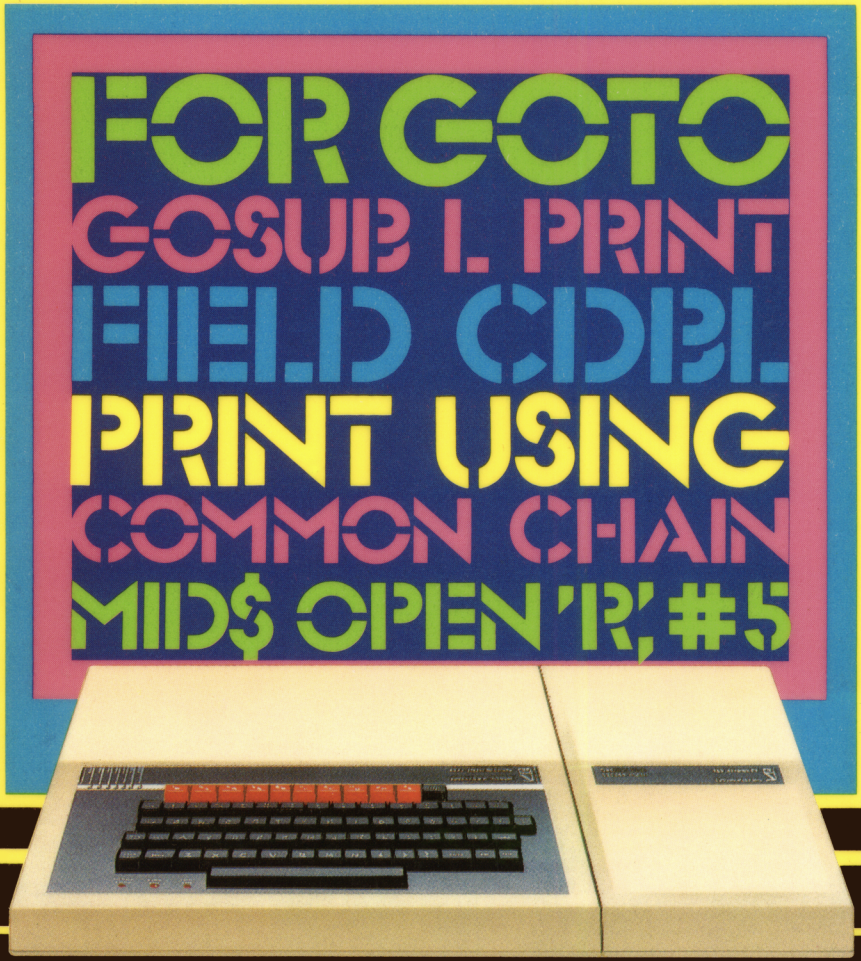
BRITISH BROADCASTING CORPORATION

MICROCOMPUTER SYSTEM

Z80 Professional BASIC USER GUIDE



Z80 Professional BASIC

for the BBC Microcomputer System

USER GUIDE

Part no 409002
Issue no 1
Date March 1984

Within this manual the term 'BBC' is used an abbreviation for 'British Broadcasting Corporation'.

All correspondence should be be addressed to:

Technical Enquiries
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

First Published 1984
Published by Acorn Computers Limited
Typeset by AMBYTE Computer Graphics

Contents

1 Introduction

The BASIC interpreter is capable of executing the commands given in Chapter 3. Each command is identified by one or more leading keywords, and may have a number of parameters. In general each parameter may be an expression, involving constants, variables and functions. Strings and various forms of numeric data types are supported, as are sequential and random file handling.

Commands are presented to BASIC on lines. A line may contain several commands, separated by colons, limited only by the line length. Lines are read from the current program in memory.

A 'Run Only' subset of BASIC is supported for use with program packages written in BASIC. There are performance advantages to using a Run Only version once a program is no longer under development.

1.1 Line Input

BASIC accepts lines from the console of up to 255 characters, terminated by a carriage return. During line input a number of control characters have special meaning, as follows:

CTRL **H**	Delete previous character on line (if any).
CTRL **I** or **TAB**	Move to next tab position, counts as one character. Tabs are set at 8 character intervals.
CTRL **J**	Move to next physical line without terminating this logical line.
CTRL **R**	Retype the line as entered so far.
CTRL **U**	Abandon the line.
DELETE	Delete previous character on line (if any).

1.2 Metalanguage

In order to describe commands and their parameters a simple metalanguage is used. The form of each command is described as it appears when entered,

with any variable or optional parts shown by 'place-holders' which refer to a concept defined elsewhere.

A concept is represented by its name enclosed in angle brackets. For example, in various places an expression yielding a numeric value is required, this is represented by: <numeric expression>

Anything not enclosed by angle brackets is required as given. For example the STOP 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 appended after the closing square bracket. For example, a string of digits, requiring at least one digit, would appear:

<digit>[<digit>]*

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

<list of: <expression> is: <expression>[,<expression>]*

or:

<list of: [#]<number> is: [#]<number>],[#]<number>]*

Note that 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.

2 The Elements of BASIC

2.1 Character Set.

BASIC assumes that the ASCII character set is in use.

Character values less than space (value 32) are treated as non-printing, and several have special meaning to BASIC.

When processing command lines BASIC does not distinguish between upper and lower case, except in strings.

2.2 White Space.

Spaces, tabs and linefeeds (or any combination thereof) are treated as <white space>. <White space> is not significant in command lines, except where it has the effect of terminating an item. Note that a carriage return immediately following a linefeed is also ignored.

2.3 Names.

Variables and Keywords have <name>s. The first character of a <name> must be alphabetic, further characters may be alphanumeric or dot. <Name>s may be at most forty characters long, all of which are significant. A <name> is terminated by any character which may not be part of a <name>.

Keywords may include other characters, and some are expressed as more than one word.

2.4 Numbers.

A <number> may be any of the following:

<unscaled number>[<type marker>]
<scaled number>[<type marker>]
<based number>
<line number>

See 2.6 below for a discussion of <type marker>s.

An <unscaled number> may take any of the following forms:

<digits>
<digits>.[<digits>]
[<digits>].<digits>

where <digits> is one or more decimal digits. <White space> is allowed in <unscaled number>s, and may be used at will as an aid to number input.

A <scaled number> may take either of the following forms:

<unscaled number>E[<sign>]<digits>
<unscaled number>D[<sign>]<digits>

where <sign> is plus or minus, and <digits> is one or more decimal digits. <White space> is allowed around the non-digit characters of the exponent part, but not within the digits portion. The value of the number is the value of the <unscaled number> multiplied by the power of 10 given by the <sign> & <digits>, the exponent part. (Note that the absolute value of the exponent part may not exceed 99.) The difference between the E and the D form of the exponent part is described below.

