 — becomes new line 60

Now RUN it.

Clever — change line 40 to read:

```
40 PRINT "What is ";C;"* ";N;TAB(25)
"Answer"; C*N
```

Run it again.

Remember TAB — very useful, but the really effective part of the line is the calculation  $C * N$ . You can see the strength of numeric variables here — the value of C is incremented each time by the FOR/NEXT loop and the new value used in the calculations!

## THE RND FUNCTION

Let's now look at another useful command — RND, which instructs the computer to choose numbers at RaNDom. This can be very effective in many types of programs especially those dealing with mathematics.

Type and RUN:

```
10 PRINT ▲ RND  
20 GOTO ▲ 10
```

Use the ESC key to stop the program and you will see that the computer has generated numbers between 0 and 1. Press any key to continue the program. When you have seen enough, press ESC twice.

We will not attempt to explain how the computer produces these numbers except to say that they are not completely random but present in random sequences within the computer's memory. Producing numbers with so many decimal places is not really much use to us here so let's introduce another command — INT.

Using the COPY cursor control keys, change line 10 to read:

```
10 PRINT ▲ INT (RND)
```

RUN it again and this time you will see that a row of 0s has been printed. RND has this time chosen numbers between 0 and 1 but the command INT (INTEGERISE) has rounded them down to 0.

Try this:

```
10 PRINT ▲ INT (RND) + 1
```

RUN it and you see a line of 1s. Again the sequence of numbers generated by the computer was between 0 and 1, but INT rounded each down to 0 and finally added 1.

Try:

```
10 PRINT ▲ INT (RND) + 2
```

RUN it to satisfy yourself that the same principle applies. Remember INT always rounds down.

We are now in a position to make RND and INT work for us. Let's see what we can do.

Type and RUN.

**RND IS  
USEFUL**



```
10 REM : Random numbers between 1 and 10
20 LET ▲ N = INT (RND (1) * 10) + 1
30 PRINT ▲ N
```

Or, for random numbers between 1 and 100.

Change line 20 to read:

```
20 LET ▲ N = INT (RND (1) * 100) + 1
```

You should have the idea now so let's try a simple program.

## **A** LIST TYPE OF CALCULATIONS IS EASILY PRINTED

Type the following:

```
10 REM : Program to show use of RND
20 LET ▲ N = INT (RND (1) * 10) + 1
30 LET ▲ C = INT (RND (1) * 10) + 1
40 PRINT "What is"; N; "+"; C; TAB (25) "Answer"; N + C
50 PRINT
60 GOTO ▲ 20
```

RUN it. Use ESC to see what is happening.

LINE 10 — chooses a random number between 1 and 10 for the numeric variable N.

LINE 20 — chooses another random number between 1 and 10 for the variable C.

## **SOME** USEFUL FORMULAE

The following are useful formulae for selecting certain random numbers.

Numbers between 1 and 15 inclusive.

```
10 LET ▲ N = INT (RND (1) * 15) + 1
```

Numbers between and 1 and 100 inclusive

```
10 LET ▲ N = INT (RND (1) * 100) + 1
```



Numbers between 10 and 100 inclusive

```
10 LET ▲N = INT (RND (1) * 91) + 10
```

Numbers between 100 and 1000 inclusive

```
10 LET ▲N = INT (RND (1) * 901) + 100
```

## VARYING THE LEVEL OF DIFFICULTY



Once you are able to choose which numbers to print, you are then in a position to vary the level of difficulty of your calculations. For instance, if you are just beginning to learn multiplication, then the numbers you choose might be between 1 and 10. If you are more advanced then you might want long multiplication, say numbers between 99 and 1000.

## DIFFERENT WAYS TO USE THE COMPUTER



Remember that you can use the computer in different ways:

(i) to list questions — which you might answer yourself on paper.

(ii) to ask a question, using the timing statement to give you the answer after you have again worked it out on paper or in your head. (Chapter — Timing).

(iii) to INPUT answers into the computer and then be told if you are right or wrong, perhaps after giving a specified number of attempts (Chapters — Looping and Conditioned Statements).

(iv) If you have a printer, you can print worksheets.

(v) You can also set out your calculations attractively by using the simple techniques already learned. The following program is for long multiplication.

```

10 PRINT "Long Multiplication" TAB
(26)"Answer"
20 PRINT▲TAB (15) "_____"
30 PRINT
40 FOR▲D = 1▲TO▲12
50 LET▲A = INT (RND (1) *900) + 100
60 LET▲B = INT (RND (1) * 90) + 10
70 LET▲C = A * B
80 PRINT "▲▲▲▲▲"; A
90 PRINT "▲▲▲▲▲*"; B
100 PRINT "▲▲▲▲▲_____" TAB (5); D
110 PRINT "▲▲▲▲▲_____"
120 PRINT "▲▲▲▲▲_____"
130 FOR▲T = 1 ▲TO▲10000 : NEXT▲ T
140 PRINT "▲▲▲▲▲_____" TAB (26); C
150 PRINT
160 NEXT▲D

```

This is what the screen looks like:

```

Long multiplication _____ Answer
980
*53      1
                                     51940
579
*11      2
                                     6369

```

etc. until twelve calculations have been printed. Note that the timing statement in line 130 only gives about 10 seconds. If you need more time to work out the sum, change 10000 to 30000 or even 60000.

## WHEN THE COMPUTER USES DECIMAL

---

At this stage, if you are attempting division calculations, you cannot guarantee that the answer will be in whole numbers. The computer always uses decimals if you have a calculation like this:

```
10 PRINT ▲ 5/2
```

Type and RUN it.

## RAISING A NUMBER TO A POWER

---

Next, in this chapter about numbers, a brief word about the sign  $\uparrow$ . This is the computer's way of showing that a number has been raised to a certain "power". For example, if you type in

```
10 PRINT ▲ 3 ↑ 2
```

and RUN it, you will get the answer 9 because  $3 \uparrow$  means  $3 \times 3$  or 3 squared.

Equally:

```
10 PRINT ▲ 2 ↑ 8
```

means  $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$ .

How many do you think this is? Try it!

## DEALING WITH FRACTIONS

---

Your computer will also deal with fractions but you must set it out as either:

```
10 PRINT ▲ 2+1/2 + 3 + 1/2
```

```
10 PRINT (2+1/2) + (3+1/2)
```

which is the way to ask it to add  $2\frac{1}{2} + 3\frac{1}{2}$ .

## **L**ET IS OPTIONAL

One final thought. The command word LET does not HAVE to be used, for example:

```
10 LET ▲ N = 1
```

can be written

```
10 N = 1
```

Throughout this book we have continued to use LET. We feel this is clearer for beginners. Similarly, we have not shortened any other commands.

## CHAPTER 7

# Strings and Things

We have already explained that “numeric variable” is the term given to a name representing a number. “String variable” is the term given to a name representing a word or collection of words.

## INTRODUCTION

(a) You will always be able to recognise a “string variable” because it has the \$ sign written after it. The \$ sign is on the 4 key.

(b) A “string variable” MUST begin with a letter and always end with the \$ sign. The variable can be of almost any length, from a single letter upwards (up to a maximum of 255 characters) e.g. A\$, B\$, ABC\$, HELLO\$.

(c) For the purposes of this book the value of the string variable (what it is equal to) must be typed within speech marks. A STRING is simply several letters, graphics or numbers. That is, the computer will print anything that is within speech marks, including numbers. It does not have to be of any particular length, for example:

```
LET ▲N$ = "John"
```

This is called establishing, or declaring a variable. The LET command is used as it was with numeric variables. You could also type the string as follows:

## HOW TO RECOGNISE A STRING VARIABLE

```
LET ▲NL$ = "John"  
LET ▲NLRP$ = "John"
```

(d) The computer will not accept command words on their own but they are acceptable within a longer variable name. For instance:

```
10 LET ▲PRINT$ = "Hello"  
20 PRINT ▲PRINT$
```

is not acceptable but

```
10 LET ▲PRINTER$ = "Hello"  
20 PRINT ▲PRINTER$
```

is. Try both of them.

The first example as you see results in a Syntax error, because the computer recognises the variable as a command. Consult your manual for all command words.

Here is a program to show a use of string variables on their own. (For simplification, all string variables are shown as single letters followed by the sign \$) Let us imagine that you wanted to display on the screen a number of items on a weekly shopping list. Your program might read as follows:

## ***A*** **EXAMPLE**

```
10 REM : Shopping List  
20 LET ▲J$ = "Jam"  
30 LET ▲C$ = "Coffee"  
40 LET ▲P$ = "Potatoes"  
50 LET ▲S$ = "Sugar"  
60 LET ▲M$ = "Milk"  
70 PRINT ▲ J$, C$, P$, S$, M$
```

Type and RUN this program. You will simply get a list of five things you may wish to buy.

## USE FOR THE EXAMPLE

However, if you only want to buy two of the items this month, you would simply alter line 70 to read:

```
70 PRINT ▲ J$, S$
```

This particular program might not seem to have much value until you consider say the 30 or so items which might make up a household shopping list. If you SAVED a program such as this on tape and, at the start of each week, scanned the program to see what you needed and then altered the last line, you would be able to print on the screen your requirements for that week. (See Chapter 12 for SAVE.) If you had a printer it would then be a moment's task to print out the list. No more forgotten items at the supermarket!

## ADAPTING THE EXAMPLE

The program would need very little adaption to print out the cost of your list, by using numeric variables as well as string variables. Look at this adapted program:

```
10 REM : Shopping list
20 LET ▲ J$ = "Jam" : LET ▲ J = .40
30 LET ▲ C$ = "Coffee" : LET ▲ C = 1.20
40 LET ▲ P$ = "Potatoes" : LET ▲ P = 2.00
50 LET ▲ S$ = "Sugar" : LET ▲ S = 1.60
60 LET ▲ M$ = "Milk" : LET ▲ M = 5.00
70 PRINT ▲ J$, C$, P$, $, M$
80 LET ▲ T = J+C+P+S+M
90 PRINT
100 PRINT "TOTAL £ " ; T
```

Note that in line 20 it is perfectly possible to have a string variable and a numeric variable providing there is a colon (:) between the LET statements.

RUN this program. Now before RUNNING it again, delete J\$ and C\$ from line 70 and J and C from line 80.

## USING THE ADAPTATION

You will see that you now have two different lists and two different totals. You could use this as a guide to see how much money you would need when you go shopping. Also, if you only had £20 and your list came to say, £25, you could try different combinations to see which would be the best way to spend your money.

## ANOTHER USE OF STRING VARIABLES

One of the uses of "string variables" is to save typing the same words time and again.

This is particularly useful in a long program where you might want to use a particular string variable several times.

## ANOTHER EXAMPLE – A QUIZ

Here is another example of string variables used in a quiz:

```
10 REM : Kings and Queens
20 LET A$ = "William I"
30 PRINT "Who was King of England in 1063"
40 INPUT "My answer is"; B$
50 PRINT "Your answer is" ; B$; "The correct answer is
";A$
60 END
```

Type and RUN this. Type RUN to RUN it again.



## **STRING VARIABLES DO VARY**



One thing that often puzzles people is that string variables do not seem to vary! For example, in the previous program `A$ = "William I"` — `A$` however did not have to equal `William I` — it is only equal to `William I` because we gave it that value!

## **A TRUE STRING VARIABLE EXPLAINED**



Now look at `B$` — this, if you like, is a true string variable because it is not equal to anything until you give it a value. You could give it the value `Elizabeth I` or `George III` — its value could vary depending on your answer, what you `INPUT`.

## **U SING THE QUIZ**



We will develop this quiz in a later chapter to show how it can be adapted to help with homework and revision.

## **A PRIMITIVE WORD PROCESSOR**



Let's now turn to look at a primitive word processor. This program permits you to tell a "story" using some of your own words for the characters and places. It is possible to get books printed with a child's name appearing in it as the hero all the way through. Perhaps this is how it's done:

```
10 REM : This shows the use of string variables in a
story.
20 PRINT "I went to see T$ and then we met K$ before
going to the Y$"
30 INPUT "My word for T$ is ";T$
40 INPUT "My word for K$ is ";K$
50 INPUT "My word for Y$ is ";Y$
60 PRINT
70 PRINT "I went to see ▲ ";T$;" ▲ and then we met ▲
";K$;" ▲ before going to the ▲ ";Y$
80 END
```

If you have young children you could use program such as this to build vocabulary. limits are only those of your imagination.

## CHAPTER 8

# Screen Clearing and Timing

This chapter is about clearing the screen for more effective displays and timing.

In a previous chapter it was explained that it is possible to clear the television screen by typing `CLS`. It is also possible to clear the screen by using the `SHIFT`, `CTRL` and `ESC` keys but this of course clears the memory.

Type and RUN:

```
10 REM : Addition
15 PRINT
20 LET ▲ A = INT (RND (1) *10) +1
30 LET ▲ B = INT (RND (1) *10) +1
40 PRINT "What is ";A;"+";B
50 INPUT "My answer is"; C
60 PRINT "Your answer is ....";C
70 PRINT "Correct answer "; A+B
80 PRINT
90 GOTO ▲ 15
```

## INTRODUCTION

## TO RECAP

## CONSIDER THIS PROGRAM

## **E**XPLANATION

- LINE 15 — empty print line
- LINE 20 — chooses a random number between 1 and 10
- LINE 30 — chooses a random number between 1 and 10
- LINE 40 — asks you to add the two numbers
- LINE 50 — asks for your answer
- LINE 60 — prints your answer
- LINE 70 — prints the correct answer
- LINE 80 — empty print line
- LINE 90 — back to the start line 15

## **T**HE UNTIDY RESULT

Answer the questions a few times and before long you will fill the screen with text which can be rather irritating and is also untidy.

