# Introducing BASIC

**Okay, now you've decided to buy a home computer — how do you go about using it?**

BASIC (Beginner's All-purpose Symbolic Instruction Code) was developed as an instructional language at the Kiewit Computation Centre of Dartmouth College in the USA. It is probably the most used computer language in the world today. Most microcomputers are able to execute programs written in BASIC, and virtually all time-shared systems and mainframe computers in universities and colleges run BASIC.

Like most computer languages, BASIC is not standardised, and so some versions have different commands, statements or functions, or differ in grammatical details from others. We are going to concentrate here on microcomputer versions of BASIC, as most people will have access to a micro.

## Nouns and Verbs

When you learnt English at school, one of the things you learnt was grammar. Every language has a grammar, a set of rules which tells you how to string words together to make sentences, and BASIC is no exception.

Rule number one of BASIC grammar: every line begins with a line number. Line numbers can usually be any number between 1 and 32 767, but usually you start at 10 or 100 and increase in 10s. This is so that if you want to insert a line between, say, 20 and 30, you can number it 25. You couldn't put a line between 20 and 21.

The nouns of BASIC, the objects it operates on, are called variables. In 'standard' BASIC (if such a thing can be said to exist), variables usually are numbers, and are named by a letter, or a letter and a number. For example, a program could use three variables called A, A1 and A2, and the BASIC interpreter will distinguish between all three of these.

Some BASIC interpreters are a bit more sophisticated and allow longer variable names such as VALUE, TAX or POWER. This is useful when reading through a program, as you can see what each variable stands for immediately.

## by Les Bell

In English, every sentence must contain a verb (usually). BASIC is the same; each line is a short sentence telling the computer to do something, and it contains a verb (usually). For example:

250 GOTO 50

This is what is called, in BASIC, a statement. It starts with a line number, and then follows an instruction, a clear, unambiguous instruction, to the computer — GOTO 50. This instructs the computer (or the BASIC interpreter, actually) to go to line 50. Now you know why every line is numbered.

BASIC statements can use a variety of different verbs. Here are some examples:

LET (as in 10 LET Y=1) The LET statement is optional in some versions of BASIC, which let you type 10 Y=1

= The = symbol is called the "assignment' symbol. This avoids confusion with the equals symbol. For example, BASIC lets you write (or type) 20 LET X=X+1. Now, in mathematics, it is obviously not true to say that X=X+1. In BASIC, it means that variable X is assigned the previous value of X, plus one. In other words, X is incremented (this is a useful function for counting things like runs through a loop of a program). It does not mean that X is equal to X+1.

FOR The FOR instruction is used to set up program loops. For example (excuse the pun):

```
10 FOR I=1 TO 10
20 INPUT A(I)
30 NEXT I
```

This short program section will execute line 20 ten times, inputting A(1), A(2), A(3), and so on up to A(10). The NEXT I statement marks the end of the loop (the I is often optional, but is useful to include).

INPUT The INPUT statement halts the running program, prints a question mark (usually) and waits for the user to type in some data, often the answer to a question. It may also automatically print the question. Like this: 40 INPUT "HOW MANY BANANAS:",A would print up HOW MANY BANANAS: and wait for you to type in the answer, followed by a carriage return, which is the signal for the program to start running again. The number you type in is stored as variable A.

GOSUB This is a very useful and powerful statement. A block of program which is used often in different parts of the main program may be written as a subroutine and then called by the program. A silly example will explain:

```
10 GOSUB 500
.
.
200 GOSUB 500
.
.
340 GOSUB 500
.
.
500 LET Y=X↑2+5↑+3
510 PRINT "Y = ", Y
520 RETURN
```

In this example, the subroutine simply evaluates a quadratic and prints the result each time it is called by the main program. ▶

# BASIC

PRINT This statement may include a list of expressions, variables or constants separated by commas or semicolons. For example:

35 PRINT "THE VALUE OF X IS ", X would print THE VALUE OF X IS 4.6239242E+09 or something similar. Note that the literal string which is to be printed is enclosed in quotes.

## Strings

As well as dealing with numeric variables for calculation, BASIC also allows you to handle words and text. This kind of information is called a string variable, and these are named differently from numeric variables by adding a dollar sign at the end of the name. So, for example, A$ and A1$ are string variables, and are different from A and A1 which are numeric.

Strings can be input, printed, compared, concatenated (strung together), and compared. For example:

100 INPUT "HELLO THERE. WHAT'S YOUR NAME?", N$
200 PRINT "NICE TO MEET YOU,", N$

is a good way of letting your computer introduce itself. Or

450 IF A$="YES" THEN 480
460 IF A$="NO" THEN 520
470 GOTO 440

is a way of letting the user answer yes or no to a question from the computer, with the computer making the appropriate response.

## Arrays

As well as variables which consist of a single number, many BASICs allow the creation of variables which are actually tables of values. These are called arrays, and they have a variety of uses. Arrays have to be dimensioned before use; in other words, the user has to tell the computer how many values the table will contain in any direction. Some BASICs limit arrays to two dimension, others allow as many dimensions as you require to the limit of memory.

For example, a program to find the inverse of a 3x3 matrix might start like this:

100 DIM A(3,3)
110 FOR I=1 TO 3
120 FOR J=1 TO 3
130 INPUT A(I,J)
140 NEXT J
150 NEXT I

This is rather crude, as a well designed program would also print information to prompt the user as he inputs data, but it does illustrate the general principle.