A <based number> may take any of the following forms:

&<octal digits>
&O<octal digits>
&H<hexadecimal digits>

where <octal digits> are digits in the range 0....7, <hexadecimal digits> are digits in the range 0..9 and letters in the range A....F (or a....f). The <based number>'s decimal equivalent may not be greater than 65535. <White space> is allowed around the non-digit characters, but not within the digits portion.

A <line number> takes the form:

<digits>

where <digits> is at least one decimal digit. Line numbers are limited to the range 0....65534.

<number>s are terminated by any character which may not appear in a <number>.

2.5 String Constants.

A <string constant> is an arbitrary collection of characters enclosed in double quotes, or starting with double quotes and terminated by end of line - in which case trailing <white space> is not included in the string. Characters outside the normal ASCII printing range may appear in string constants, with the exception of Null (character value 0).

2.6 Data Types.

BASIC handles two broad types of data, numeric and string.

Strings may be from 0 to 255 characters long. The characters in a string are represented by values in the range 0....255 (ie they are bytes). It is assumed that the ASCII character set is in use.

Numeric data may take one of three forms, Integer, Single Length and Double Length.

Integer data is held as two byte two's complement numbers, so can take values in the range −32768....+32767.

Single Length data is held in a four byte binary floating point format, with three bytes of mantissa and one byte of binary exponent. The largest absolute value representable is aproximately 1.7E+38, the smallest (other than zero) is approximately 2.9E−39. The three byte mantissa affords a little over seven decimal digits of precision.

Double Length data is held in an eight byte binary floating point format, with seven bytes of mantissa and one byte of binary exponent. The largest absolute value representable is aproximately 1.7E+38, the smallest (other than zero) is approximately 2.9E−39. The seven byte mantissa affords a little over sixteen decimal digits of precision.

The type of a <number> is implied by its form :

<Based number>s are always Integer, the unsigned value of which is mapped to its two's complement equivalent (unsigned values greater than 32767 map to 65536 − unsigned value).

<Unscaled numbers> are treated as Integer if there is no decimal point and the value can be represented in the Integer range. Otherwise the number is treated as Single Length if there are seven or less significant digits and Double Length if there are eight or more significant digits.

<Scaled numbers> are Single or Double Length, depending on the number of significant digits or the letter which introduces the exponent part. E implies Single Length, unless the unscaled part has eight or more significant digits, and is therefore already Double Length. D implies Double Length.

The type of an <unscaled number> or of a <scaled number> may be changed from its implicit type by a trailing <type marker>. The type markers are :

%	which forces the number to be rounded to the nearest Integer. If the value is beyond the range of an Integer this causes an overflow error.
!	which forces the number to Single Length format.
#	which forces the number to Double Length format.

(Note that these are the same as the type markers on variable names, see below.)

2.7 Variables.

Variables have three attributes, name, type and organisation. A <variable name> takes the form <name>[<type marker>]. The type of data referred to by a <variable name> is either given by the <type marker>, or is assumed to be the current default. The <type marker>s are:

%	for Integer
!	for Single Length
#	for Double Length
$	for String

The default type assumed depends on the first character of the <variable name>, and may be set dynamically by `DEFINT`, `DEFSGL`, `DEFDBL` and `DEFSTR` commands. The initial default for all variables is Single Length.

Variables may be <simple variable>s which have a single value, or <array variable>s which are a collection of values of the same type. Arrays are described below.

Note that the same <name> may be used for variables of different types and organisations, and that these constitute different variables.

Variables do not need to be declared in BASIC, they are brought into being by their first use, with a value of zero or null string as appropriate.

2.8 Arrays.

BASIC supports arrays of all data types. An array is a collection of data items of the same type, each of which is referenced by the array variable name followed by a suitable number of subscripts, thus :

<variable name>(<subscripts>)

where <subscripts> is: <integer expression>[,<subscripts>]
and the <integer expression> yields a value in the correct range.(Note that square brackets may be used instead of round brackets.)

The type of the data in the array is given by the type of the <variable name>.

An array may have any number of dimensions. Arrays may be declared explicitly in a `DIM` command, or implicitly by use. When an array is declared the number of dimensions and the upper bound on each is set (in an implicit declaration the upper bound is taken as 10). Thereafter any reference to the array must use the same number of subscripts, and each one must be in range. It is not possible to change the dimensions of an array.

The lower bound on all array subscripts is 0 by default, or may be set to 0 or 1 by the `OPTION BASE` command. It is not possible to change this lower bound while any arrays exist.

2.9 Type Compatability and Conversion.

BASIC generally treats all numeric types as being compatible with each other. String type is only compatible with itself.

When dealing with numbers of different types BASIC will type convert automatically. Where there are no other constraints numbers are 'widened' to the 'largest' type involved, so Integers are widened to Single Length, which are in turn widened to Double Length. Widening can never fail.

Numbers may also be forced to a given representation. Double Length to Single Length is straightforward, the mantissa is rounded from seven bytes to three bytes, while this involves a considerable loss of precision, it seldom causes Overflow. Converting either floating point format to Integer requires the number to be rounded to an integer, which must then lie in the Integer range of −32768....+32767, otherwise an Error 6 (Overflow) is generated.

2.9.1 Rounding.

There are several different ways in which a number in one representation may be rounded to another, smaller, representation. A given number X in the larger representation, if it cannot be exactly represented in the smaller, will be bracketed most closely by two values X1 and X2, where X1 < X < X2. The various rounding schemes choose one of these values thus :

a. Truncation or Round Toward Zero

The excess digits are simply discarded; positive X → X1,
negative X → X2.

b. Round Toward -Infinity

X → X1

c. Round Toward Infinity

X → X2

d. Round Away From Zero

Round toward −Infinity for negative numbers; negative X → X1.
Round toward +Infinity for positive numbers ; positive X → X2.

e. Round to Nearest.

X → X1 or X2, depending on which is nearest to X. Various schemes exist for choosing between X1 and X2 when X is exactly half way between the two. BASIC rounds away from zero in this case.

Unless otherwise stated, the term 'rounding' refers to Round to Nearest as defined above.

2.10 Unsigned Integer.

In some instances BASIC requires an Integer in the full unsigned 16 bit range of 0....65535 - for example PEEK requires a machine memory address. Floating point numbers are rounded to integer, which must then lie in the range −32768....+65535. The unsigned equivalent of the two's complement form of negative numbers is then used - that is to say negative numbers are treated as 65536+number.

2.11 Expressions.

There are four classes of expression, numeric, string, relational and logical. The precedence of operators is designed so that expressions work as might be

expected. Where the order of evaluation is not forced either by the precedence rules or by brackets, evaluation proceeds from left to right.

2.11.1 Numeric Expressions.

The syntax of numeric expressions is :

<numeric expression> is: <numeric item>[<operator> <numeric expression>]

<numeric item> is: [<unary minus>]<numeric item>
or: <numeric constant>
or: <numeric variable>
or: <numeric function>
or: (<numeric expression>)
or: (<relational expression>)

Where <numeric variable> is a reference to a simple variable or an array item of any of the numeric types. A <numeric function> is a function returning any of the numeric types.

