# 3 BASIC primer

A brief introduction to programs written using the AMSTRAD BASIC language

Subjects covered in this chapter

* The rules of syntax and descriptions of syntax
* The PRINT commands, streams and display formatting
* ZONEs

## 3.1 BASIC basics

The fundamental relationship of the CPC464's built-in BASK! to the internal operation of the computer is introduced in Appendix II. If you have not previously programmed a computer, then we'll try and help you along gently - but it may be necessary to make some assumptions that might cause confusion to the novice. If so, we suggest that you browse through the many books available that are designed to introduce the fundamentals of programming to the newcomer.

You should be able to work through this chapter following the simple exercises herein without requiring a complete appreciation of what's happening - although the more of the rules you learn, the easier it will all become.

BASIC is the language that comes supplied 'built-in' with your CPC464. It's right there when you switch on, and makes it's presence felt with the prompt word:

***Ready***

BASIC is the simplest language to learn. It is organised with clearly defined words and 'grammar', and operates completely logically - as long as you understand the rules.

AMSTRAD BASIC is capable of executing the commands given in Chapter 8. Each command is identified by one or more leading keywords, and may have a number of parameters - some of which are optional. In general each parameter may be an expression, involving constants, variables and functions. Combinations of numbers and letters are known as strings, and various forms of numeric data types are supported including decimal, hexadecimal and binary types.

Files on the cassette are handled sequentially ie one after another - as opposed to randomly, where a file can be selected directly from the midst of many others without first passing through the unwanted files.

## 3.2 The structure of a BASIC program

Program instructions are presented to BASIC on lines. A line may comprise several commands, separated by colons, limited only by the maximum line length of 255 characters. A 'character' is a number, letter, space. In Direct Mode lines are typed in from the keyboard and must not commence with a number. In Program Mode lines are read from the current program in memory, and are executed in strict sequence according to the number that appears as the first entry on the line.

AMSTBAD BASIC allows the user to add and remove lines from the program whilst in direct mode, and to amend existing lines. Before RU Nning or L I S Ting a program, BASIC will internally re-organise the order of the lines in ascending numerical sequence, regardless of the order in which they were entered.

## 3.3 Line Input.

BASIC accepts lines of up to 255 characters, terminated by [ENTER]. During line input, it is possible to edt the current line, and to use the COPY cursor facilities to insert characters from elsewhere on the visible screen display - see chapter one section 1.2.7.

All keywords must be delimited by a separator - which is either a space, mathematical operator (+ — etc.) or other recognised character. This is because it is permissable to use variables where a keyword (or reserved word) forms part of the variable - obviously a reserved word cannot be a variable unless 'surrounded to prevent the computer from recognising the start and end of the instruction.

Keywords may be entered either as lower case letters (un[SHIFT]ed), or CAPITAL letters ([SHIFT]ed).

The command P R I NT may be abbreviated to a question mark ? and when used in this form, the delimiter is not required. Mathematical operators (+—*/MOD\) also perform the function of delimiting the keyword, so the following is valid, although not encouraged, since it may encourage bad habits in program entry where spaces are necessary:

```
for n= 1 to 50:?n:next
```

Similarly, a single quote mark ' ([SHIFT]7) can be used as a substitute for :REM in remark statements.

Additional spaces will be ignored, and may used to 'format' the program listing to indicate areas of loops etc.

## 3.4 Terminology

In order to describe BASIC commands and keywords, a formal but simple terminology must be used. Each command is described as it should appear when entered at the keyboard, with any variable or optional parts shown by various types of brackets or parentheses which refer to items detailed in' the subsequent description.