## **A**NOTHER APPROACH

Let's see if we can tidy it up a little. Break into the program (press ESC) and change line 15 to read:

```
15 CLS
```

RUN the program again. This time the screen has been cleared before the program starts, which looks better doesn't it but now look. After you have INPUT your answer, the screen clears with remarkable speed and the next question is printed. This gives you no time to see if your answer was correct. There are ways round this and here is one. Break into the program and type:

```
80 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T
```

# **A SLOWER, MORE READABLE RESULT**



This will over-write the existing line 80. RUN the program again and you will see that you now have sufficient time to read the answer before the screen is cleared.

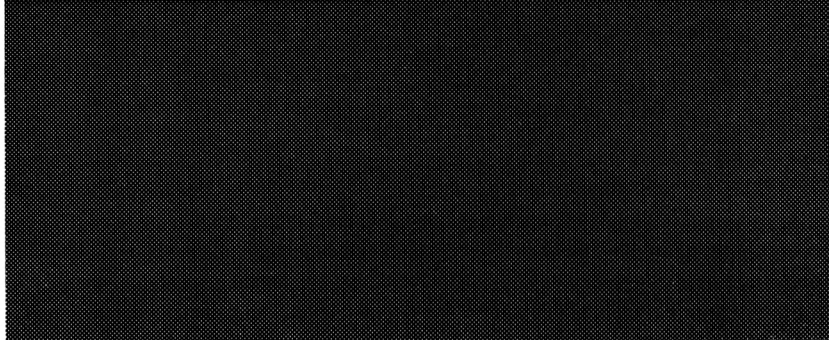
If you like, line 80 acts as a clock. The statement effectively stops the program for a given length of time.

FOR ▲ T = 1 TO 1000 : NEXT ▲ T = 1½ secs  
approximately

FOR ▲ T = 1 TO 5000 : NEXT ▲ T = 6 secs  
approximately

FOR ▲ T = 1 TO 10000: NEXT ▲ T = 12 secs  
approximately

You can see how the combination of CLS and the timing loop can help to improve your programs. You will find it useful to always begin all your programs with the CLS command, e.g. 10 CLS. This gets rid of all program listings etc. ensuring that you have a clear screen where your program starts RUNNING.



## CHAPTER 9

# Looping and Branching

## USING GOTO TO REPEAT

You can use GOTO in a program when you want to repeat something.

Type and RUN:

```
10 REM : This is an example of looping
20 PRINT "This will help to show a use for GOTO"
30 GOTO ▲ 20
```

The computer proceeds to PRINT the contents of line 20 indefinitely because after first doing so, line 30 sends the program back to line 20 each time. The flickering bottom line is proof that the computer is still printing line 20. Break into the program by pressing the ESC key twice.

This process is called LOOPING because the program goes round in a loop.

## GOTO BRANCHING

## GOING ROUND IN A LOOP

GOTO, however, has another use. Consider this program!

```
10 REM : This is an example of Branching
20 PRINT "Hello"
30 INPUT "What's your name"; A$
40 GOTO ▲ 60
50 PRINT "That's a funny name!"
60 PRINT "Hello "; A$; " I'm pleased to meet you"
```

Type and RUN it.

INPUT your name when asked and see what happens.

You might have expected the reply "That's a funny name!" because of the PRINT statement at line 50. We have, however, managed to stop the computer from being rude to you by placing a GOTO statement at line 40 — GOTO 60. This means exactly what it says — GOTO line 60 — do not execute any lines in between! The GOTO command is therefore very easy to understand. It is also one of the most effective commands at your disposal. When used with other commands it also becomes extremely useful.

## OTHER USES OF GOTO AND BRANCHING USING IF-THEN

Let's introduce another two new words — IF and THEN.

Type and RUN:

```
10 REM : Another way to count ...
20 PRINT "I want this program to count from 0 to 10"
30 LET ▲ C = 0
40 PRINT ▲ C
50 LET ▲ C = C+1
60 IF ▲ C=11 ▲ THEN ▲ GOTO▲80
70 GOTO ▲ 40
80 PRINT "The End"
```

This is what has happened:

LINE 20 — Print line

LINE 30 — LET C = 0 simply means let the count start at 0

LINE 40 — Prints C

LINE 50 — LET C=C+1 means let the new count equal the old count plus one. Therefore the count works like this:



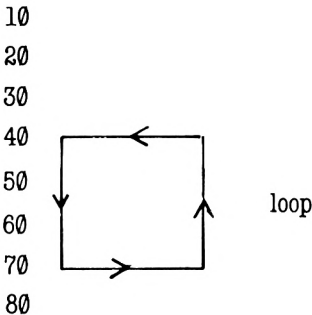
```
C = 0  
PRINT 0  
C = 0 + 1  
PRINT 1  
C = 1 + 1  
PRINT 2 etc
```

As the count continues the program runs through line 60 — IF C = 11 THEN GOTO 80. IF the count doesn't equal 11 THEN this instruction is ignored and the program continues to line 70 and therefore loops back to line 40. When the count does equal 11, the computer takes notice of the instruction in line 60 and goes to line 80 where the print statement is "The End".

IF statements are considered as "conditional statements". That is, the computer will act as an instruction after considering the information received.

## How BRANCHING AND GOTO LOOPING WORK

This diagram may help you to see what is happening with the GOTO, IF-THEN statements.



There is another way to END a program. This is by using the command END.

## THE END COMMAND

Change line 60 to read:

```
60 IF ▲ C = 11 ▲ THEN ▲ END
```

and now delete line 80.

RUN the program again. We will discuss END in more detail shortly but first let's look at a slightly more involved program which uses GOTO and IF-THEN.

Type and RUN:

```
10 REM : French Vocabulary Program
20 PRINT "Do you want to give answers in French
    (Press 1) or in English (Press 2)"
30 INPUT "My choice is ";A
40 IF ▲ A = 1 ▲ THEN ▲ GOTO▲70
50 IF ▲ A = 2 ▲ THEN ▲ GOTO ▲ 110
60 GOTO ▲ 30
70 PRINT "What is the French for...the table"
80 INPUT "My answer is";B$
90 IF ▲B$= "la table" THEN ▲ GOTO ▲ 200
100 GOTO ▲ 70
110 PRINT "What is the English for...la table"
120 INPUT "My answer is"; C$
130 IF ▲ C$ = "the table" THEN ▲ GOTO ▲ 200
140 GOTO ▲ 110
200 PRINT "Well done, correct answer"
```

## OVER TO YOU

This is an unsatisfactory program in a number of ways, not least because it only tests one word! However, its main purpose is for you to see how GOTO and IF-THEN are put to work within a program. There is a French vocabulary test in Chapter 19 which is of real use and worth SAVEing on tape.

Program commentary:

LINE 20 — asks whether you want to answer in English or French.

LINE 30 — asks you to INPUT your choice either 1 or 2

LINE 40 — IF you INPUT 1 THEN the computer branches to line 70 for the French question and ignores lines 50 and 60. IF-THEN conditional statement — IF the condition is fulfilled THEN....

LINE 50 — IF you INPUT 2 THEN the computer branches to line 110 for the English question and ignores lines 60 – 100.

LINE 60 — this is an interesting line. If you type any number other than 1 or 2, line 60 will send the program back to line 30. This will continue until you type 1 or 2.

LINES 70 and 80 — question and answer lines

LINE 90 — another IF-THEN GOTO line. IF your answer is correct THEN GOTO line 200 for some praise. This is a branch — lines 100 to 140 are ignored.

LINE 100 — sends the program back to line 70 to repeat the question should you get it wrong. One important point. Your answer must be exactly the same as the answer in line 90, otherwise the computer will treat it as incorrect.

LINES 110 and 120 — question and answer line.

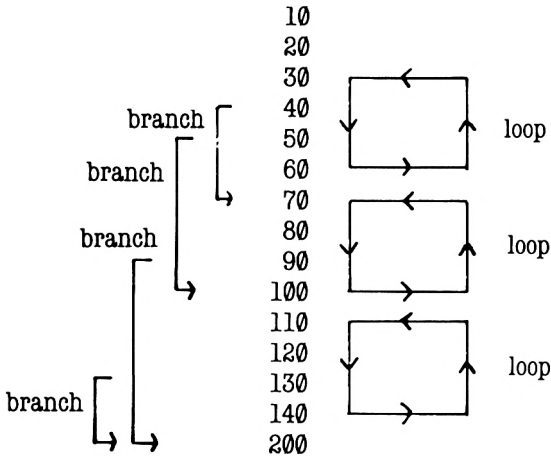
LINE 130 — IF answer is correct THEN GOTO line 200 for praise.

LINE 140 — back to line 110 for another try

LINE 200 — print line.

# How BRANCHES AND GOTO LOOPING WORK

This diagram may help to see the flow of the program.



# THINGS TO NOTICE

Some points to notice. Lines 30, 40 and 50 all mention A, and also the numbers 1 and 2. 'A' is an example of a numeric variable. Simply, 'A' can either be the number 1 or 2, i.e. it can vary — hence numeric variable (see chapter 6 if you are not sure).

## STRING VARIABLES



LINES 80 and 90 both mention B\$. When you see the sign \$ together with a letter, you have come across a string variable (see Chapter 7). In this instance the string is going to be two words — hopefully la table! However, the answer you give may be la plume, or something else — your answer can vary as B\$ is a string variable.

## MORE DETAIL ABOUT END



You will recall that we introduced the command END earlier in this chapter. This command can be used to finish off your program in a more satisfactory way. Type and RUN:

```
10 PRINT "Hello"  
20 PRINT "there!"
```

You will see that once the program has been RUN, the computer tells you that it is Ready for more instructions. Add this further line:

```
30 END
```

and RUN it again.

Just the same result — obviously END is not necessary in as simple a program as this. Now add this line:

```
25 GOTO ▲ 10
```

and RUN it again.

The screen is filled again as the computer continues printing Hello There. The END statement has no effect at all because line 25 — GOTO 10 returned the computer to the first print line.

You can see, therefore, if you want to look back at the counting program it is important to place the END command where it is needed — not necessarily at the end of the program listing.

## **IF-THEN CONDITIONAL STATEMENTS**

After that little diversion, let's get back to conditional statements. Type and RUN:

```
10 PRINT "What is 3+4"  
20 INPUT "My answer is...";A  
30 IF ▲ A=3+4 ▲ THEN ▲ PRINT "Correct"  
40 IF ▲ A <> 3+4 ▲ THEN ▲ PRINT "Incorrect"
```

Answer the question both correctly and incorrectly. After you have given the computer your answer it will apply it firstly to line 30, (because that is the next line in the program) and IF the answer is correct, that is IF  $A = 3+4$ , the instruction in that line will be effected, i.e. it will print the word "correct". Should the answer be wrong i.e. IF  $A \neq 3+4$  the computer considers that line 30 does not apply and therefore passes to line 40 and prints the word "incorrect".

You will know that the sign = means "equal to" but you may not be sure about the sign <> at line 40. <> means "is not equal to" and is just the symbol < together with the > symbol. Therefore whenever IF is used in this way you may well have to use both = and <>, as the computer's next move will be conditional on the information it has received.

## **ELSE — A BETTER WAY**

There is another and probably better way of writing lines 30 and 40. Change line 30 to read:

```
30 IF ▲ A = 3+4 ▲ THEN ▲ PRINT  
    "Correct" ELSE ▲ PRINT "Incorrect"
```

Now delete line 40 and RUN the program again getting the answer both right and wrong.

You can see what has happened. The computer considers the answer — IF it agrees that your answer is correct THEN it prints “correct” — or ELSE it prints the word “incorrect”. In other words, either your answer is correct or ELSE it isn't!

We feel, however, that as you feel your way you may be happier at first using the sign = and <>. Both of these signs can be used with words as well as numbers.

There are other signs which are used in the same way as = and <>. These are:

< which means “less than”

> which means “greater than”

<= which means “less than or equal to”

>= which means “greater than or equal to”.

Here is a program which demonstrates the different functions of these signs. Type and RUN:

```
10 CLS
20 PRINT "Give me a number between 3 and 8"
30 INPUT "Number";A
40 PRINT
50 PRINT ▲ A
60 IF ▲ A < 4 ▲ THEN ▲ PRINT "Low"
70 IF ▲ A < = 5 ▲ THEN ▲ PRINT "Five or under"
80 IF ▲ A > = 6 ▲ THEN ▲ PRINT "Six or over"
90 IF ▲ A > 7 ▲ THEN ▲ PRINT "High"
100 PRINT
110 GOTO ▲ 20
```

## **B**OTH = AND <> CAN ALSO BE USED WITH WORDS

## **F**OUR MORE SIGNS

## **E**XAMPLE

# **E**XPLANATION

If you INPUT the numbers 3 to 8 in sequence you will see how the computer acts on the instructions in the conditional lines. The computer will consider the IF lines in sequence; if the number input is 3 it will act as follows:

LINE 60 — is the number less than ( $<$ ) 4?  
— answer yes — then print  
“low”.

LINE 70 — is the number 5 or less than  
( $\leq$ ) 5? — answer yes —  
then print “five or under”.

LINE 80 — is the number greater than or  
equal to ( $\geq$ ) 6 — answer no  
— no action.

LINE 90 — is the number greater than ( $>$ )  
7 — answer no — then no  
action.