The <operator>s, in order of precedence, are :

^	Exponentiation	Raises the value on the left to the power of the value on the right. Forces both values to Single Length before operation, and produces a Single Length result.
-	Unary minus	Notwithstanding the syntax, Unary minus has a lower precedence than Exponentiation.
*	Multiply	Floating point or Integer multiplication.
/	Divide	Floating point division. If either value is Integer it is forced to Single Length before operation.
\	Integer Division	Forces both values to Integer. The result is truncated to an Integer.
MOD	Integer modulus	Forces both values to Integer. The result is the remainder after Integer arithmetic.
+	Addition	Floating point or Integer addition.
−	Subtraction	Floating point or Integer subtraction.

Note that Exponentiation binds tightest and that Multiply and Divide have equal precedence, as do Addition and Subtraction. Except where the operator

has particular requirements, the values on either side are forced to the same type by widening.

Floating point division by zero generates an Error 11. If there is an `ON ERROR GOTO` set, then it is invoked in the usual way. If there is no `ON ERROR GOTO` set, this does not terminate execution, as the zero may be the result of underflow in a previous operation. The 'Division by zero' message is generated, the largest representable number (approx. 1.7E+38) with appropriate sign is returned, and execution continues.

Integer division by zero generates an Error 11, in the usual way.

In the event of overflow when the arguments are Integer the values are widened to Single Length and the operation repeated.

In the event of overflow when the arguments are floating point an Error 6 ('Overflow') is generated. If there is an `ON ERROR GOTO` set, then it is invoked in the usual way. If there is no `ON ERROR GOTO` set, the 'Overflow' message is generated, the largest representable number (approx. 1.7E+38) with appropriate sign is returned, and execution continues.

2.11.2 String Expressions.

The syntax of string expressions is :

<string expression> is: <string item>[+<string expression>]
<string item> is: <string variable>
or: <string constant>
or: <string function>
or: (<string expression>)

Where <string variable> is a reference to a simple variable or an array item of type string. A <string function> is a function returning a value of type string.

The effect of the + operator on strings is to concatenate them. A new string is produced which is the first string immediately followed by the second. If the resulting string would be longer than 255 characters, an Error 15 is generated.

2.11.3 Relational Expressions.

Relational expressions compare two numeric values or two string values. Thus :

<relational expression>
is: <numeric expression><relation operator><numeric expression>
or: <string expression> <relation operator><string expression>

Note that <relation operator>s have a lower precedence than any of the operators in either string or numeric expressions, and a higher precedence than any logical operator.

The <relation operator>s are :

<	Less than
<= =<	Less than or Equal
=	Equal
>= =>	Greater than or equal
>	Greater than
<>	Not equal

Where the arguments are numeric the types are made the same by the process of widening described in 2.9.1 above. The meaning of the relation is as might be expected.

Where the arguments are strings, the meaning of the relations requires some explanation:

Two strings are equal when they are the same length, and corresponding characters are the same.

One string is less than another if :

they are equal up to the end of the first string and the second string is longer

or: the first character which is not the same in the two strings is smaller in the first string. (Where the values representing the characters are being compared.)

BASIC does not support a Boolean type. Relational expressions yield an Integer value, −1 for True and 0 for False. Note that the `IF` and `WHILE` commands treat a value of 0 as False and any other value as True.

2.11.4 Logical Expressions.

BASIC does not support a Boolean type. Logical expressions perform bit-wise Boolean operations on Integers. Logical expressions are :

<argument>[<logical operator><argument>]

where:

<argument> is: `NOT` <argument>
or: <numeric expression>
or: <relational expression>
or: (<logical expression>)

The two arguments for a logical operator are forced to Integer, Error 6 results if an argument will not fit into the Integer range. The operation is then performed for each bit of the 16 bit two's complement representation of the Integer.

The **NOT** unary operator inverts each bit in the argument (0 becomes 1, and vice versa).

The dyadic operators, in order of precedence, and their effect on each bit are :

AND	Result is 0 unless both argument bits are 1
OR	Result is 1 unless both argument bits are 0
XOR	Result is 1 unless both argument bits are the same
EQV	Result is 0 unless both argument bits are the same
IMP	Result is 1 unless the right argument bit is 1 and the left 0

The result of a relational expression is either −1 or 0. The representation for −1 is all bits of the Integer =1; for 0 all bits of the Integer are 0. The result of a logical operation on two such arguments will yield either −1 or 0.

2.12 Functions.

Functions are subroutines which take a number of parameters, and which return a single value. Functions are invoked by their use as arguments in expressions, the value of the argument being the value returned by the function. There are three classes of function, built-in, user and external user.

2.12.1 Built-in Functions.

These are part of the BASIC language, and are described in Chapter 4. Unlike user functions the type of the result is not necessarily dictated by the type of the function name. For example the function **ABS** returns a result of the same type as its argument. Some built-in functions have optional parameters, and some allow different parameter types, neither of which is possible with user functions.

2.12.2 User Functions.

User functions are defined by the **DEF FN** command. This associates a name with a formal parameter list and an expression.

Function names take the form FN<name>[<type marker>], where the FN serves only to distinguish the name from variable names, and does not count toward the forty possible characters of the <name> part. As with variables, functions with the same name but different types are different functions.

The formal parameter list declares a number of variables local to the function. When the function is invoked actual parameters must be given which conform in number and type to the formal parameters - noting that BASIC will automatically change numeric types as required. The actual parameters are expressions, which are evaluated before the function is called, and whose values are copied to the formal parameters.

The body of the function is a single expression. It is not possible to have any form of control structure in a function. The expression may include references to other functions and to global variables as well as the formal parameters - though, of course, a formal parameter with the same name as a global variable makes the global variable's value inaccessable.

When a function is evaluated the result of the expression is converted to the same type as the function name, and the converted value returned.

2.12.3 External User Functions.

An external user function is a machine code subroutine whose address in memory has been declared in a **DEF USR** command. External user functions take one parameter of any type, and may return a value of any type. See Appendix C for a full description of support for externally developed subroutines.

2.13 Decimal Fractions and Binary Floating Point.

It is a sad fact that few decimal fractions have an exact representation as binary floating point numbers. Floating point numbers are rounded before they are printed so the inexact representation may be obscured.The effect may be seen when comparing values which are the result of separate sequences of arithmetic operations. The two values may appear equal when printed, but their binary representations are different. The same effect may be seen if two such numbers are subtracted and the result is not zero.

The effect may also be seen when Single Length numbers are converted to Double Length. For example if 0.1! is converted to Double Length the result is 0.1000000014901161#, which is close, but not exactly the same.

Unless decimal fractions are avoided altogether, by multiplying all numbers by suitable powers of ten (treating all money values as pence, for instance) it is impossible to avoid these effects. They can be reduced by rounding explicitly using the built in function **ROUND**, which will round numbers which print the same to the exactly the same value. Alternatively the slight differences may be retained, and account taken of their existence when numbers are compared or subtracted.