These items are represented by the various terms, and enclosed in angle brackets ‹›. For example, in various places an expression which yields a numeric result is required, this is represented by :

{numeric expression,

Anything not enclosed by angle brackets is required as given. For example the S T 0 P command takes the form :

***STOP***

Where there is an optional part in a definition, the optional part is enclosed in square brackets. For example, if a numeric expression were to be optional then it would appear:

[<numeric expression,]

If an optional part may be repeated (so may appear any **number** of times, including none at all) an asterisk is included after the closing square bracket. For example, a string of digits, requiring at least one digit, would appear :

**‹digit›[‹digit›]***

Such an expression might be:

34
or 344
or 345678 etc

In many places a list of items separated by commas is used. A short form is used, which is best illustrated by example, thus :

‹list of: (expression) means: **‹expression›[,‹expression›]*** or:
‹list of: **[#]‹number›** means: **[#]‹number›[,[#]‹number›]***

an example of which is:

**3,4**
or **3,4,4**
or **3,4,5,6,7,8** etc

The list may be a single object. If the list contains more than one object, then each additional object must be preceded by a comma delimiter, since it marks the boundary or limit between items that the computer must treat separately.

Numbers may be expressed in several forms:

a. (unscaled numbers,

. . . . ..**are** numbers without exponentiation - i.e. those with no exponent part.

b. <scaled numbers) are numbers that are 'raised to **the** power' or 'scaled' using the form:

**2E4** (2 times 10 to the power of 4, or 2.10")

.... .the exponent part may be either positive, or negative.

c. ‹based numbers) are numbers that are declared to be either binary or hexadecimal (see Appendix II):

| | |
|---|---|
| Decimal form (the default condition) | **100** |
| Hexadecimal form | **&64 *or* &H64** ..... (the 'H' is optional) |
| Binary form | **&X1 10 0100** ..... (the 'X' is compulsory) |

## 3.5 Practice makes perfect - introducing PRINT

In order to demonstrate the way this terminology works, here are some examples of BASIC at work.

One BASIC keyword that displays most of the features of the terminology is the simple command PRINT. A command is a BASIC keyword or statement that will operate with the computer in either direct entry mode or program mode. A function requires the presence of a command to 'invoke' the function eg:

```
PRINT FRE("")
```

To get the CPC464 to give you the answer to a question, you must tell it three things:

1. Where you want the answer to appear - ie the screen, the printer or 'elsewhere'
2. You must give the computer the 'data' to work with
3. You must tell the computer what to do with the data

***PRINT*** is used to tell the computer to put the results of a command onto a specific output 'stream' - where a stream is identified by a number from 0 to 9, which is described in the analysis of BASIC as a ‹stream expression> - which is the number that defines the particular stream to be used:

0...7 are text streams to text 'windows' that have been previously set up with the ***WINDOW*** command.

8 is the parallel printer port, and can only be used if a Centronics compatible printer has been attached correctly.

9 is a cassette output file- which must have been properly opened earlier in the program.

A concise form of the PRINT command (not employing the '***PRINT USING***' format template) is thus:

***PRINT [#(stream*** expression›][‹print list›]

. . ..so far. The square brackets mean that you don't have to declare the <stream expression,, nor do you actually have to give the PRINT command a list of anything to print at all (in which case the computer responds with an empty line -try it). If you don't direct the output to a particular stream the CPC464 assumes you mean stream **#0**, the default screen stream. Try this line (remember to press **[ENTER]** at the end to tell the computer to deal with the instructions you have given it):

```
PRINT "HELLO"
```

The computer responds:

***HELLO***

Note that the quotations marks "" have been omitted from the 'output stream'. Double quote marks are used by BASIC strictly to delimit (ie. mark the beginning and end of) constant strings.

Now type:

```
PRINT #Ø,"HELLO"
```

. . .and the result is the same.

But type:

```
PRINT #4,"HELLO"
```

. . ..and the computer has put the result at the top left of the screen because this is the first entry on screen stream number 4 - which defaults to cover the entire text area unless previously defined by the W I N D 0 W command. The starting position for all text on a screen stream is the top left - and stream 4 is as yet unused. The sign-on message used stream 0 (the default), so the text was sent to the stream that appeared after the characters already displayed there.

This feature of AMSTRAD BASIC is particularly powerful, since it allows complex screen displays to be built using simple commands and definitions, that contribute towards ease of producing a tidily formatted screen display.