If the number INPUT is 8, then the same principle applies except that the computer acts on lines 80 and 90.

However, if the number INPUT is either 4 or 5, the computer will consider line 60 and ignore it because neither 4 or 5 is less than ( $<$ ) 4. It will though act on line 70 because both numbers fulfil the condition in that line — 4 is less than 5 whilst 5 is equal to 5.

If the number, INPUT are either 6 or 7, the computer will only act on line 80. Why?

That's right!  $> =$  means greater than or equal to.

## **T**WO IMPORTANT WORDS USED WITH IF

Let us now look at two more words which can be used with IF-THEN. These are AND and OR. They are known as Logical Operators.



## THE MEANING OF "AND"

AND means that one statement AND another must be true for the computer to comply with the command.

## AN EXAMPLE OF ITS USE

Type and RUN this short program:

```
10 REM : Logical operators
20 CLS
30 PRINT "Give me the lowest four numbers in
   sequence which are divisible by 5"
40 INPUT "Lowest number" ; A
50 INPUT "Next lowest"; B
60 INPUT "Next lowest"; C
70 INPUT "Next lowest"; D
80 IF ▲ A = 5 ▲ AND ▲ B = 10 ▲ AND ▲ C = 15 ▲
   AND ▲ D = 20 ▲ THEN ▲ PRINT "Correct" ELSE ▲
   PRINT "Incorrect"
```

The conditional line 80 is dependent on the answer at lines 40, 50, 60 and 70 being correct to get the computer to say so. A wrong answer at any of those lines will result in the computer printing "incorrect". Why? That's right — because all the answers must be correct — A = 5 AND B = 10 AND C = 15 AND D = 20.

Now change line 80 to read:

```
80 IF ▲ A = 5 ▲ OR ▲ B = 10 ▲ OR ▲ C =
   15 ▲ OR ▲ D = 20 ▲ THEN ▲ PRINT
   "Good try" ELSE ▲ PRINT "Sorry all
   wrong"
```

and add line 90:

```
90 PRINT "The answer is 5,10,15,20"
```

The condition in the new line 80 means that if ANY of the answers given are correct the computer will print "Good try" followed by the correct answer.

## LOGICAL OPERATORS CAN BE USED WITH WORDS

As with the signs = and <> the operators AND and OR can be used with words as well as numbers. Their use can be very effective where your computer is being used as a teaching aid. Consider the following:

### AN EXAMPLE

```
10 CLS
20 PRINT "What is the capital of France"
30 INPUT "My answer is "; A$
40 IF ▲ A$ = "Paris" THEN ▲ PRINT "Correct"
50 IF ▲ A$ <> "Paris" THEN ▲ PRINT "Incorrect, the
    answer is Paris"
```

Type and RUN it.

For the sake of argument let us suppose that the word Paris is incorrectly spelt, e.g. Parris. The answer is almost correct but the computer only recognises Paris as the correct answer. We could, therefore, change lines 40 and 50 to include as many variations of spelling as we realistically think possible, for example:

```
40 IF ▲ A$ = "Paris" OR ▲ A$ = "Parris" OR ▲ A$ =
    "Pariss" ▲ THEN ▲ PRINT "Correct"
50 IF ▲ A$ <> "Paris" AND ▲ A$ <> "Parris" AND ▲
    A$ <> "Pariss" THEN ▲ PRINT "Incorrect, the
    answer is Paris"
```

Try it again, using the variations.

Just a reminder — we could have typed the IF-THEN lines as:

```
40 IF ▲ A$ = "Paris" OR ▲ A$ = "Parris" OR
    ▲ A$ = "Pariss" THEN ▲ PRINT "Correct"
    ELSE ▲ PRINT "Incorrect, the answer is
    Paris"
```

Leave this program on the screen whilst we consider the final word to be looked at in this chapter — NOT. This, like AND and OR, is also a Logical Operator and again is usually used with the command IF.

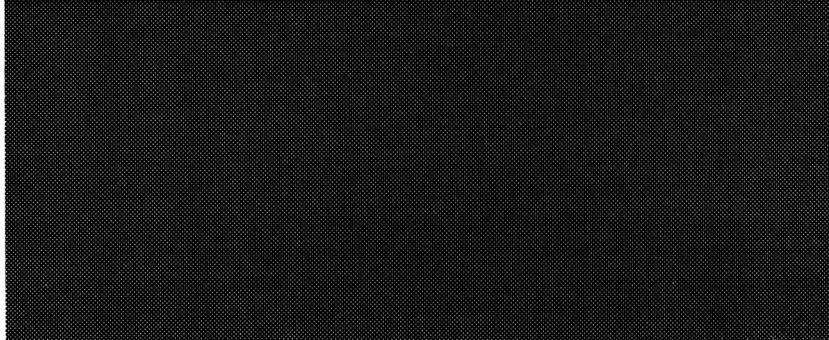
## **A** **AN EXAMPLE** **OF THE USE** **OF "NOT"**



It is unlikely however that you will use this command, as it is easier to get the same result using the sign <> or the command ELSE. However, change line 50 to read:

```
50 IF ▲ NOT ▲ A$ = "Paris" AND ▲ NOT ▲  
    A$ = "Parris" AND ▲ NOT ▲ A$ =  
    "Pariss" THEN ▲ PRINT "Incorrect, the  
    answer is Paris"
```

RUN the program again — making sure you get the answer wrong of course. The result is the same as before.



## CHAPTER 10

# Subroutines

You have seen in the previous chapter that the GOTO instructs the computer to skip lines in a program. There is another command which has a similar function — GOSUB.

GOSUB is short for GO TO SUBROUTINE. It can be used when a number of identical lines are repeated at different stages within a program.

Consider this program BUT DO NOT TYPE IT:

```
10 LET ▲ A = 5
20 LET ▲ B = 6
30 PRINT "What is"; A; "+"; B
40 INPUT "My answer is...." ; C
50 IF ▲ C = A + B ▲ THEN ▲ PRINT "Correct" ELSE
   ▲ PRINT "Incorrect — the answer is "; A + B
60 LET ▲ A = 7
70 LET ▲ B = 4
80 PRINT "What is "; A; "+"; B
90 INPUT "My answer is .... " ; C
100 IF ▲ C = A + B THEN ▲ PRINT "Correct" ELSE ▲
    PRINT "Incorrect — the answer is "; A + B
110 END
```

## INTRODUCTION

## WHEN TO USE GOSUB

## FOR EXAMPLE

## **E**XPLANATION

LINE 10 and 20 — set up the variables.  
LINE 30 — asks the question  
LINE 40 — invites the answer  
LINE 50 — considers the answer and responds  
LINE 60 — 100 — repeat the exercise and you can imagine the length of program necessary to ask 20 questions with answers.

## **A<sub>N</sub>** **IMPROVED** **EXAMPLE**

This is much better; type it in:

```
10 REM : Subroutine
20 LET ▲ A = 5
30 LET ▲ B = 6
40 GOSUB ▲ 1000
50 LET ▲ A = 7
60 LET ▲ B = 4
70 GOSUB ▲ 1000
80 END
1000 PRINT "What is "; A;" + "; B
1010 INPUT "My answer is ...." ; C
1020 IF ▲ C = A + B ▲ THEN ▲ PRINT "Correct" ELSE
    ▲ PRINT "Incorrect — the answer is "; A + B
1030 RETURN
```

## **E**XPLANATION

Let us examine the program as the computer obeys each instruction.

LINE 20 and 30 — set the variables.  
LINE 40 — sends the computer to the subroutine at line 1000.  
LINE 1000 — asks the question.  
LINE 1010 — invites your answer.  
LINE 1020 — considers your answer and responds.

LINE 1030 — RETURN sends the computer back to the main program at the line IMMEDIATELY after the GOSUB line.

LINE 50 and 60 — set up the new variables.

LINE 70 — sends the computer back to the subroutine.

LINES 1000 to 1020 — repeat the exercise.

LINE 1030 — RETURNS the computer to the line after GOSUB in line 70.

LINE 80 — ENDS the program. The command END is important here because if it were to be left out the program would eventually crash.

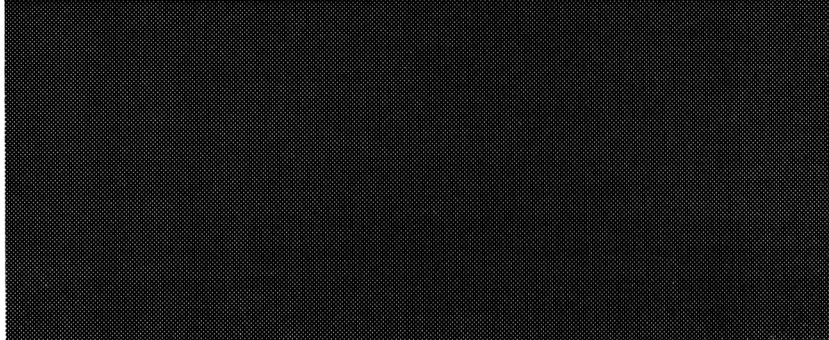
Delete line 80 and RUN the program again to see what happens.

You can see from this program that it would now be quite easy to add further lines similar to 20, 30, 50 and 60, to instruct the computer to ask more and more questions.

Subroutines (GOSUB-RETURN) are a very important part of programming and you will find that you use them more and more as your programs get longer.

## **CONCLUSION**







## CHAPTER 11

# Summary

Before you go any further you should satisfy yourself that you have understood what you have read so far. The following is a useful check list.

(1) Do you know the uses of the following keys: ENTER, SHIFT, CTRL, ESC, COPY, CAPS LOCK, DEL, CLR?

(2) Can you move the cursor around the screen?

(3) Do you know how the computer uses “ ”, ;, ? and when to leave spaces?

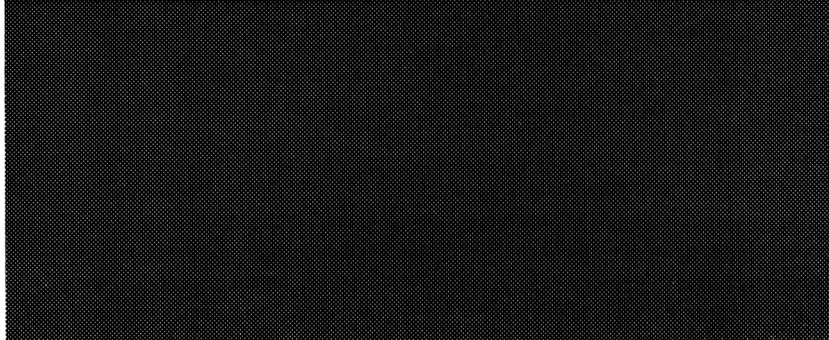
(4) Do you understand the command words:

PRINT, RUN, END, NEW, LET, GOTO, GOSUB, RETURN, CLS, FOR, NEXT, INPUT, INT, RND, TAB, LOCATE, REM, ELSE, IF, THEN, LIST, RENUM?

(5) Can you give examples of numeric and string variables? Can you write simple programs using them?

(6) Can you recognise looping, branching and subroutines? Can you write simple programs involving them?

If you feel unhappy about any of the above you should go back and refresh your memory. If you feel confident, then go straight ahead.



## CHAPTER 12

# Using the datacorder

The built-in Datacorder gives the Amstrad a significant advantage over other home computers which have to use plugged-in tape recorders to load and save programs. Loading from such recorders can often prove frustrating. No such problems for you.

The introductory manual which accompanies your computer is very clear about how to load and save tapes, so hints from us, we feel, will suffice.

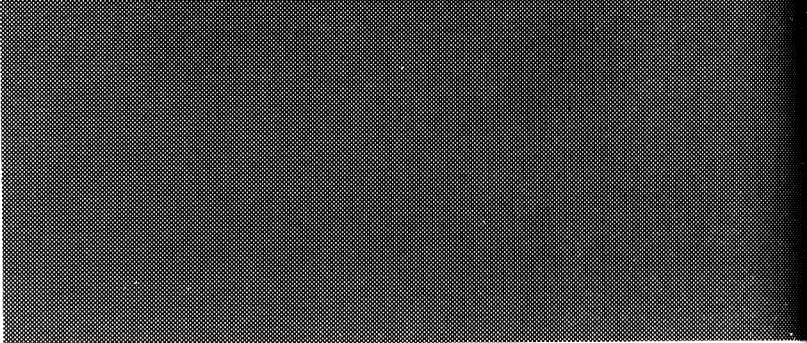
It is advisable when saving your own programs to bear in mind:

- (a) Make sure that your program RUNS before saving it.
- (b) Use the VERIFY command to confirm that you have SAVED the program listing as it appears on the screen.

## AMSTRAD ADVANTAGE

## HOW TO USE PRE- RECORDED PROGRAMS

## SAVING PROGRAMS ON TAPE



**(c)** SAVE each program twice or three times in case anything goes wrong with one of the programs.

**(d)** Don't use long tapes. C15 or less is all you need.

## CHAPTER 13

# The Printer

The CPC 464 certainly has an advantage over many home computers in that it is capable of connecting with a printer of commercial standard.

**C**OMMERCIAL  
ADVANTAGE

You will of course have to consider its value to you, as these "add-ons" are not cheap. If you intend to put your computer to real use, then a printer becomes a must and should not be purchased without professional advice.

**C**ONSIDER  
BEFORE  
BUYING

If you already have a printer then it is obviously possible to produce "hard copy" of all your programs. You can then keep a file of your best programs. Browsing through these can often give you a good idea for other programs or suggest improvements to your existing ones.