3 Overview of Commands

This chapter is not intended to describe the commands in detail, that is done in chapter 4. Here the commands are described in groups, in order to give some idea of their purpose and to place them in context.

3.1 Loading and Running of Programs.

`CLEAR, CHAIN, CHAIN MERGE, COMMON, LOAD, MEMORY, RUN`

`CLEAR` clears away all variables, closes all files and generally resets BASIC, except that the current program is retained. The memory available to BASIC, the size of stack to be used, the maximum number of files and the maximum random record size may be changed at this point.

`MEMORY` allows the memory available to BASIC, the size of stack to be used, the maximum number of files and the maximum random record size to be changed.

`CHAIN` and `CHAIN MERGE` enable one program to load and enter another. With `CHAIN MERGE` all or part of the current program may be retained. Variables may be retained past `CHAIN` or `CHAIN MERGE` - all may be retained, or only those specified in `COMMON` statements.

`LOAD` discards the current program and variables and resets all other status. A new program is then loaded, and may be entered at once.

The simplest `RUN` command starts executing the current program. Other forms of the command cause execution to start at a given line number or are similar to `LOAD`.

3.2 Program Termination.

`END`

Programs stop after the last line has been executed, and BASIC returns to the system level. END may be used to terminate the program at some other point, or points.

3.6 Line Printer Output.

`LPOS, LPRINT, WIDTH LPRINT`

These commands are equivalent to the console output ones.

3.7 Files.

BASIC supports the reading and writing of disc files, with sequential or random access. In order to access a particular file it must be opened, at which time the access method is declared, and the disc file is associated with a ‹file number›. Thereafter the file is referenced using the ‹file number›. A given ‹file number› may refer to only one file at a time, but may be re-used once the previous file has been closed.

Unless otherwise set when BASIC is initialised (see Appendix A) or in `CLEAR` or `MEMORY` commands, ‹file number›s are in the range 1..3. That is to say BASIC can handle up to three files at once.

3.7.1 Directory Access.

`DIR, ERA, FILES, KILL, NAME, REN`

The `DIR` and `FILES` commands generate directory listings to the console.

The `ERA` and `KILL` commands delete files from disc.

The `NAME` and `REN` commands rename files on disc.

3.7.2 In General.

`CLOSE, OPEN, RESET`

The `OPEN` command specifies the name of the file, the access method required, and the file number which is to be used. The file name is given as a string, whose format must conform to the underlying operating system's rules.

`CLOSE` is the inverse operation to `OPEN`. If the file is open for Sequential Output any data still in buffers is written away. The disc file is closed and the file number released for other use.

The RESET command closes all files and resets the file system. A RESET command is usually required before discs are changed.

3.7.3 Input Access - Sequential.

EOF, INPUT, LINE INPUT

Input Access allows ordinary text files to be read. INPUT and LINE INPUT whose first argument is #<file number> are similar to the console input commands, except that they read lines from file.

It is an error to attempt to read past the end of an input file. The EOF function may be used to detect the end of file condition, and avoid reading any further.

3.7.4 Output Access - Sequential.

PRINT, WRITE

Output Access allows ordinary text files to be written. PRINT commands whose first argument is #<file number> are similar to the console output commands, except that data is written to file. Note that files are assumed to be infinitely wide.

WRITE commands whose first argument is #<file number> are particularly useful for generating files to be read at some later date using INPUT, since INPUT expects items to be separated by commas and the double quotes around strings avoids any ambiguity.

3.7.5 Random Access.

CVD, CVI, CVS, FIELD, GET, LSET, MID$, MKD$, MKI$, MKS$, PUT, RSET

Random Access files are viewed as a number of fixed size records. The maximum record size may be set to almost any value when the BASIC interpreter is initialised (see Appendix A), but defaults to 128 bytes. Each record is referenced by ‹record number›, which is in the range 1....32767.

3.3 Control Structures.

FOR, GOSUB, GOTO, IF, ON x GOSUB, ON x GOTO, RETURN, WHILE

BASIC supports two forms of loop: FOR and WHILE. A FOR loop has a control variable associated with it, which steps through a range of values - one per iteration - until a given end value is reached. The size of each step may be specified, including negative steps. The BASIC FOR loop is skipped if the initial value of the control variable exceeds the end value. NEXT marks the end of a FOR loop. A WHILE loop repeats until the given condition becomes false. The BASIC WHILE loop is skipped if the condition is false the first time. WEND marks the end of a WHILE loop. WHILE loops may be nested.

The IF command may be used to choose alternative or optional actions. IF is followed by an expression. If the expression yields a non zero value the THEN part is chosen, otherwise the ELSE part is chosen, if it is present. Both the THEN and ELSE parts may contain more than one command, but they must all be on the same line with the IF command. IF commands may be nested.

Simple subroutines are supported. GOSUB calls the subroutine that starts at the line given. RETURN terminates the subroutine, and returns control to the next command after the matching GOSUB. All variables in BASIC are global. Subroutines may recurse.

GOTO causes an unconditional branch to the line given.

ON x GOSUB and ON x GOTO commands evaluate the expression x, and use the result to choose one out of a list of lines to call or jump to.

3.4 Variables.

DEFINT, DEFSNG, DEFDBL, DEFSTR, DIM, ERASE, OPTION BASE

Where a variable or function name is given without an explicit type marker BASIC must assume a type. The type assumed depends on the first character of the name (which is an alphabetic character). DEFINT, DEFSNG, DEFDBL and DEFSTR commands may be used to define the default type for each letter. Note that these default settings may be changed at will, but that the potential for confusion is enormous.

Unless otherwise explicitly declared in a DIM command the maximum for each subscript of an array is set to 10 by the first use of the array (implicit declaration). The minimum for each subscript of an array may be set by an OPTION BASE command to 0 or 1, but defaults to 0 otherwise. It is an error to attempt to change this base value once it has been set, or once an array has been declared (explicitly or implicitly).

Arrays may occupy large areas of memory. The ERASE command may be used to reclaim the memory used by arrays which are no longer in use.

3.5 Console I/O.

INPUT, LINE INPUT, POS, PRINT, WIDTH, WRITE

INPUT fetches a line from the console, and may interpret parts of it as numeric input and parts as plain text, depending on the parameters of the INPUT command.

LINE INPUT fetches a line from the console without interpretation (but with line imaging) and assigns it to a string variable.

All output to the console interacts with the console width, as set by a WIDTH command (the default width is 80). BASIC will insert carriage returns in order to keep the console position within the console width. The current position across the console is returned by the POS function. If the console is itself inserting carriage returns after a given column position, WIDTH may be used to inform BASIC, so that unnecessary carriage returns can be avoided.

PRINT outputs numbers and strings to the console. Numbers are output 'free format'. PRINT divides the console into 'print zones' which impose some order on the output. The ZONE command may be used to change the size of print zones. The keyword USING introduces a format template, which controls the output of all further numbers and strings, without reference to print zones.