Similarly, one can use arrays in LET statements:

40 LET A(I,J)=B(I,J)+C(I,J)

would add corresponding elements of two matrices together.

## BASIC Arithmetic

Arithmetic in BASIC is fairly straightforward; assignment statements (LET statements) read from left to right following the rules of algebraic hierarchy (sines, cosines, powers, etc are calculated first, followed by multiplication and division, then addition and subtraction, unless this sequence is altered by the inclusion of brackets).

Most BASICs offer a wide range of operators: numeric operators such as +, −, x, /, and ↑, relational operators such as =, <, >, ≥, >=, =>, <=, =<, which give a result of 1 if the expression is true and 0 if it is false, and the Boolean operators AND, OR and NOT which also result in 1 if true and 0 if false.

Some examples may help to make these clear.

50 IF X>10E6 THEN END

will end a program if too large a value is input.

540 IF A(3)>A(2) AND A(2)>A(1) THEN PRINT "DECREASING"

will check to see that three elements of an array are in decreasing order.

## Functions

As well as the standard arithmetic, relational and Boolean operators, BASIC offers a number of built-in functions. For a typical BASIC interpreter, these might be:

ABS(expr) which returns the absolute value of the expression

INT(expr) which returns the integer part of the expression

LEFT$(X$,Y) which returns the leftmost Y character in string X$ (similarly for RIGHT$(X$,Y)

LEN (X$) which returns the length of the specified string

CHR$(expr) which returns the ASCII character corresponding to the value of the expression. For example, CHR$(7) is a bell ring.

STR$(expr) which returns a string with the specified numeric value.

ASC(X$) which returns the ASCII code of the first character in the specified string.

SIN(expr) which returns the sine of the argument.

RND(expr) which returns a random number between 0 and 1. ●

## Example Program

To see how the various statements, functions, operators and commands we have learnt actually operate in practice, it is best to look at an example program. In this case, we shall write a program to do some useful engineering problem-solving – designing an impedance matching pi network.

In the listing of this program, notice that lines 100 to 170 are devoted to a short description of the program to help the user. Lines 180 to 230 actually input the data, and again, these contain prompts which are printed to assist the user. Leaving these out would make the program shorter and quicker to run, but more confusing.

Lines 240 and 250 perform checks for improper conditions on the input data, jumping to sections of program which output appropriate error messages. Again, longer error messages help the user more.

The actual calculations are performed in lines 260 to 330. These are straightforward enough and easy to follow, except for lines 300 and 310 where the calculation of X3 has been split over two lines to make it easier to read. Notice the print statements in lines 120, 170, 200, 230, 330 which space out the output to improve legibility. The colons between PRINT statements allow multiple statements on one line.

```
100 PRINT "PI NETWORK IMPEDANCE MATCHING PROGRAM"
120 PRINT: PRINT
130 PRINT "GIVEN THE VALUES OF R SOURCE AND R LOAD,"
140 PRINT "AND THE CENTRE FREQUENCY AND Q (CENTRE"
150 PRINT "FREQUENCY OVER HALF-POWER BANDWIDTH),"
160 PRINT "THIS PROGRAM CALCULATES C1, C2 AND L1."
170 PRINT: PRINT
180 INPUT "WHAT IS THE RESISTIVE SOURCE IMPEDANCE?",R1
190 INPUT "AND THE LOAD RESISTANCE?",R2
200 PRINT
210 INPUT "CENTRE FREQUENCY?",F
220 INPUT "AND THE Q?",Q
230 PRINT
240 IF R2>R1 THEN 390
250 IF Q<=SQR(R1/R2-1) THEN 420
260 X1=R1/Q
270 C1=1/(2*3.14159*F*X1)
280 X2=R2/SQR(R2/R1*(Q↑2+1)-1)
290 C2=1/(2*3.14159*F*X2)
300 X3=Q*R1/(Q↑2+1)
310 X3=X3*(1+R2/(Q*X2))
320 L1=X3/(2*3.14159*F)
330 PRINT: PRINT
340 PRINT "HERE ARE THE VALUES OF YOUR PI NETWORK."
350 PRINT "C1 = ";C1;"F"
360 PRINT "C2 = ";C2;"F"
370 PRINT "L1 = ";L1;"H"
380 END
390 PRINT "LOAD IMPEDANCE IS HIGHER THAN SOURCE --"
400 PRINT "IT CAN'T BE DONE"
410 GOTO 170
420 PRINT "Q IS TOO LOW. TRY AGAIN, WITH SENSIBLE"
430 PRINT "VALUES THIS TIME."
440 GOTO 170
```

**A Sample Run** (User input is underlined)

```
RUN
GIVEN THE VALUES OF R SOURCE AND R LOAD,
AND THE CENTRE FREQUENCY AND Q (CENTRE
FREQUENCY OVER HALF-POWER BANDWIDTH),
THIS PROGRAM CALCULATES C1, C2 AND L1.

WHAT IS THE RESISTIVE SOURCE IMPEDANCE? 500
AND THE LOAD RESISTANCE? 50

CENTRE FREQUENCY? 3.5E6
AND THE Q? 10

HERE ARE THE VALUES FOR YOUR PI NETWORK.
C1 = 9.09E-10 F
C2 = 2.74E-09 F
L1 = 2.93E-06 H
READY
```