Moreover, if you hit problems with a program, a print-out of where you have got to will make it easier for you to discuss with other computer fanatics the problems you have faced, in the hope that a solution can be found. Exchanging programs by swapping printouts is also a good deal cheaper than swapping by cassette.

## **C**OMMANDS FOR THE PRINTER



Several of the programs in this book are test programs. They need very little adaptation to be able to print, say, a score out of 20 in a vocabulary test together with a complete printout of the words and the correct answers. This "hard copy", in its turn, provides a basis for further future revision.

## **P**RINTING LISTINGS

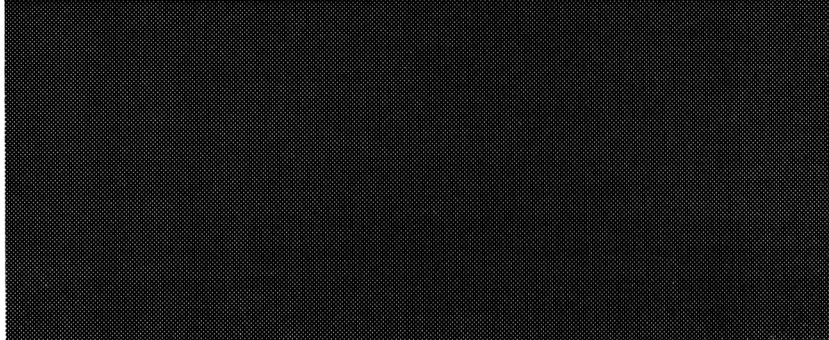


It is also possible to produce worksheets for all educational subjects, quiz programs and of course copies of graphic designs.

This has been just a short insight into the possible uses of a printer. We hope it has been of use in assessing its value.

## **SECTION 3**

# **INTRODUCTION TO COLOUR, SOUND AND GRAPHICS**





## CHAPTER 14

# MODES — and a few tips

We are now going to look briefly at the different types of screen display which are built into the Amstrad. You will remember that we explained in Chapter 3 that the screen is split into 40 vertical columns and 25 horizontal rows.

There are actually three different sizes of screen layouts — they are called MODES and these are:

MODE 1 — which is 40 columns wide by 25 rows  
— this is the size of screen seen when the computer is first switched on.

MODE 0 — which is 20 columns wide by 25 rows.

MODE 2 — which is 80 columns wide by 25 rows.

Here is a short program to demonstrate the three effects.

Type and RUN:

```
10 MODE ▲ 1
20 GOSUB ▲ 1000
30 MODE ▲ 0
40 GOSUB ▲ 1000
```

## THREE DIFFERENT MODES

## AN EXAMPLE USING THE THREE MODES

```

50 MODE ▲ 2
60 GOSUB ▲ 1000
70 END
1000 LOCATE ▲ 12,12
1010 PRINT "Wow!"
1020 FOR ▲ T = 1 ▲ TO ▲ 5000: NEXT ▲ T
1030 RETURN

```

You will see that the exclamation has been printed first in usual size (MODE 1) at position, 12,12 (LOCATE) — then in larger print (MODE 0) — and finally in small print (MODE 2).

## SCREEN CLEARING

LINE 10 — is interesting. It is not really necessary but it has done the job that the command CLS DOES — it has cleared the screen before starting the program.

LINE 30 — then sets the computer into MODE 0, a smaller screen, thereby making the printing bigger. We will look a little closer at MODE 0 in the chapter about colour.

LINE 50 — finally sets the screen as its largest size thereby reducing the size of print. This MODE is very useful when typing in programs because so much more text can be displayed making it easier to follow.

## SOMETHING TO NOTICE

Notice also how we have used the subroutine (lines 1000 to 1030) technique in this program. You will see too that the computer remains in MODE 2. To return to it's switched-on state, simply type:

MODE ▲ 1 and press ENTER

We can actually restore to MODE 1 via our program. Type this extra line:

```
65 MODE ▲ 1
```

Now RUN the program again.

## AUTO

Let's now try to make life a little easier for you and perhaps help to save some typing time. Consider the following program but don't type it in:

```
10 CLS
20 PRINT "Hints and tips"
30 PRINT
40 PRINT "Always use AUTO"
50 PRINT "It saves time!"
```

Now type the word AUTO and press ENTER.

You will see that the number 10 has been printed and the cursor is ready to accept your program line. Now type the first line of the program:

```
CLS
```

Press ENTER and 20 is printed — the cursor is waiting for you again. Now type the rest of the program and after pressing ENTER after line 50 press the ESC key. The computer can now accept the command RUN. Line 60 will not be included in your program — type LIST to confirm.

AUTO is obviously very useful when entering programs and can be pre-set to any line number you require as follows. Type:

```
AUTO ▲ 2,2
```

and press ENTER a few times.

Press ESC

## ANY SEQUENCE

Type:

AUTO ▲ 50, 100

Type:

AUTO ▲ 100, 100

AUTO on it's own always starts at line 10 and goes up in tens.

Here's another little tip.

## SAVE TYPING TIME

Type and ENTER (don't forget AUTO):

```
10 CLS
20 PRINT "Another tip"
30 PRINT "The ? is short for PRINT"
40 PRINT "That's useful!"
```

Line 30 says "The ? is short for PRINT". Type this extra line using SHIFT and ?

50 ? "So it is"

This really can save time when you think that PRINT is probably the most used command of all! We shall however still use the word PRINT in all programs in this book, but if you feel confident about using them go ahead!

## ANOTHER TIME SAVER

Here's another hint — add this line but don't type the last set of speech marks:

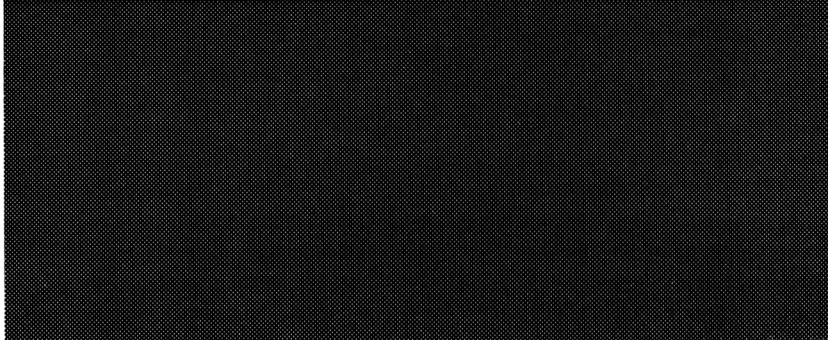
```
60 PRINT "Look no speech marks at the end of  
this sentence.
```

## **B**EWARE!



Just one point about speech marks — only leave them off if it is the end of a program line. Don't leave them out in the middle of a line otherwise it will result in a syntax error. Again, for the purposes of this book all speech marks are included.

As you progress you will find other short cuts which will reduce typing and programming time.



## CHAPTER 15

# Colour

This chapter serves as a brief introduction to colour. The Amstrad has a large range of colours available — some 27 in all. Let's see if we can demonstrate some of them. For those who have the green screen monitor, the changes in colour will appear as varying shades of green — but you will still get the general idea of colour change.

The screen is actually made up of two parts — the border and the screen display area. When the computer is switched on both the border and screen are blue.

Type:

BORDER ▲ 0

Press ENTER.

Now try:

BORDER ▲ 1

Press ENTER.

So you now know that when the computer is switched on, the border colour is colour 1.

The border colour can change through all 27 colours — they are numbered from 0 to 26.

This program will demonstrate all 26 colours:

```
10 REM : Border colours
20 FOR ▲ A = 0 ▲ TO ▲ 26
30 BORDER ▲ A
40 FOR ▲ T = 1 ▲ TO ▲ 1000 : NEXT ▲ T
50 NEXT
```

## INTRODUCTION

## BORDER

You will remember how we use FOR-NEXT. The loop increases the value of A each time, therefore the first value 0 = colour black, second value 1 = colour blue and so on until colour 26.

Now before RUNNING it again type:

MODE ▲ 2

and press ENTER.

You can see that even in MODE 2 the border colour can change to all 27 colours. The same applies to MODE 0.

Refer to the AMSTRAD manual for the full range of border colours. SHIFT, CTRL, ESC.

## **P**APER — THE SCREEN BACKGROUND



Let's now look at the screen display area and see what we can do with that.

Type:

PAPER ▲ 0

Press ENTER and apparently nothing has happened.

Type CLS and press ENTER. Again nothing unusual here.

Type:

PAPER ▲ 1

Press ENTER, type CLS and press ENTER. Well at least that shows that PAPER 0 is blue!

The screen area colours can therefore be changed in a similar way to border colour. There is an important difference, however. The screen colours are determined by the MODE. MODE 1 allows for four colour changes — Blue, Yellow, Light Blue and Red.

MODE 2 allows for only two colour changes — Blue and Yellow.



## MORE COLOURS HERE



MODE 0 — the colour mode — allows for sixteen changes. Remembering the BORDER program.

Type and RUN:

```
10 MODE ▲ 0
20 FOR ▲ A = 0 ▲ TO ▲ 15
30 PAPER ▲ A : CLS
40 FOR ▲ T = 1 ▲ TO ▲ 1000 : NEXT ▲ T
50 NEXT ▲ A
```

Again you can see the computer demonstrating the background colours available. The last two are interesting — colour 14 is flashing between blue and yellow and colour 15 between pink and sky blue.

Again consult your AMSTRAD manual for the full range. SHIFT CTRL, ESC.

## FLASHING COLOURS



## PEN — THE TEXT COLOUR



We have looked at BORDER and PAPER — let's now change the colour of the printing.

Type:

```
PEN ▲ 0
```

and press ENTER.

This time the pen colour is the same as the border colour because the cursor has vanished.

Try:

```
PEN ▲ 1
```

ENTER.

So yellow is pen colour 1.

Again the PEN colours are governed by the MODE.

MODE 1 — has four colours available — blue, yellow, pale blue and red.

MODE 2 — has two — blue and yellow.

MODE 3 — has the same sixteen as the PAPER colours.

## EXPERIMENT YOURSELF

Experiment with these colours to familiarise yourself with them.

Using these features we can begin to make our programs look more effective, for example.

## EXAMPLE

```
10 REM : Simple effects
20 MODE ▲ 0
30 BORDER ▲ 0
40 PAPER ▲ 0 : CLS
50 PEN▲14
60 LOCATE▲9,9
70 PRINT "We"
80 LOCATE ▲ 9,10
90 PRINT "Won"
100 LOCATE ▲ 7,11
110 PRINT "The Cup"
120 GOTO ▲ 120
```

Program explanation:

LINE 20 — sets the MODE.

LINE 30 — BORDER colour

LINE 40 — screen colour: CLS clears the screen to the set colour.

LINE 50 — PEN colour — which is the flashing yellow/blue.

LINES 60-110 — positions the cursor.

LINE 120 — this may be new to you. It has the effect of stopping the program — but in reality it serves to keep the Ready message and cursor off the display.

Now change line 40 to read:

```
40 PAPER ▲ 3
```

and RUN it again.

This time you will see that only the paper behind the printing is red.

Finally let's look at the command INK.

This command can also be used to change the colour of the PAPER and the PEN.

**INK**



**EXAMPLE OF  
THE USE OF  
INK**



IMPORTANT — Press SHIFT CTRL and ESC before typing:

```
10 REM: Colour change with INK.
20 CLS:LOCATE ▲ 7, 13
30 PRINT "Paper Blue — Pen bright yellow"
40 GOSUB ▲ 1000
50 INK ▲ 0, 24 : INK ▲ 1,1
60 PRINT "Paper bright yellow — Pen blue"
70 GOSUB ▲ 1000
80 INK ▲ 0, 6 : INK ▲ 1,18
90 PRINT "Paper bright red — Pen bright green"
100 GOSUB ▲ 1000
110 INK ▲ 0, 26 : INK ▲ 1,0
120 PRINT "Paper bright white — Pen black"
130 GOSUB ▲ 1000
140 INK ▲ 0, 1 : INK ▲ 1,24
150 END
1000 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T
1010 CLS
1020 LOCATE ▲ 7,13
1030 RETURN
```

Now run the program. You can see what the INK commands in lines 50, 80 and 110 have done. This is how INK works.

## **E**XPLANATION

---

Both the screen (paper) colour and text (pen) colour have their own number which the computer recognises when used with the command INK.

INK 0 = screen or paper colour

INK 1 = text or pen colour

## **C**OMPUTER LANGUAGE

---

Therefore to change either screen or text we must tell the computer — in language it understands — what we want to do. For example, to change a screen colour we must tell the computer that it is the screen — number 0 — and also tell it the colour number, e.g. 1 is blue, 6 is bright red etc.

## **C**HANGE PAPER COLOUR

---

Therefore INK 0, 6 tells the computer to change the screen colour to bright red.

Similarly with the text or pen colour. The text or pen's number is 1.

## **C**HANGE PEN COLOUR

---

Therefore INK 1,1 tells the computer that the text colour should be changed to bright green:

80 INK▲0, 6 : INK▲1,18

means that the screen is bright red and the text bright green.

## SWITCH-ON STATE



You will see that at the end of the program the screen returns to its “switch-on” state. Do you know why — line 140 may give you a clue!