WRITE outputs numbers and strings to the console, inserting commas between each item and enclosing strings in double quotes. Numbers are output in 'free format'.

BASIC maintains a buffer for each file open for random access. The GET command reads a given record from disc into the random record buffer. The PUT command writes the current contents of the random record buffer to the given record on disc.

Data is read from, and written to, the random record buffer with the aid of string variables defined by FIELD commands. A FIELD command declares a number of field string variables, each associated with a given part of the random record buffer. Data may be read straightforwardly using the string variables. Data to be written to the buffer may not be simply assigned to the string variables, one of the special assignments LSET, MID$ or RSET must be used.

Each FIELD command defines a template for the data in the random record buffer. Any number of such templates may be defined for a given file, allowing different record organisations at different times.

In order to read and write numeric data a mechanism is provided to convert the numeric data types to strings, and back again. MKD$, MKS$ and MKI$ produce strings from Double Length, Single Length and Integer data respectively. CVD, CVS and CVI reconvert such strings. (Note that the strings produced are, in fact, copies of the internal form of the numbers, so numbers written this way are read back exactly as they were.)

3.8 Constant Data.

DATA, READ, RESTORE

All the DATA commands in a program, taken in line number order, make up a sort of file. This 'file' may be read using READ commands. The position within this 'file' may be reset by RESTORE commands.

3.9 Machine Level Operations.

CALL, DEF USR, INP, OUT, PEEK, POKE, USR, WAIT

Two forms of external subroutine are supported. CALL invokes a machine code routine at a given address, passing a number of parameters. USR invokes a machine code function at an address previously defined by a DEF USR command. USR passes one parameter, and returns one value as the value of the function. See Appendix C for a more detailed description.

PEEK and POKE read and write bytes of machine memory.

INP, OUT and WAIT give access to machine I/O space.

3.10 Error Trapping.

ERL, ERR, ERROR, ON ERROR GOTO, RESUME

When BASIC detects an error its default action is to issue an error message and return to Direct Mode. This action may be overridden by an ON ERROR GOTO command, which sets the line number of a routine to process errors. When an error occurs ERL is set to the current line number, and ERR is set to the error number. The error processing routine may inspect these variables to decide whether it is capable of recovering or not. The RESUME command may be used to return to normal running.

3.11 Creation of Programs.

SAVE

SAVE allows the current program to be written to a file. The program may be written in one of three forms:

ASCII a text file

Compressed a copy of the internal form of the program, requires less disc space and loads faster than ASCII files.

Protected similar to Compressed, except that the program is encrypted.

When a Protected program is loaded all commands which give the user access to the program text are inhibited. Once a program has been protected it cannot be unprotected.

4 The Commands and Built In Functions

This chapter contains a detailed description of all Commands and Built In Functions, and is arranged in alphabetical order of keyword.

The following terms are used throughout the chapter:

<address expression>

A <numeric expression>, which when rounded to an integer yields a value in the range –32768....65535. Negative values are then converted to their unsigned equivalent (by adding 65536).

<array variable name>

The name of an array, which may include its <type marker>.

<expression>

An expression yielding a value of any type.

<file name expression>

A <string expression> whose value is a valid file name.

<file number expression>

An <integer expression> which yields a value in the range 1....<files>, where <files> is the maximum file number set when BASIC was initialised - or by the latest **CLEAR** or **MEMORY** command (see Appendix A).

<integer expression>

A <numeric expression>, which when rounded to an integer (rounding away from zero) yields a value in the range –32768....32767.

<line number>

A decimal number in the range 0....65534.

<line number range>

A <line number range> specifies all lines whose numbers are within the inclusive range given. The range may take one of the forms:

<line number>

Only the given number is in the range.

<line number>-

Lines from the given line to the end of the program.

-<line number>

Lines from the beginning of the program to the given line.

<logical expression>

BASIC does not support a separate Boolean type. Integer value 0 is treated as false, and any other Integer value as True. A <logical expression> is, therefore, an <integer expression> by another name.

<numeric constant>

See Section 2.4.

<numeric expression>

An expression which returns a value of any of the numeric types, Integer, Single Length and Double Length.

Note that this includes expressions previously categorised as relational and logical.

<numeric variable>

A reference to a variable, which may be an array item, whose type is one of the numeric types.

<quoted string>

A <quoted string> is between 0 and 255 characters enclosed in double quotes, or starting with double quotes and terminated by carriage return.

<simple variable>

A reference to a variable excluding array variables. The reference may include a <type marker>.

<string expression>

An expression which yields a value of type String.

<string variable>

A reference to a variable, which may be an array item, whose type is String.

<type marker>

One of the characters :

%	indicating Integer
!	indicating Single Length
#	indicating Double Length
$	indicating String

<variable>

A reference to any form of variable. This includes variables which are items in an array, in which case suitable subscripts must be included.

<variable name>

The name of a <simple variable>, which may include its <type marker>.

ABS

Absolute value. Function

Use
To determine the absolute value of a given expression.

Form

`ABS(<numeric expression>)`

Notes.
`ABS` of a negative value returns that value negated. `ABS` of a positive value returns the value unchanged.

Associated Keywords.
`SGN`

ASC

Get ASCII value of character. Function

Use
To get the numeric value of the first character of a string, given that the ASCII encoding of characters is used.

The inverse of `CHR$`.

Form

`ASC(`<string expression>`)`

Where the <string expression> yields a string at least one character long.

Associated Keywords.
`CHR$`

ATN

Arc-tangent. Function

Use
To calculate the arc-tangent of a given value. The result returned is in radians, in the range −PI/2....+PI/2

Form

`ATN(`<numeric expression>`)`

Notes.
While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `ATN` returns a Single Length result.

Associated Keywords.
`SIN, COS, TAN`

CALL

Call external subroutine. Command

Use
Allows externally developed subroutines to be invoked from BASIC.

Form

`CALL` <simple variable>[`(`<argument list>`)`]

The <simple variable>'s value gives the address of the subroutine. The <simple variable> must have been set to the result of an <address expression>.

The <argument list> is a <list of: <variable>. The addresses of the variables in the argument list are passed to the subroutine.

Notes.
See Appendix C for a detailed discussion of the calling sequence, variable formats etc.

Associated Keywords.
`DEF USR, USR, UNT`

returns to System level.

CDBL

Convert to Double Length. Function

Use
To take a given expression and return its value as a Double Length number.

Form

CDBL(<numeric expression>)

Notes.
Many decimal fractions do not have exact binary floating point representations. Converting a single length floating point number to a double length one will highlight the effects of this. For instance:

```
100 DATA 0.1
110 READ A!
120 PRINT A!,CDBL(A!)
```

produces:

```
 .1  .100000001490116
```

Associated Keywords.
CSNG, CINT, CVD, MKD$

CHAIN
CHAIN MERGE

Chain into new program. Command

Use

Allows one program to load and enter another, optionally retaining some or all of the current program and variables.