BASIC will print anything you put inside the double quotes without 'taking any notice' of it. Reserved words may be used in the (print list):

```
PRINT "4*4"
```

and the computer responds with:

```
4*4
```

In order to instruct BASIC to operate on the multiplication, 4 multiplied by 4, the numbers and the operator (the multiplication symbol *) must be left accessible to BASIC, and directed to a print stream:

```
PRINT 4*4
```

. . ..and BASIC returns the answer

```
 16
```

Notice that the number is offset one column in from the left margin, since BASIC reserves this space for the minus symbol (-) that identifies a negative result.

The P R I N T command has many other forms, using the full formatting facility of a thorough implementation of the standard series of templates.

The <print list› in the P R I NT command refers to the list of items to be printed. This may be either a number, variable or a string expression - which means a previously defined string variable (eg H E L LO$), or anything else contained within double quotes "".

P R I NT US I NG tidies up numbers into a fixed format for printing, so that columns can be aligned, and unwanted remainders or fractional parts discarded.

## 3.6 The PRINT USING formatand ZONES

At switch-on, BASIC sets the screen Z 0 N E width into positions 13 columns wide. When the print instruction includes a comma , the next item is tabbed forward to the next ZONE position. If there are fewer columns available on a line than specified in the Z 0 N E command, then BASIC will start the next item to be printed on a new line. It does not break the item across the edge of a line.

With no U S I N G format specified, BASIC prints positive numbers preceded by a space, negative numbers preceded by a - sign. All numbers are followed by a space. The decimal point is omitted if there is no significant fractional part of the item to be printed.

AMSTRAD BASIC does not support the [TAB] key as a column tab, since there is considerable lack of unity on the meaning and function of this feature in various dialects of BASIC. Pressing the [TAB] key prints a right arrow character →(the same as [CTRL] and the I key together), but otherwise has no purpose in AMSTRAD BASIC.

## 3.7 PRINTTAB( (integer expression, )(‹print list›)

The effect of this is best illustrated by example. Enter this, and see the result:

```
5 MODE 2: INK 1,0: INK 0,9
10 FOR N=l TO 5
20 ZONE 40
30 PRINT TAB(N*4)"HI",N
40 NEXT
```

This program illustrates both the ZONE together with the comma , and the T A B () functions at work. Run it again with line 10 altered to:

```
10 FOR N=-5 to 5
```

The TAB instruction moves the start of the P R I NT forward by the number of spaces specified in the (integer expression>. (ZONE may be set in the range 1....255 - see the definitions in Chapter 8.)

The P R I NT US I N G form is used to format the results of calculations where real number results would produce an untidy array of decimal places. It is a complicated concept that is best appreciated through practical examples, since the technical form:

PRINT [#stream expression,, ] [‹**print list**›][‹**using clause**›][‹**separator**›]

Could be justly described as non-user-friendly, especially since the using clause, further divides down to:

U S I N G string **expresion**› ; [using list,]

where the using **list**› further subdivides to

‹**expression**›[‹**separator**›‹**expression**›]*

Try the following:

```
PRINT 123.456, USING "###.##";4567.896
```

the result..

```
 123.456        %4567.90
```

. . ..**illustrates** several points. Firstly the item to be printed before the US I NG clause is not affected. Secondly, the US I NG clause allocates the number of 'output' positions available for the print item following (which may be a variable of course), and if this item exceeds the allocated space to the left of the decimal point, it still displays, but uses the % to indicate that an overflow has **occured**. Next, the comma after 12 3 **.4** 5 6 caused the following number to be printed at the start of the next print zone. If it had been a semi-colon, the number would have been printed alongside the 12 3 **.4** 5 6 one space to the right. Numbers are always separated by one space when printed on the same line - for obvious reasons!

Finally note that the expression is rounded and does not simply discard the digit(s) beyond the last place used in the format.

Try this:

```
PRINT 123.456, USING "#####.##+";4567.899
```

. . ..**and** the sign appears at the end of the formatted number. A minus sign will precede a negative number by default.

P R I N T U S I N G is a very useful facility in producing any form of tabulated result output, It will always warn you if the format specified is too restrictive (using the % ).