We can of course change the paper colour without changing the pen colour, or vice versa. Type this additional line and RUN the program again:

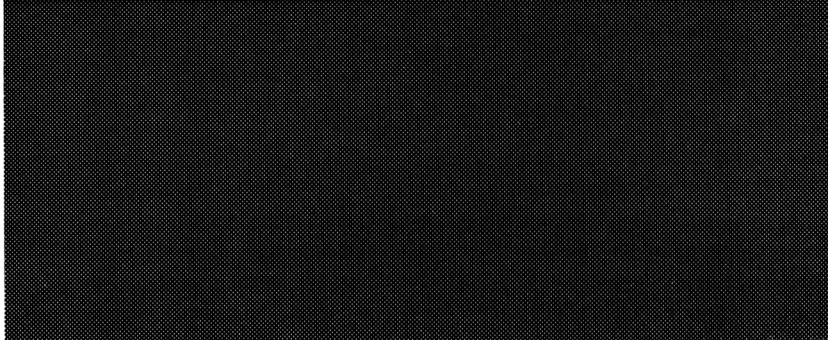
```
145 INK ▲ 1,6
```

You can see that the ready message and cursor have changed to red. This shows that the computer stays in its last instructed colours. To get back to the “switch-on” state — if you want to — simply type INK, 1,24 and press ENTER.

Experiment until you are sure that you know how to change colours using all commands covered in this chapter — BORDER, PAPER, PEN and INK. Look at the colour charts in your User Instruction Manual.

## EXPERIMENT





## CHAPTER 16

# Graphics

## AMSTRAD CHARACTER SET

The AMSTRAD, like most home computers, has a number of character symbols pre-set into its memory. Some of these can be obtained direct from the keyboard via the CTRL key. Hold this key down and press some of the letter keys.

We can also use these symbols within a program by using the command CHR\$ followed by the number of the symbol required placed in brackets.

Type and RUN:

```
10 CLS
20 LOCATE ▲ 20, 12
30 PRINT ▲ CHR$ (225)
```

Change line 30 to:

```
30 PRINT ▲ CHR$ (248)
```

RUN it again.

## ASCII

The Amstrad in common with all home computers has a character set. This set is known as the ASCII code (American Standard Code for Information Interchange). Some of the codes are commands. Change line 30 to read:

```
30 PRINT ▲ CHR$ (7)
```

and try:

```
30 PRINT ▲ CHR$ (12)
```

The majority of code numbers — from 32 to 255 — are all characters. Some are the letters of the alphabet, some numbers and others are the pre-set symbols. Let's have a look at all these symbols. Remember how we use FOR/NEXT?

## DISPLAYING THE CHARACTER SET

Type and RUN:

```
10 FOR ▲ A = 32 ▲ TO ▲ 255
20 PRINT ▲ CHR$(A)
30 FOR ▲ T = 1 ▲ TO ▲ 500 : NEXT ▲ T
40 NEXT ▲ A
```

## SAMPLE ANIMATION

We can do quite a few things with these symbols.

Type and RUN:

```
10 CLS
20 FOR ▲ A = 1 ▲ TO ▲ 40
30 LOCATE ▲ 1,12
40 PRINT ▲ TAB(A); CHR$(251)
50 NEXT ▲ A
```

Program explanation:

LINE 20 — sets the loop to the number of columns.

LINE 30 — positions the cursor each time on the middle row.

LINE 40 — prints the character 251 at the next TAB setting.

## IT'S UP TO YOU

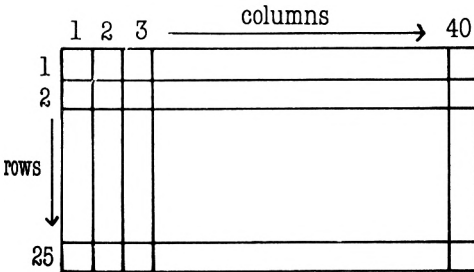
If you have a vivid imagination there is no end to the shapes and movement available to you from the character set.



You will, of course, want to draw your own shapes and designs and the next few pages will give you a basis from which to work.

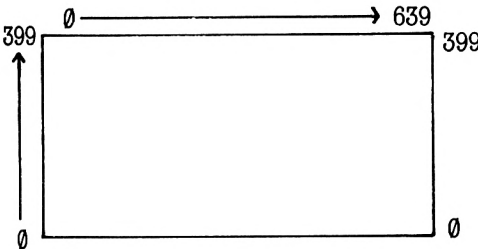
## SCREEN FORMAT

We need to look at the way the screen is laid out for the graphics cursor. You will remember that in Chapter 4 we explained that the screen is laid out as 25 rows of 40 columns:



## WHAT IS A PIXEL?

The layout is really rows of squares in which the letter or character is printed using the PRINT command. Each of these squares is actually broken down into a series of segments known as pixels. The graphics screen consists of 400 rows of pixels x 640 columns and looks like this:



The graphics screen has its own cursor which, unlike the character cursor, is invisible. This cursor allows us to PLOT an exact position on the screen and DRAW lines anywhere we wish.

## GRAPHICS CURSOR

## USING PLOT AND DRAW

To do this we use the PLOT command to position the cursor and then the DRAW command to draw the lines. To show exactly what we mean type this program (first press CTRL and ESC):

```
10 CLS  
20 PLOT ▲ 0, 0
```

Now type RUN and you will see a small dot at the bottom left hand side of the screen. This is the pixel at position 0, 0.

Now add:

```
30 DRAW ▲ 0, 399  
70 GOTO ▲ 70
```

and RUN it again.

## SCREEN REFINEMENT USING GOTO

You can now see that a line has been drawn from the bottom left position 0, 0 to the top left 0, 399. Line 70 merely prevents the character cursor and the Ready message appearing on the screen. Now add:

```
40 DRAW ▲ 639, 399
```

RUN it again

Add:

```
50 DRAW ▲ 639, 0
```

RUN again, and finally:

```
60 DRAW ▲ 0, 0
```

Just to prove that this is drawn at the very outside of the graphics screen add:

```
15 BORDER ▲ 3
```

Now change line 70 to read:

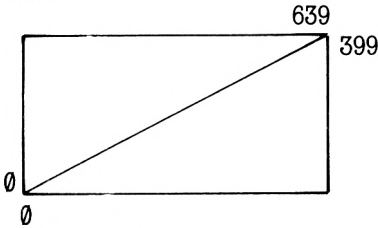
```
70 DRAW ▲ 639, 399
```

and add:

```
80 GOTO ▲ 80
```

# DIAGONAL LINES

This time a diagonal line is drawn to position 639 on row 399.



Now change line 80 to:

```
80 PLOT ▲ 0, 399
```

and add:

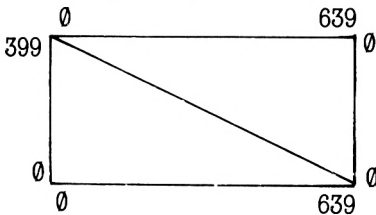
```
90 DRAW ▲ 639, 0
```

```
100 GOTO ▲ 100
```

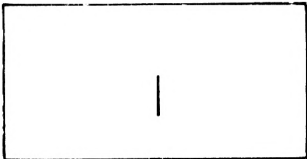
This time a diagonal is drawn from the opposite corners.

LINE 80 — positions the graphics cursor at position 0, 399

LINE 90 — draws the line to position 639, 0



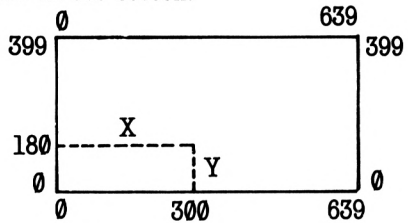
So to DRAW lines between given points we must first PLOT the starting position and then DRAW the line to the finishing position. This is how we do it.



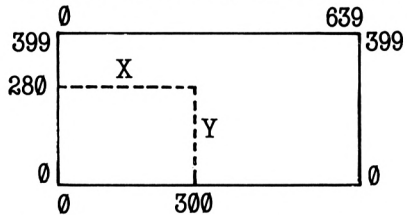
To DRAW a vertical line as shown in the diagram we must first PLOT the start position and the

## CO-ORDINATES X AND Y

finish position. These positions are known as co-ordinates — X and Y. The X co-ordinate PLOTS the position for the cursor across the screen whilst the Y co-ordinate determines the position on the screen.



The X co-ordinate is quoted first in the PLOT command, therefore start position = PLOT 399,



Again the X co-ordinate is quoted first, finish position = DRAW 300, 280

Our program looks like this:

```
10 CLS
20 PLOT ▲ 300, 180
30 DRAW ▲ 300, 280
```

There are two commands which will help you locate the graphics cursor at any time.

## USE FOR X POS AND Y POS

Type:

PRINT ▲ X POS

and ENTER

Now type:

PRINT ▲ Y POS

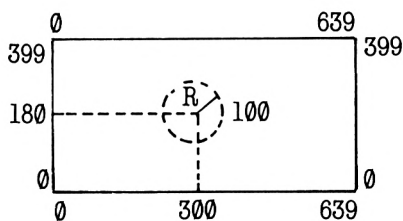
and ENTER

Experiment with the PLOT and DRAW commands until you are confident that you can draw straight lines anywhere on the screen.

## DON'T FORGET TO EXPERIMENT

Let's now move on to drawing circles. We can draw a circle using PLOT after calculating the position of each pixel in the circle. It is not quite as difficult as it sounds. First of all we need to PLOT the centre of the circle — let's use our line program and PLOT the cursor at position 300, 180. We then need to PLOT a point on the circumference — the radius. Let's say our radius is to be 100 (pixels).

## ROUND IN CIRCLES



We must then PLOT each pixel position from the centre of the circle.

## RADIUS AND CIRCUMFERENCE

You will probably know that there are 360 degrees in a circle, so by using the commands DEG, COS and SIN....

Type and RUN:

## MATHEMATICAL SIGNS DEG, COS, SIN

```

10 CLS
20 FOR ▲ A = 1 ▲ TO ▲ 360
30 DEG
40 PLOT ▲ 300, 180
50 PLOT ▲ 300 + 100 * COS (A), 180 + 100
  * SIN (A)
80 NEXT ▲ A

```

We can also DRAW the circle in a different way. Delete line 30 and RUN again.

## ANOTHER WAY TO DRAW CIRCLES

By changing line 50 we can increase or decrease the size of the circle. Change line 50 to:

```

50 PLOT ▲ 300 + 25 * COS (A), 180 + 25 * SIN (A)
RUN

```

Add these additional lines:

```

60 PLOT ▲ 300, 280
70 PLOT ▲ 300 + 25 * COS (A), 280 + 25 * SIN (A)

```

Obviously you can draw as many circles as you like!

## COLOURING CIRCLES

Let's fill them in now. We need to change the PLOT command in lines 50 and 70 to DRAW. Do you know why? — because DRAW tells the computer to DRAW a straight line between two points. Change the lines and watch what happens:

```

50 DRAW ▲ 300 + 25 * COS (A), 180 + 25 * SIN (A)
70 DRAW ▲ 300 + 25 * COS (A), 280 + 25 * SIN (A)

```

O.K. Now replace line 30:

```

30 DEG

```

and see what happens this time.

## CREATE YOUR OWN DESIGN

What about drawing one inside another? Change your program to read:

```
10 CLS
20 FOR ▲ A = 1 ▲ TO ▲ 360
30 DEG
40 PLOT ▲ 300, 180
50 PLOT ▲ 300 + 100 * COS (A), 180 + 100 * SIN (A)
60 DRAW ▲ 300 + 25 * COS (A), 180 + 25 * SIN (A)
80 NEXT
```

It's not just circles. Change line 60 to

```
60 DRAW ▲ 300 + 25 * COS (A), 280 + 25 *
SIN (A)
```

and add

```
70 DRAW ▲ 300, 350
```

LINE 70 DRAWS the line from the centres of the circle to position 300, 350 and line 60 fills in all the pixels.

## THE ORIGIN COMMAND

There is another way to draw circles with the AMSTRAD. The command ORIGIN positions the centre of the circle. You must, of course, PLOT its centre as before.

Type and RUN:

```
10 CLS
20 FOR ▲ A = 1 ▲ TO ▲ 360
30 DEG
40 ORIGIN ▲ 300, 180
50 PLOT ▲ 100 * COS (A), 100 * SIN (A)
60 NEXT ▲ A
```

Now change line 50 to:

```
50 DRAW ▲ 100 * COS (A), 100 * SIN (A)
```

You may feel that the ORIGIN command is easier to use — it's up to you.

## A DASH OF COLOUR

Finally in this section let's add a touch of colour. Remember the program which DRAWS two small circles?

Type and RUN:

```
10 CLS
20 FOR ▲ A = 1 ▲ TO ▲ 360
30 DEG
40 ORIGIN ▲ 300, 180
50 DRAW ▲ 25 * COS (A), 25 * SIN (A)
60 ORIGIN ▲ 300, 280
70 DRAW ▲ 25 * COS (A), 25 * SIN (A)
80 NEXT ▲ A
```

Now add these additional lines:

```
45 INK ▲ 1,3
65 INK ▲ 2,9
```

and change lines 50 and 70 to read:

```
50 DRAW ▲ 25 * COS (A), 25 * SIN (A), 1
70 DRAW ▲ 25 * COS (A), 25 * SIN (A), 2
```

RUN it again — easy isn't it. I'm sure you can see what we've done. First of all we have specified the INK colour in line 45 and called it INK 1. We have then put the INK 1 in the DRAW line 50. Then we have specified the INK colour for the second circle in line 65 and put INK 2 in the DRAW line 70.

## IMAGINATION

Again — experimentation is the only way to learn with these commands. Your designs are limited only by your own imagination.