Form

```
CHAIN<string expression>[,<line number expression>][,ALL]]
CHAIN MERGE <string expression>
                    [,[<linenumber expression>]
                      [,ALL][,DELETE<line number range>]]
```

Where:

The <string expression> gives the name of the file containing the required program. If no type extension is given then '.BAS' is assumed.

The <line number expression> is a <numeric expression>, which when rounded to an integer (rounding to nearest) yields a value in the range −32768....65534. Negative values are then converted to their unsigned equivalent (by adding 65536), and must then be in the range 0....65535. This defines where to start executing from once the new program is loaded:

0 or absent	:	starts executing at the lowest numbered line.
1....65534	:	starts executing at the given line. If the line does not exist an Error 8 results.
65535	:	returns to System level.

ALL causes all variables in the current program to be retained. If ALL is omitted then no variables are retained, unless otherwise specified by COMMON command(s).

DELETE causes the lines in the <line number range> given to be removed from the current program before the new program is loaded. (DELETE may be specified with the CHAIN command, but is wholly redundant.)

Notes.
The differences between CHAIN and CHAIN MERGE are as follows:

CHAIN

Deletes all current progam lines.

Resets all DEFINT, DEFSNG, DEFDBL and DEFSTR settings. (So any common variables should have explicit types, or the new program must re-issue suitable DEF commands.)

Resets OPTION BASE setting, unless any arrays are passed to the new program.

CHAIN MERGE

Deletes only those lines specified in the DELETE clause (if any).

Retains all DEFINT, DEFSNG, DEFDBL and DEFSTR settings.

Retains OPTION BASE setting.

Actions common to CHAIN and CHAIN MERGE are :

- All User Functions are forgotten.
- ON ERROR GOTO is turned off.
- All open files are retained.
- RESTORE action is taken.
- All active FOR, WHILE and GOSUB commands are forgotten.

During a CHAIN MERGE operation a line in the new program with the same number as an existing line will replace the existing line.

If a protected program is CHAIN MERGEd with an unprotected one, the result is a protected program.

Associated Keywords.
COMMON, LOAD

CHR$

Convert to character. Function

Use
To convert a numeric value to its character equivalent, given that the ASCII coding of characters is used.

The inverse of ASC.

Form

`CHR$(`<integer expression>`)`

Where <integer expression> must yield a value in the range 0....255.

Associated Keywords.
`ASC`

CINT

Convert to integer. Function

Use

To convert the given value to integer representation, if possible.

Form

`CINT (`<numeric expression>`)`

The <numeric expression> is rounded to integer, which must then be in the range −32768....32767.

Associated Keywords.

`CSNG, CDBL, MKI$, CVI, INT, FIX, ROUND, UNT`

CLEAR

Clear all variables and files. Command

Use

To set all variables to zero if numeric, or null if string. Erase all arrays. Forget all User Functions. Close all files.

May also set stack size and total memory available to BASIC as well as the maximum number of files and the maximum random record size.

Form

```
CLEAR [,[<address expression>][,[<address expression>]
                [,[<address expression>][,<address expression>]]]]
```

The first <address expression> gives the address of the highest byte in memory which may be used by BASIC. If omitted the current setting is retained.

The second <address expression> gives the number of bytes that BASIC should use for stack. If omitted the current setting is retained. If present, the expression must yield a value of at least 256.

The third <address expression> gives the maximum number of files that may be handled at once. If omitted the current setting is retained.

The fourth <address expression> specifies the maximum random record size. If omitted the current setting is retained.

Notes.

The optional parameters allow the maximum number of files, the maximum random record size and total memory size to be set in much the same way as in the initial command line (see Appendix A).

`CLEAR` causes BASIC to forget any active `FOR`, `WHILE` or `GOSUB` commands.

Associated Keywords.
`MEMORY`

CLOSE

Close one, or more, files. Command

Use
To finish using files.

Form

CLOSE [<list of: [#]<file number expression>]

If the file number list is omitted all files are CLOSEd.

Each <file number expression> gives the number of a file to be closed.

Notes.
When a sequential output file is closed all outstanding data is written to the file.

After a CLOSE the file numbers involved no longer have any file associated with them, and they may be re-used.

CLOSEing a file number which is not in use has no effect, and is ignored.

Associated Keywords.
RESET

COMMON

Declare variables to be retained during CHAIN or CHAIN MERGE. Command

Use
`COMMON` statements allow variables to be retained selectively past `CHAIN` and `CHAIN MERGE` operations.

Form

`COMMON` <list of: <variable reference>

Where <variable reference> is : <variable name>
or : <array variable name>`()`

Note.
When a `CHAIN` or `CHAIN MERGE` command is executed all `COMMON` statements in the program are found and obeyed. (This is done before any program lines are deleted !) At all other times `COMMON` statements are ignored.

Associated Keywords.
`CHAIN, CHAIN MERGE`

COS

Cosine. Function

Use
To calculate the Cosine of a given value, given that the argument is expressed in radians.

Form

`COS(`<numeric expression>`)`

The <numeric expression> gives the angle in radians, and must yield a value in the approximate range −200,000....200,000.

Notes.
With angles very much greater than 2*PI the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI....+PI. Rather than return a very inaccurate figure, BASIC will not evaluate `COS` for values much beyond the range given above.

While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `COS` returns a Single Length result.

Associated Keywords.
`SIN, TAN, ATN`

CSNG

Convert value to Single Length. Function

Use
To convert the value of the given expression to Single Length representation.

Form

`CSNG(`<numeric expression>`)`

Associated Keywords.
`CINT, CDBL, MKS$, CVS`

CVD

Convert string to double length numeric. — Function

Use
To reconvert a string produced by `MKD$`, to recover the original Double Length value.

Form

`CVD(`<string expression>`)`

Where the <string expression> must yield a string eight bytes long.

Notes.
`MKD$` creates an eight byte string, which is an exact representation of a Double Length value, requiring no binary to decimal conversion. The string may be used, for instance, to put Double Length values into a random record.

Associated Keywords.
`MKD$, CVI, CVS`

CVI

Convert string to integer. Function

Use
To reconvert a string produced by `MKI$`, to recover the original integer value.

Form

`CVI (`<string expression>`)`

Where the <string expression> must yield a string two bytes long.

Notes.
`MKI$` creates a two byte string, which is a representation for an Integer value requiring no binary to decimal conversion. The string may be used, for instance, to put Integer values into a random record.

Associated Keywords.
`MKI$, CVS, CVD`

CVS

Convert string to single length number. Function

Use
To reconvert a string produced by `MKS$`, to recover the original single length value.

Form

`CVS (`<string expression>`)`

Where the <string expression> must yield a string four bytes long.

Notes.
`MKS$` creates an four byte string, which is an exact representation of a Single Length value, requiring no binary to decimal conversion. The string may be used, for instance, to put Single Length values into a random record.