## CHAPTER 17

# Sound

The Amstrad has the facility to produce high quality sound both musically and in the form of arcade style sound effects. This chapter is aimed at getting you started on producing sounds and music so that you can begin to understand a quite difficult concept.

Basic sounds and musical notes are however very easy to produce. Sounds are produced by the command SOUND. Each note has a number (a full list of musical notes appears at the end of this chapter). The first number that appears after the command SOUND, however, is the channel number and must always be included.

There are three channels which make it possible to play up to three different SOUNDS together. We are only using channel 1 in this chapter. Turn the volume control at the side of the computer to it's maximum setting. Let's see what we can do.

Type and RUN:

```
10 SOUND ▲ 1, 478
```

The sound produced by this line is Middle C so, using the musical note list, it should be fairly easy to produce a chromatic scale.

Type and RUN:

**H**IGH  
QUALITY FROM  
THE AMSTRAD

**B**ASIC MUSIC

**P**LAY A NOTE

# MUSICAL SCALE

```
10 SOUND ▲ 1, 478
20 SOUND ▲ 1, 426
30 SOUND ▲ 1, 379
40 SOUND ▲ 1, 358
50 SOUND ▲ 1, 319
60 SOUND ▲ 1, 284
70 SOUND ▲ 1, 253
80 SOUND ▲ 1, 239
```

We can also tell the computer how long each note should last — we have to add a number to our type program lines. Change the following lines to read:

```
10 SOUND ▲ 1, 478, 200
50 SOUND ▲ 1, 319, 200
80 SOUND ▲ 1, 239, 200
```

This third number can range from 0 to 32767 and each 100 = 1 second's duration.

Change line 80 to:

```
80 SOUND ▲ 1, 239, 500
```

We have already used three numbers separated by commas after the command SOUND. Each of these numbers represents a different part of the SOUND — the first number is Channel 1, the second is the note or noise played and the third is the duration or length of time that the note lasts. Altogether there are seven different parts to every sound. The first two MUST be typed in — the other five are optional. Here is another option. Alter the scale program to read:

```
10 SOUND ▲ 1, 478, 200, 6
20 SOUND ▲ 1, 426, 50, 2
30 SOUND ▲ 1, 379, 50, 2
40 SOUND ▲ 1, 358, 50, 2
50 SOUND ▲ 1, 319, 400, 6
60 SOUND ▲ 1, 284, 50, 2
70 SOUND ▲ 1, 253, 50, 2
80 SOUND ▲ 1, 239, 500, 6
```

I think you know what the fourth part of the SOUND is! That's right — it's a volume control. Let's see how it can be useful to control the volume of notes.

## SING ALONG

Type and RUN:

10	REM	:	Sur le pont d'Avignon
20	SOUND	▲	1, 478, 50, 4
30	SOUND	▲	1, 478, 50, 4
40	SOUND	▲	1, 478, 50, 4
50	SOUND	▲	1, 426, 50, 4
60	SOUND	▲	1, 426, 50, 4
70	SOUND	▲	1, 426, 50, 4
80	SOUND	▲	1, 379, 50, 4
90	SOUND	▲	1, 358, 50, 4
100	SOUND	▲	1, 319, 50, 4
110	SOUND	▲	1, 478, 50, 4
120	SOUND	▲	1, 506, 50, 4
130	SOUND	▲	1, 478, 50, 4
140	SOUND	▲	1, 426, 50, 4
150	SOUND	▲	1, 638, 50, 4

This plays the first four bars of the old French tune "Sur le pont d'Avignon" but it may be difficult to recognise because the first notes run into each other. Add these lines.

## SOUNDS BETTER

25	SOUND	▲	1, 0, 5
35	SOUND	▲	1, 0, 5
45	SOUND	▲	1, 0, 5
55	SOUND	▲	1, 0, 5
65	SOUND	▲	1, 0, 5
75	SOUND	▲	1, 0, 5

That's better — SOUND 0 is silent and the 5 after SOUND 1, 0 leaves just the right delay between the three C notes and three D notes. Almost right — change lines 40 and 70 to read:

40	SOUND	▲	1, 478, 100, 4
70	SOUND	▲	1, 426, 100, 4

That sounds about right to me at least! You may like to experiment further until it sounds good to you.

## **R**ULES FOR THE "SOUND" COMMAND



You can see therefore that it is fairly easy to produce simple tunes using the first four parts of the SOUND command, that is:

- 1 — CHANNEL
- 2 — NOTE
- 3 — DURATION
- 4 — VOLUME

## **S**OUND ENVELOPES



After a while you will come to appreciate that these notes sound rather bland or toneless and to get the full range of sounds from the Amstrad you will have to look to the next two parts of the SOUND command:

- 5 — VOLUME ENVELOPE
- 6 — TONE ENVELOPE

These parts need quite a bit of understanding—simplified they mean that you can specify the volume of one note and make it go up or down or stay constant. You can also specify the tone of the note.

The volume ENVELOPE is governed by the command ENV and must be followed by a series of 4 numbers.

Type and RUN:

10 ENV ▲ 1, 10, 1, 50

20 SOUND ▲ 1, 478, 500, 4, 1

The volume ENV tells the computer that the volume of the note middle C should rise. The ENV command's numbers correspond as follows.

## FOUR RULES FOR "ENV"



First number — number given to the ENvelope so that it can be specified in the SOUND command. The ENvelope number is shown as the fifth number in the SOUND command, therefore if we give the ENV the number 1 (ENV 1), then the SOUND command reacts — SOUND 1, 478, 100, 4, 1.

Second number — the number of steps in each second as specified in the fourth number which is called STEPTIME — range 0 to 127.

Third number — varies the sound level from a number starting at 0 up to 15, or in a negative form.

Fourth number — the length of time between the steps in the second number. This time is counted in 100ths of seconds (between 0 — 255 + Max 2.55 secs).

You will see how the STEP TIMES are varied by changing line 10 to read:

```
10 ENV ▲ 1, 10, 1, 100
```

Type and RUN. You will notice that the steps last longer. You may have been a little puzzled by the words "negative form" in the third number. Well try:

```
10 ENV ▲ 1, 10, — 2, 50
```

```
20 SOUND ▲ 1, 478, 500, 4, 1
```

Experiment with the command ENV, changing the last three numbers to see what different sounds you can get. Remember the first number in the ENV command must always appear in the SOUND command at the fifth position.

## THE ENT COMMAND



The ENT command governs the tone of the envelope and also has four parts — the first must correspond with ENV and appears at the sixth position in the SOUND command.

Type and RUN:

10 ENV ▲ 1, 10, 1, 50

20 ENT ▲ 1, 10, 5, 50

30 SOUND ▲ 1, 478, 500, 4, 1, 1

The ENT command sequence of numbers is:  
First number — number given to the tone envelope (ENT) so that it can be specified in the SOUND command. (Position six.)

Second number — number of steps in each second as specified in the fourth number. The range of numbers is 0 to 230.

Third number — varies the tone of each step, that is the tone is either higher or lower. Range of tone period is -128 to +127.

Fourth number — the length of time between the steps in the second number. Again the time is counted in 100ths of a second between 0-255.

## SOUND EFFECTS

The final part of the SOUND command is the NOISE. This appears as the seventh number in the SOUND command.

Type and RUN:

10 ENV ▲ 1, 100, 1, 10

20 ENV ▲ 1, 100, -2, 10

30 SOUND ▲ 1, 200, 500, 4, 1, 1, 10

Ride em cowboy!!!

Again experiment with all these commands.

## MUSICAL NOTES

### OCTAVE — 3

Note	Number
C	3822
C#	3608
D	3405
D#	3214
E	3034
F	2863
F#	2703
G	2551

### OCTAVE — 2

Note	Number
C	1911
C#	1804
D	1703
D#	1607
E	1517
F	1432
F#	1351
G	1276

G# 2408  
 A 2273  
 A# 2145  
 B 2025

**OCTAVE — 1**

C 956  
 C# 902  
 D 851  
 D# 804  
 E 758  
 F 716  
 F# 676  
 G 638  
 G# 602  
 A 568  
 A# 536  
 B 506

**OCTAVE — 1**

Note	Number
C	239
C#	225
D	213
D#	201
E	190
F	179
F#	169
G	159
G#	150
A	142
A#	134
B	127

**OCTAVE — 3**

C 60  
 C# 56  
 D 53  
 D# 50  
 E 47  
 F 45  
 F# 42  
 G 40  
 G# 38  
 A 36  
 A# 34  
 B 32

G# 1204  
 A 1136  
 A# 1073  
 B 1012

**OCTAVE — 0**

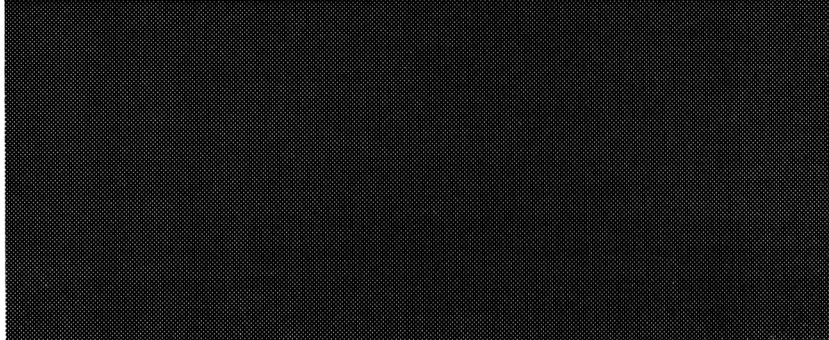
C 478  
 C# 451  
 D 426  
 D# 402  
 E 379  
 F 358  
 F# 338  
 G 319  
 G# 301  
 A 284  
 A# 268  
 B 253

**OCTAVE — 2**

Note	Number
C	119
C#	113
D	106
D#	100
E	95
F	89
F#	84
G	80
G#	75
A	71
A#	67
B	63

**OCTAVE — 4**

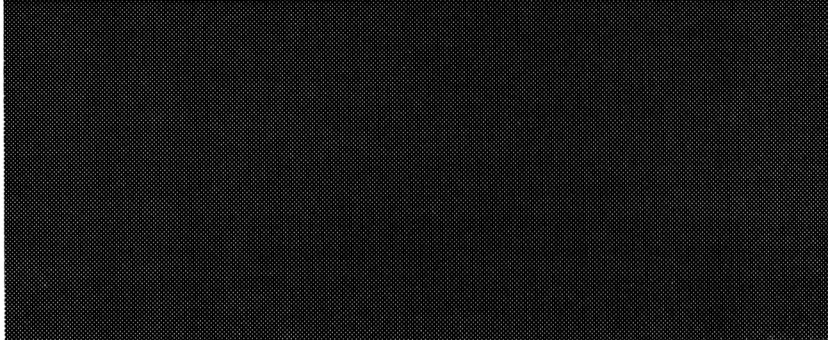
C 30  
 C# 28  
 D 27  
 D# 25  
 E 24  
 F 22  
 F# 21  
 G 20  
 G# 19  
 A 18  
 A# 17  
 B 16





## **SECTION 4**

# **MORE BASIC TECHNIQUES**



## CHAPTER 18

# How to handle DATA

## AN EXAMPLE OF READ AND DATA EXPLAINED

This chapter is meant as an introduction only. It concentrates on showing examples of how you can begin to use READ, DATA and DIM.

The first program shows you how to use DATA. Look at the program explanation carefully.

Type and RUN:

```
10 REM : Averages using READ and DATA
20 CLS
30 LET ▲ T = 0
40 FOR ▲ K = 1 ▲ TO ▲ 5
50 READ ▲ N
60 PRINT ▲ N
70 LET ▲ T = T + N
80 NEXT ▲ K
90 LET ▲ A = T/5
100 PRINT "The average is "; A
110 DATA ▲ 6,4,7,2,1
```

In this program the numeric variable T stands for total and is set in line 30 initially to 0.

LINE 40 — sets the FOR-NEXT loop. This loop executed 5 times, during which line 50 READS N, the first item of the DATA Line, i.e. 6. The command READ tells the computer to search for the DATA line.

LINE 60 — prints N

LINE 70 — then establishes a new total T(6)  
—  $T(6) = T(0) + N(6)$

LINE 80 — increments the loop counter.

The second time round the loop, line 50 READS N which is the next item in the DATA line — 4.

LINE 40 establishes the new total  $T(10)$  —  
 $T(6) = T(6) + N(4)$

The program runs through the loop a third, fourth and fifth time. After the fifth execution the program then runs through line 90 which sets up a new variable A (average).

Average = Total  $\div$  5. After five loops the total is now 20, therefore  $20 \div 5$  will be the average.

Line 100 prints the average.

## HOW YOU GO ON

To adapt this program to find the average of different totals is quite easy — simply change the value of line 40 to 1 TO 10 1 TO 50 or whatever and increase the number of items accordingly in the DATA statement. Finally change line 90 to T/10 or T/50 etc.

## ANOTHER EXAMPLE EXPLAINED

READ and DATA statements, are very commonly used in computer programs. Programs can be written which use string variables instead of numeric variables. The next program uses both string and numeric variables together.

Type and RUN:

```
10 REM : PASS/FAIL TEST
20 CLS
30 FOR ▲ K = 1 ▲ TO ▲ 5
40 READ ▲ N$, M
50 PRINT ▲ N$, M
60 IF ▲ M >= 50 ▲ THEN ▲ PRINT "Pass" ELSE ▲
   PRINT "Fail"
70 PRINT
80 NEXT ▲ K
100 DATA "Fred", 25, "Joan", 44, "Alan", 63, "Rita", 89,
   "Brian", 67
```

This is what the screen displays:

```
FRED      25
FAIL
JOAN      44
FAIL
ALAN      63
PASS
RITA      89
PASS
BRIAN     67
PASS
```

This time there are 10 items of DATA, 5 names and 5 numbers representing the marks (M) they obtained.

LINE 40 — READS the name then the marks

LINE 50 — prints the names and mark

LINE 60 — compares the mark obtained with the pass mark — 50. If the mark is  $> 50$  (greater than 50) then PASS is printed otherwise FAIL is printed.

LINE 70 — empty print line for display purposes.

Again the FOR/NEXT loop (lines 30 — 80) runs through five times.

Can you adapt the program to print a PASS/FAIL list for twelve pupils, where the pass mark is 80 and the marks are 32, 94, 26, 83, 70, 55, 99, 37, 85, 10, 91, 70?

**A**ND IT'S  
OVER TO YOU



**A**N EXAMPLE  
OF ARRAYS



READ and DATA can also be used with the command word DIM. Type and RUN the following:

```

10 REM : BUBBLESORT
20 CLS
30 DIM ▲ A (5)
40 FOR ▲ K = 1 ▲ TO ▲ 5
50 READ ▲ A(K)
60 NEXT ▲ K
70 FOR ▲ C = 1 ▲ TO ▲ 4
80 FOR ▲ J = 1 ▲ TO ▲ 4
90 IF ▲ A(J) < A(J+1) THEN ▲ GOTO ▲ 130
100 LET ▲ D = A(J)
110 LET ▲ A(J) = A(J+1)
120 LET ▲ A (J+1) = D
130 NEXT ▲ J
140 NEXT ▲ C
150 FOR K = 1 TO 5
160 PRINT K ▲ A(K)
170 NEXT ▲ K
180 DATA ▲ 6,3,7,2,1

```

When we use the command DIM (Dimension) it is a way of reserving space in the computer which can be used later for storing information. We talk about DIMensioning an array. Line 30 DIM A (5) tells the computer to reserve five spaces for a numeric variable, A.

The numeric variable A is actually recognised as A(1), A(2), A(3), A(4) and A(5) and is represented by the DATA in line 180 — 6,3,7,2,1.

The program works as follows:

LINE 70 — sets up the number of passes through the DATA line.

LINE 80 — controls the number of comparisons in each pass, that is 4, e.g. 6 with 3, 6 with 7 etc.

The computer READs the DATA on line 180 and compares the first two numbers.

LINE 90 — If the first number compared is less than the second, then the program goes to the next comparison and compares the second number with the third.

If the statement in line 90 isn't true, then lines 100, 110 and 120 change the first two numbers round, then the second and third etc.

Lines 150, 160 and 170 — finally print the list of numbers in ascending order, e.g. 1, 2, 3, 6, 7.

This program is called a “bubblesort”.

## A DIFFERENT USE OF DATA

The next program shows another use for DIM plus READ and DATA:

```
10 REM : Printing chosen sentences
20 CLS
30 DIM ▲ A$ (5)
40 FOR ▲ J = 1 ▲ TO ▲ 5
50 READ ▲ A$ (J)
60 NEXT ▲ J
70 PRINT "Here is a list of sentences. Type in the
   number of the sentence you require."
80 PRINT
90 FOR ▲ Q = 1 ▲ TO ▲ 5
100 PRINT ▲ Q, A$ (Q)
110 NEXT ▲ Q
120 PRINT
130 INPUT "The sentence I require is";B
140 CLS
150 PRINT ▲ A$ (B)
160 DATA "The weather is sunny", "It is foggy", "The
   weather is raining", "It is snowing", "The weather
   is windy".
```

This is what the screen displays

Here is a list of sentences. Type in the number of the sentence you require:

- 1 The weather is sunny
- 2 It is foggy
- 3 The weather is raining
- 4 It is snowing
- 5 The weather is windy

The sentence I require is?

In this example the DIMENSIONAL array in line 30 is set to reserve five spaces — 30 DIM A\$(5) — to store our five sentences. You can of course, set the array to 10, e.g. DIM A\$(10) for 10 sentences or 20, e.g. DIM A\$(20) for 200 sentences.

LINES 40, 50, 60 — READ the DATA in line 160  
LINE 70 — prints the instruction  
LINE 90, 100, 110 — print the sentence in the  
DATA line.

LINE 130 — asks your choice.

LINE 150 — prints your chosen sentence.

On its own this program may not seem very useful. However, where you have to choose a selection of comments from a large number of comments, or where you might be writing many letters with slight variations, you can see how useful an adaptation of such a program might be.

## **S**ORTING LETTERS

Finally a simple alphabetical bubblesort.  
Compare this with the numeric bubblesort:

```
10 REM : Simple alphabetic bubblesort
20 DIM ▲ A$(26)
30 FOR ▲ K = 1 ▲ TO ▲ 26
40 READ ▲ A$(K)
50 NEXT ▲ K
60 FOR ▲ B = 1 ▲ TO ▲ 25
70 FOR ▲ J = 1 ▲ TO ▲ 25
80 IF ▲ A$(J) < A$(J+1) THEN ▲ GOTO ▲ 120
90 LET ▲ S$ = A$(J)
100 LET ▲ A$(J) = A$(J+1)
110 LET ▲ A$(J+1) = S$
120 NEXT ▲ J
130 NEXT ▲ B
140 FOR ▲ K = 1 ▲ TO ▲ 26
150 PRINT ▲ A$(K);
160 NEXT ▲ K
170 DATA ▲ Q, W, E, R, T, Y, U, I, O, P, A, S, D, F, G, H, J,
    K, L, Z, X, C, V, B, N, M
```

Not only does the computer have the ability to distinguish between higher and lower NUMBERS, but it can also order LETTERS. That is, it is able to rank in alphabetical order — or reverse alphabetic order!



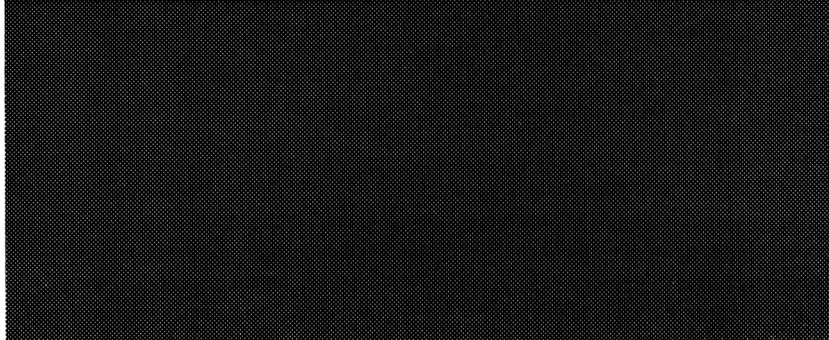
If you compare the numeric and alphabetic bubblesorts you will see how the arrays vary, e.g. DIM A(5) and DIM A\$(26).

Look also at lines 60 and 70. You should be able to experiment with these programs.

The preceding programs have served as a introduction to simple uses of DIM, READ and DATA. Experiment with some ideas of your own.

**N**OW YOU  
CAN  
EXPERIMENT





## CHAPTER 19

# How to Adapt Programs

## INTRODUCTION

We feel that many educational programs do not fit the individual's needs. We have shown already how to alter simple number programs according to the level of difficulty required. Here is a more complex program which you may not fully understand yet, but you can adapt to your needs:

```
10 REM : Multi choice (random) program
20 DIM ▲ A$ (10)
30 DIM ▲ B$ (10)
40 DIM ▲ T(5)
50 GOSUB ▲ 380
60 CLS
70 FOR ▲ N = 1 ▲ TO ▲ 5 : LET ▲ T(N) = 0 : NEXT ▲
  N
80 FOR ▲ N = 1 ▲ TO ▲ 5
90 LET ▲ R = INT (RND * 10) + 1
100 FOR ▲ Z = 1 ▲ TO ▲ 5
110 IF ▲ T(Z) = R ▲ THEN ▲ GOTO ▲ 90
120 NEXT ▲ Z
130 LET ▲ T(N) = R
140 NEXT ▲ N
150 LET ▲ Q = INT (RND * 5)+1
160 PRINT "CHEMISTRY QUIZ"
170 PRINT
180 PRINT "What is the chemical symbol for "; A$
  (T(Q))
190 FOR ▲ N = 1 ▲ TO ▲ 5
200 PRINT ▲ N, B$ (T(N))
210 NEXT ▲ N
220 PRINT
230 PRINT "Answer (1-5) =";
```

```

240 LET ▲ R$ = INKEY$ : IF ▲ R$ <> "1" AND ▲ R$
    <> "2" AND ▲ R$ <> "3" AND ▲ R$ <> "4" AND
    ▲ R$ <> "5" THEN ▲ GOTO ▲ 240
250 PRINT ▲ R$
260 IF ▲ VAL (R$) = Q ▲ THEN ▲ PRINT "Correct":
    GOTO ▲ 310
270 PRINT "Incorrect — wait for correct answer!"
280 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T
290 PRINT
300 PRINT "Answer was "; B$ (T (Q))
310 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T
320 CLS
330 INPUT "Another go — yes or no"; G$
340 IF ▲ G$ = "yes" THEN ▲ GOTO ▲ 60
350 CLS
360 PRINT "END OF QUIZ"
370 END
380 FOR ▲ N = 1 ▲ TO ▲ 10 : READ ▲ A$(N), B$(N):
    NEXT ▲ N : RETURN
390 DATA ▲ HYDROGEN, H, POTASSIUM, K,
    CARBON, C, NITROGEN, N, OXYGEN, O
400 DATA ▲ SULPHUR, S, HELIUM, He, COPPER, Cu,
    PHOSPHORUS, P, ZINC, Zn

```

## A MORE COMPLEX PROGRAM

Here are some examples of the screen display:

CHEMISTRY QUIZ

What is the chemical symbol for OXYGEN

- 1 O
- 2 He
- 3 K
- 4 Zn
- 5 N

Answer (1-5) = 1

Correct

CHEMISTRY QUIZ

What is the chemical symbol for COPPER

- 1 S
- 2 He
- 3 H
- 4 Cu
- 5 N

Answer (1-5) = 5

Incorrect — wait for correct answer!

Answer was Cu.

## ***E*XPANATION**



What the program does:

(a) It stores in its DATA statement chemical names and their symbols — see lines 390 and 400.

(b) The computer then chooses the full word for the symbol at random and asks “What is the chemical symbol for HYDROGEN (for instance). You are then given a choice of 5 symbols to choose from and asked to press the appropriate number from 1 to 5.

The correct answer is always one of the choices.

(c) If you are correct the computer tells you; if you are incorrect the answer is given.

(d) You are then asked if you want another go. If you type “yes” you get another go. If you type “no” the program ends.

Type it carefully and RUN it.

## ***T*HE PROGRAM TEACHES AS WELL AS TESTS**



The beauty of a program such as this is that it teaches as well as tests. You could easily learn these symbols without any prior knowledge. You can, of course, easily increase the number of items in the DATA line so that more questions can be asked — you will have to change the DIM statements and the random selection (line 90).

## HOW TO ALTER THE PROGRAM

Let us say that you wanted to ask questions about the capitals of Europe — this is what you would need to alter:

LINE 180 to read ; 180 PRINT "What is the capital of "; A\$(T(Q))

LINE 390 and 400 — DATA FRANCE, PARIS, RUSSIA, MOSCOW, etc. to a total of ten countries and capitals.

NOTE: It is not necessary to use speech marks unless you are using a space, colons, commas etc. See DATA lines in the next program as an example.

## ANOTHER EXAMPLE

This program is shorter and simply sets up two arrays on lines 30 and 40, one for French words and the other for the English equivalent. You are asked to give the French for the English word, which is presented on the screen in the form of a question. Ready to test your French?

```
10 REM : Vocabulary
20 CLS
30 DIM ▲ F$(5)
40 DIM ▲ E$(5)
50 DIM ▲ C(5)
60 FOR ▲ K = 1 ▲ TO ▲ 5
70 READ ▲ F$(K), E$(K)
80 NEXT ▲ K
90 LOCATE ▲ 8, 12
100 PRINT "French Vocabulary Quiz"
110 FOR ▲ L = 1 ▲ TO ▲ 5
120 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T : CLS
130 LET ▲ T = 0
140 LET ▲ R = INT (RND (1) * 5) + 1
150 IF ▲ C(R) = 1 ▲ THEN ▲ GOTO ▲ 140
160 LET ▲ C(R) = 1
170 PRINT "What is the French for "; E$(R)
```

```

180 PRINT
190 INPUT "Answer is ...."; A$
200 PRINT
210 IF ▲ A$ = F$(R) THEN ▲ GOTO ▲ 260 ▲ ELSE ▲
    LET ▲ T = T+1
220 IF ▲ T = 3 ▲ THEN ▲ GOTO ▲ 240 ▲ ELSE ▲
    PRINT "Incorrect — try again"
230 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T : CLS :
    GOTO ▲ 170
240 PRINT "Hard luck — the correct answer is "; F$(R)
250 GOTO ▲ 130
260 PRINT "Well done — answer correct!": NEXT ▲ L
270 FOR ▲ T = 1 ▲ TO ▲ 5000 : NEXT ▲ T : CLS
280 LOCATE ▲ 16, 12 : PRINT "The End" : GOTO ▲
    280
290 DATA "le jardin", "the garden", "le chat", "the
    cat", "le chien", "the dog"
300 DATA "la robe", "the dress", "la cravate", "the tie"

```

If you answer correctly it is confirmed in line 260. If your answer is incorrect you are asked the same question again — after three attempts the answer is given. The program randomises the questions but, because of the array DIM C(5) in line 50 and lines 150 and 160, you will not be asked the same question more than once. Note also that this program is very particular. You must INPUT your answer EXACTLY as it appears in the DATA lines. Using capital letters or slight spelling mistakes will give the response "incorrect". You will have to use the ESC key to get out this program.