Associated Keywords.
`MKS$, CVI, CVD`

DATA

Declare constant data. Command

Form

DATA <list of: <constant>

<constant> may be any form of <numeric constant>, a <quoted string> or an <unquoted string>.

a <numeric constant> may be surrounded by <white space>, which is ignored.

a <quoted string> may be surrounded by <white space> which is ignored. The trailing double quotes may be omitted if the string is the last object on the line.

an <unquoted string> is between 0 and 255 characters which are taken to be a string. A comma, colon or carriage return terminates <unquoted string>s in this context. An <unquoted string> may be surrounded by <white space>, which is ignored.

Notes.
All the DATA commands in line number order within a program form a list of constants which may be read into variables by the READ command. As each constant is read a pointer is advanced to the next one. The RESTORE command moves the pointer to a specified position in this list.

DATA commands are not executed. The constants are not checked for validity until they are READ, at which point numeric constants may only be read into numeric variables, and strings (of either type) may only be read into string ;variables.

Associated Keywords.
READ, RESTORE

DEC$

Generate formatted string representation of number. — Function

Use
To create a string representation of the value of a given expression, according to a given format template.

Form

`DEC$(`<numeric expression>`,`<string expression>`)`

The value of the <numeric expression> is converted to a decimal string given the format template in the <string expression>, which is interpreted as in `PRINT USING`.

Notes.
The format template is not a full `PRINT USING` string, and may only contain the following characters:

`+ - $ * # , . ^`

Associated Keywords.
`STR$, HEX$, OCT$, PRINT USING`

DEF FN

Define User Function. Command

Use
BASIC allows the program to define and use simple value returning functions. DEF FN is the definition part of this mechanism.

Form

DEF FN<name>[(<formal parameters>)]
=<general expression>

The <name> takes the same form as a <variable name> (including an optional <type marker>), the name of the function being FN<name>. The value returned by the function is of the same type as <name>.

The <formal parameters> take the form <list of: <simple variable name>.

The <general expression> may involve not only the formal parameters, but also other variables or functions. The result of the expression must be compatible with the type of the function.

Notes.
A user function is invoked when it is used as an argument in an expression. The invocation takes the form:

FN<name>[(<actual parameters>)]

The <actual parameters> take the form <list of: <general expression>. There must be the same number of actual parameters as formal parameters. The type of each actual parameter must be compatible with the type of the corresponding formal parameter.

The formal parameters are local to the function. Variables with the same names as formal parameters are not affected by invoking the function, nor are they accessible from within the function.

If the type of the function is not explicit, then the default when the DEF FN is executed applies. If the type of a formal parameter is not explicit, then the default when the function is invoked applies.

DEF SEG

Command

Use

This command is used by 16 bit versions of BASIC. In the 8 bit versions it is ignored.

DEF USR

Define external function. Command

Use

Provides an alternate mechanism to CALL for invoking external function subroutines. This mechanism allows for only one parameter, but allows the external function to return an anonymous value.

Form

DEF USR[<digit>]=<address expression>

Ten separate functions may be defined, USR0 to USR9. If the digit is omitted, '0' is assumed.

The <address expression> gives the address of the external function.

Notes.

An external function is invoked when it is used as an argument in an expression, taking the form :

USR[<digit>](<general expression>)

Where the <digit> may be one of 'Ø'....'9', if omitted then 'Ø' is assumed. There is only one parameter, which is mandatory.

USR[<digit>]s may be redefined at will.

See Appendix C for details of external subroutines in general, and restrictions and caveats on external functions in particular.

Associated Keywords.

CALL, VARPTR

DEFINT DEFSNG DEFDBL DEFSTR

Set default type for names. Command

Use
All variable and function names have a type associated with them. This type may be explicit in the name, or implicit, depending on the defaults set by these commands.

Form

`DEFINT` <letter range list>
`DEFSNG` <letter range list>
`DEFDBL` <letter range list>
`DEFSTR` <letter range list>

Where <letter range list> is: <letter range>[,<letter range list>]
and <letter range> is : <letter>
or : <letter>-<letter>
where the form <letter>-<letter> defines an inclusive range of letters.

Notes.
When a variable name without an explicit <type marker> is met the current default type is assumed. The default type depends on the first character of the name. The default is set for each letter by the latest of these commands in whose <letter range list> the letter was included.

`DEFINT` sets the default to Integer.
`DEFSNG` sets the default to Single Length.
`DEFDBL` sets the default to Double Length.
`DEFSTR` sets the default to String.

The initial state of the default type for all letters is Single Length. The state is reset to this by :

`LOAD`, `RUN`, `CHAIN`, `CLEAR`

DIM

Declare array dimensions. Command

Use

In order to allocate memory for an array BASIC must know how many dimensions it has, and the maximum permissible value of each one. The `DIM` command allows dimensions to be set explicitly.

Form

`DIM` <list of: <subscripted variable>

Where <subscripted variable> is : <variable name>`(`<dimension list>`)`
and <dimension list> is: <list of: <integer expression>

Notes.

Each <integer expression> in the <dimension list> sets the maximum value to be allowed for the corresponding subscript. The minimum value of each subscript is 0 or 1 as set in an `OPTION BASE` command, or 0 by default.

Array dimensions may be set explicitly in a `DIM` command, or implicitly when the array is first encountered. When an array is declared implicitly the maximum value for each subscript is set to 10.

It is an error to attempt to change the dimensions of an array which has already been declared (explicitly or implicitly).

There is no restriction on the number of subscripts an array may have, other than line length and store size !

Associated Keywords.

`OPTION BASE`

DELETE

Delete lines of program. Command

Use
To remove part of the current program.

Form

DELETE <line number range>[,<line number>]

Notes.
Removes the lines in the given range, and has the following side effects:

-All User Functions are forgotten.
-ON ERROR GOTO is turned off.
-RESTORE action is taken.
-All active FOR, WHILE and GOSUB commands are forgotten.

If the <line number> parameter is omitted BASIC returns to System level after executing a DELETE, otherwise BASIC starts executing at the given line -if the <line number> given is zero, BASIC starts execution at the lowest numbered line.

Associated Keywords.
CHAIN, CHAIN MERGE.

DIR

Directory Listing Command

Use
To print a directory listing to the console. This may be a complete listing, or a partial listing depending on the given file names, which may contain 'wild card' characters.

Form

`DIR` [<list of: <filename>]

Notes.
If the list of filenames is omitted, then all files on the default drive are listed. If a number of filenames is given, then each name is taken in turn and all files consistent with the name are listed. The 'wild card' characters '?' and '*' may be used in the usual way.

`DIR` treats all the rest of the current line as its argument, irrespective of colons or single quotes.

Associated Keywords.
`FILES, FIND$`

DISPLAY

Display contents of file on console. Command

Use
To print a given file on the console.

Form

`DISPLAY` <string expression>

The <string expression> yields a string giving the name of the file to be printed on the console. The name may not contain 'wild card' characters.