If you want to adapt this program for more questions you will need to look at lines 30, 40, 50, 60, 110, 140 and of course, the DATA lines. Line 170 will also have to be changed if you want to ask different questions. You don't have to change E\$ and F\$. Remember that these "string variables" can stand for anything. They stand for English and French in our program merely to help you to understand the program.

## HOW TO ADAPT IT

This program is capable of handling other types of material. For example, your question could be "What is the French/German/Arabic for ...." or it could be "Who was monarch in ....". The DATA lines will of course have to have two blocks of information, one representing the question and the other the answer, e.g. "the house", "das Haus" or "1627", "Charles 1".

## VARIATIONS ON A THEME

The next two programs are variations on a theme, one giving a score, the other using a scoring routine to increase the degree of difficulty:

```
10 REM Score
20 CLS
30 LOCATE ▲ 12, 10 : PRINT "Multiplication"
40 LET ▲ S = 0
50 FOR ▲ Q = 1 ▲ TO ▲ 10
60 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T
70 LET ▲ A = INT(RND(1)* 10)+ 1
80 LET ▲ B = INT(RND(1)* 10)+ 1
90 PRINT ▲ Q ▲ TAB (7); "What is"; A; "▲";B
100 PRINT
110 INPUT "Answer is ...."; C
120 PRINT
130 IF ▲ C = A*B ▲ THEN ▲ LET ▲ S = S + 1 ▲ ELSE
    ▲ GOTO ▲ 150
140 PRINT "Correct" : GOTO ▲ 160
150 PRINT " Incorrect — the answer is "; A * B
160 NEXT ▲ Q
170 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : CLS
180 LOCATE ▲ 3, 15: PRINT "You scored" ;S; " out of
    10": GOTO ▲ 180
```

```
10 REM : Increasing the difficulty
20 CLS
30 LOCATE ▲ 9, 10 : PRINT "Multiplying by
    degrees"
40 LET ▲ S = 0
50 FOR ▲ Q = 1 TO 10
```



```

60 GOSUB ▲ 1000
70 GOSUB ▲ 5000
80 NEXT ▲ Q
90 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : CLS
100 LOCATE ▲ 3, 15
110 IF ▲ S < 8 ▲ THEN ▲ PRINT "You only scored"
;S; "out of 10. Try again!" : GOTO 40
120 IF ▲ S > 8 ▲ THEN ▲ PRINT "Well done —";S;"
out of 10"
130 PRINT "Try the next ten — they are harder"
140 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : LET ▲
S=0
150 FOR ▲ Q = 1 ▲ TO ▲ 10
160 GOSUB ▲ 2000
170 GOSUB ▲ 5000
180 NEXT ▲ Q
190 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : CLS
200 LOCATE ▲ 3, 15
210 PRINT "You scored" ;S; "out of 10 this time"
220 END
1000 LET ▲ A = INT (RND (1) * 10)+ 1
1010 LET ▲ B = INT (RND (1) * 10)+ 1
1020 RETURN
2000 LET ▲ A = INT (RND (1) * 100) + 1
2010 LET ▲ B = INT (RND (1) * 10) + 1
2020 RETURN
5000 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T: CLS
5010 PRINT ▲ Q ▲ TAB (7); "What is" ;A; """;B
5020 PRINT
5030 INPUT "Answer is ...."; C
5040 PRINT
5050 IF ▲ C = A * B ▲ THEN ▲ LET ▲ S = S+1 ▲ ELSE
▲ GOTO ▲ 5070
5060 PRINT "Correct" : GOTO 5080
5070 PRINT "Incorrect — the answer is"; A*B
5080 RETURN

```

Note how in the second program we have written the random number selection lines and the question and answer lines as subroutines using GOSUB. This way saves typing time and also leaves plenty of scope to insert further subroutines should you wish to increase further the degree of difficulty.

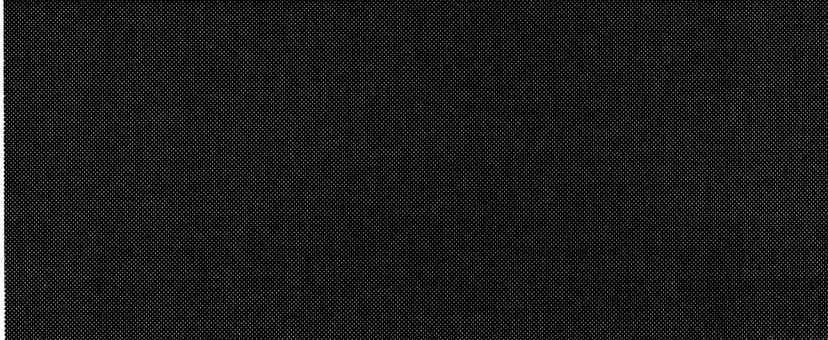
# A TEST OF LOGIC

The final program is a guessing game which is also a test of logic! Who can guess the number in the least attempts:

```
10 REM : Number guessing game
20 CLS
30 LOCATE ▲ 15, 12 : PRINT "Guess the number"
40 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : CLS
50 LET ▲ N = INT (RND * 100) + 1
60 LET ▲ X = 0
70 LOCATE ▲ 5, 12 : PRINT ▲ TAB (5) "I am thinking
  of a number between"
80 PRINT ▲ TAB (17) "1 and 100"
90 PRINT
100 LET ▲ X = X + 1
110 IF ▲ X = 10 ▲ THEN ▲ GOTO ▲ 170
120 INPUT "Have a guess"; G
130 PRINT
140 IF ▲ G = N ▲ THEN ▲ CLS : LOCATE ▲ 1, 12 :
  PRINT "Well done. You got the number in
  ";X;"guesses" : GOTO▲180
150 IF ▲ G < N ▲ THEN ▲ PRINT "You have guessed";
  TAB(35); G; "but that is too low — try again":
  PRINT : GOTO ▲ 100
160 PRINT "You have guessed"; TAB (35);G; "but that
  is too high — try again": PRINT : GOTO▲ 100
170 FOR ▲ T = 1 ▲ TO ▲ 3000 : NEXT ▲ T : CLS :
  LOCATE ▲ 6, 12 : PRINT "Sorry, no more guesses
  left"
180 PRINT
190 PRINT ▲ TAB (10) ; "The answer was "; N
200 END
```

You can adapt this to give any number of attempts by altering line 110 and you can also alter the limits of the numbers selected by changing line 50.





# INDEX

Adaptation of program .....	131	Example programs .....	59
Addition .....	37	—Addition .....	69
AND function .....	73	—Alphabetical bubblesort .....	128
ASCII .....	103	—AND demonstration .....	73
AUTO command .....	91	—Averages .....	123
AUTO-repeat .....	10	—Border demonstration .....	95
BORDER command .....	95	—Branching .....	63
Brackets .....		—CHRS demonstration .....	103
Branching .....	60	—Circle demonstration .....	109
Bubblesort .....	126	—Design demonstration .....	110
CAPSLOCK .....	8	—Division demonstration .....	35
Channel .....	116	—FOR/NEXT demonstration .....	44
CHR# .....	103	—French vocabulary .....	66
CIRCLES .....	109	—GOSUB .....	78
Circumference .....	109	—Guessing game .....	138
CLR key .....	16	—IF/THEN demonstration .....	72
CLS .....	12	—INK demonstration .....	97
Colour .....	95	—INPUT demonstration .....	31
Colouring circles .....	110	—INT demonstration .....	48
Comma .....	23	—Kings and Queens .....	56
Command words .....	13	—LOCATE demonstration .....	24
Conditional statements .....	70	—Multi-choice Chemistry Quiz .....	131
Co-ordinates .....	107	—Multi-choice French Quiz .....	66
COPY key .....	17	—Multiplication .....	50
COPY Cursor .....	17	—Multiplication tests .....	136
COS function .....	109	—Numeric bubblesort .....	126
Counting .....	64	—Numeric variables demonstration .....	46
CRSR keys .....	8	—Operators demonstration .....	73
CTRL Key .....	8	—OR demonstration .....	74
Cursor .....	7	—PAPER demonstration .....	97
		—PEN demonstration .....	98
		—PRINT a list of numbers .....	40
		—PRINT demonstration .....	127
DATA command .....	123	—Recharge demonstration .....	99
DATACORDER .....	83	—RND demonstration .....	48
Decimal .....	51	—Scale .....	114
DEG function .....	109	—Shopping list .....	54
DEL key .....	16	—String variables .....	58
DIM command .....	125	—Sur le pont d'Avignon .....	115
Division .....	39	—Word processor .....	57
Duration .....	116		
EDIT command .....	17	Flashing colours .....	97
ELSE function .....	71	FOR command .....	43
END command .....	66	Formulae .....	49
ENT command .....	117	Fractions .....	52
ENTER KEY .....	10		
ENV command .....	116	GOSUB command .....	77
ESC KEY .....	8	GOTO command .....	63
		Graphics .....	103
		Graphics cursor .....	105

IF command .....	64	RND command .....	46
INK command .....	99	RUN .....	11
INPUT command .....	31	SAVE command .....	84
INT function .....	48	Screen clearing .....	12
Keys .....	8	Screen layout .....	24
LET command .....	52	Set up .....	7
Line numbers .....	11	SHIFT key .....	8
LIST command .....	19	Signs .....	
LOCATE command .....	24	—+ .....	39
Logical operators .....	74	—- .....	40
Long multiplication .....		—• .....	40
Loops .....	60	—/ .....	36
Lower case .....	8	—= .....	71
Memory clearing .....	16	—<> .....	71
MODES .....	89	—< .....	71
Multiplication .....	40	—> .....	71
Music .....	114	—<= .....	71
Musical notes .....	114	—>= .....	71
Musical scale .....	114	—? .....	92
NEW command .....	13	SIN function .....	109
NEXT command .....	43	SOUND command .....	113
NOT function .....	75	Sound effects .....	118
Note .....	116	Spaces .....	12
Numeric variables .....	40	Speech marks .....	26
OR function .....	73	String variables .....	69
ORIGIN command .....	111	Subtract .....	38
PAPER command .....	96	SYNTAX ERROR .....	9
PEN command .....	97	TAB command .....	25
Pixels .....	105	Tapes .....	83
PLOT command .....	106	THEN command .....	63
PRINT command .....	21	Timing .....	63
Printer .....	85	Tone envelope .....	117
Punctuation marks .....	23	Typing errors .....	15
Quotation marks .....	26	Upper symbols .....	9
Radius .....	109	VERIFY command .....	83
Random numbers .....	46	Volume .....	116
READ command .....	123	Volume envelope .....	116
READY .....	7	Word processor .....	57
REM command .....	13	XPOS command .....	108
RENUM command .....	22	YPOS command .....	108
RETURN .....	77		









**This book is designed to guide the new Amstrad user through the first few weeks of programming – from the moment the machine is plugged in to a proficiency in basic programming.**

**Topics include:**

- ★ **Introducing the keyboard**
- ★ **Useful computing aids such as recorders and printers**
- ★ **Using easy commands**
- ★ **How to build a program**
- ★ **Programming techniques with advice on variables, looping, branching and counting**
- ★ **Practical applications**

**This book is essential reading for any first-time Amstrad user. It manages to avoid technical jargon and is also great fun to use. All example programs are presented in quiz, game or self-questionnaire form, fully illustrated for easy reference.**

**'... an excellently presented introduction to Basic ... the answer to all your introductory problems.'** WHAT MICRO

**'... remarkably understanding and sympathetic little book. It starts right at the beginning.'** WHICH MICRO



1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

1998

1999

2000

2001

2002

2003

2004

2005

2006

2007

2008

2009

2010

2011

2012

2013

2014

2015

2016

2017

2018

2019

2020

2021

2022

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

2033

2034

2035

2036

2037

2038

2039

2040

2041

2042

2043

2044

2045

2046

2047

2048

2049

2050

2051

2052

2053

2054

2055

2056

2057

2058

2059

2060

2061

2062

2063

2064

2065

2066

2067

2068

2069

2070

2071

2072

2073

2074

2075

2076

2077

2078

2079

2080

2081

2082

2083

2084

2085

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

2099

2100

2101

2102

2103

2104

2105

2106

2107

2108

2109

2110

2111

2112

2113

2114

2115

2116

2117

2118

2119

2120

2121

2122

2123

2124

2125

2126

2127

2128

2129

2130

2131

2132

2133

2134

2135

2136

2137

2138

2139

2140

2141

2142

2143

2144

2145

2146

2147

2148

2149

2150

2151

2152

2153

2154

2155

2156

2157

2158

2159

2160

2161

2162

2163

2164

2165

2166

2167

2168

2169

2170

2171

2172

2173

2174

2175

2176

2177

2178

2179

2180

2181

2182

2183

2184

# AMSTRAD

# CPC



**MÉMOIRE ÉCRITE**  
**MEMORY ENGRAVED**  
**MEMORIA ESCRITA**



<https://acpc.me/>

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.

[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.

[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.