Notes.
The file specified is read and written directly to the console.

Tabs and carriage returns are treated in the usual way, and console width is enforced. All other characters are sent to the console exactly as read from the file, so control characters and characters outside the normal ASCII range may be included in the file to produce special console effects.

Associated Keywords.
`TYPE`

END

End of program. Command

Use
To signal the end of a program, and stop cleanly.

Form

```
END
```

Notes.
Any number of `END` commands may appear anywhere in a program, though an `END` is assumed after BASIC has obeyed the last line of a program.

`END` closes all files and returns to System level.

Associated Keywords.
`STOP, SYSTEM`

EOF

End of File test. Function

Use
To test if the given file is positioned at end of file. Returns −1 (true) if is at end of file. Returns 0 (false) otherwise.

Form

EOF(<file number expression>)

Notes.
The file may not be open for Output.

ERA

Erase files. Command

Use
To erase a file or list of files.

Form

ERA <list of: <filename>

Notes.
Each file specified is erased. If a filename contains 'wild card' characters, then all matching files are erased.

ERA takes the rest of the current line as its argument, irrespective of colon or single quotes.

Associated Keywords.
KILL

ERASE

Erase arrays. Command

Use
When an array is no longer required ERASE may be used to remove it and reclaim the memory it occupies ready for other use.

Form

ERASE <list of: <variable name>

Where each <variable name> is the name of an array, without any subscript part.

Notes.
It is an error to ERASE an array which has not been declared (explicitly or implicitly).

Once an array has been ERASEd it is possible to declare it again with new dimensions.

Associated Keywords.
DIM

ERR
ERL

Error and Error Line Numbers. Variables

Use
These variables may be used in error handling subroutines to discover the error number (`ERR`) and the line being executed when the error occurred (`ERL`).

Notes.
These variables may be used anywhere, except that they may not be assigned to.

Associated Keywords.
`ON ERROR, ERROR`

ERROR

Cause Error action to be taken. Command

Use
Invoke Error Action with a given error number. The error number may be one already used and recognised by BASIC, in which case the action taken is the same as would be taken if such an error had been detected by BASIC. Error numbers beyond those recognised by BASIC may be used by the program to signal its own errors.

Form

`ERROR` <integer expression>

Where the <integer expression> must evaluate to a number in the range 1....255.

Notes.
In the event of BASIC attempting to print an error message for an error number that it does not recognise, the message '`Unknown error`' is produced.

Associated Keywords.
`ON ERROR, ERR, ERL`

EXP

Exponential. Function

Use
Calculate e to the given power. Where e is the number whose natural logarithm is 1 (approx. 2.7182818).

Form

`EXP(`<numeric expression>`)`

Notes.
`EXP` of numbers much greater than 88 will cause an overflow error.

`EXP` of numbers much less than −88.7 underflows, and yields the value zero.

While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `EXP` returns a Single Length result.

Associated Keywords.
`LOG`

FIELD

Define a field layout for a given file. Command

Use
A FIELD command defines a template for records in a file open for random access, and associates a string variable with each field.

Form

FIELD [#]<file number expression>,<list of: <field>

Where:
<file number expression> gives the file number to which this template applies. The file must be open for random access.

<field> is : <field size> AS <string variable>

where <field size> is an <integer expression>, value in the range 0....255. The sum of all the <field size>s must be less than, or equal to the record size.

Notes.
Each field in the template is given a size and a string variable is associated with it. The fields are mapped onto the record, with the first field corresponding to the first few bytes, the second the following bytes, and so on.

The string variables are effectively pointers into the record buffer. Executing a FIELD command neither reads nor writes data, but simply sets the string variables to point at the current contents of the buffer.

Each FIELD statement executed for a given file sets up a separate template for the record buffer. There is no restriction on the number of such templates, and they will all be valid.

Warning.
The string variable associated with each field must not be assigned to except by using LSET, RSET or MID$. Any other form of assignment will not write to the record buffer, but will create a new string which will not be stored in the random record buffer and break the association of the string variable name with the record buffer.

Associated Keywords.
OPEN, GET, PUT, LSET, RSET, MID$

FILES

Directory listing. Command

Use
To print a directory listing to the console. This may be a complete listing, or a partial listing depending on a given file name, which may contain 'wild card' characters.

Form

`FILES` [<string expression>]

Where the <string expression> specifies which files are to be listed. If the expression is omitted, then all files on the default drive are listed.

Notes.
The 'wild card' characters '?' and '*' may be used in the string expression to specify a number of files in the usual way.

Associated Keywords.
`DIR, FIND$`

FIND$

Look for given file. Command

Use
`FIND$` looks up the given file and returns its name, if found.

Form

`FIND$(`<string expression>[`,`<integer expression>]`)`

The <string expression> gives the name of the file to be searched for. The resulting string may contain 'wild card' characters, which are treated in the usual way.

The <integer expression>, if present, must yield a value in the range 1....255.

Notes.
If the file sought is found `FIND$` returns a twelve character string containing the name of the file, complete with any file attribute bits. (Note that the `STRIP$` function will remove these attribute bits.)

If no matching file is found, then a null string is returned.

The second argument is useful when the <string expression> contains 'wild card' characters. If the value of the <integer expression> is n, then `FIND$` looks for the nth matching file. If the second argument is omitted then `FIND$` looks for the 1st matching file.

FIX

Fix number to integer. Function

Use
To truncate given value to an integer. Note that this does not convert floating point numbers to integer format, but simply removes any fraction part, rounding the value toward zero.

Form

FIX(<numeric expression>)

Associated Keywords.
CINT, INT, ROUND

FOR

FOR loop. Command

Use
Execute a body of program a given number of times, stepping a control variable between a start and an end value.

Form

FOR <simple variable><start> TO <end> [STEP <step size>]

<simple variable> is the control variable for the FOR loop, it must be an Integer or Single Length, and may not be an array item.

<start> is a <numeric expression> giving the initial value for the control variable.

<end> is a <numeric expression> giving the limiting value for the control variable.

<step size> is a <numeric expression> giving the value to add to the control variable after each iteration. If the step size is omitted, a step of 1 is assumed. Note that the <step size> may be negative.

The results of the <start>, <end> and <step size> expressions are forced to the same type as the control variable.

Notes.
The FOR command initiates a FOR loop. The end of the FOR loop is marked by a NEXT command.

When the FOR command is executed the <start>, <end> and <step size> expressions are evaluated, then the control variable is set to the <start> value. If the initial value is beyond (see below) the end value then the FOR loop is skipped, otherwise the loop is entered. When the matching NEXT is found the step size is added to the control variable, and control returns to the start of the FOR loop unless the control variable is now beyond the end value.

If the step size is positive (or omitted) the FOR loop terminates when the control variable is greater than the end value. If the step size is negative the FOR loop terminates when the control variable is less than the end value.

The NEXT command which matches a given FOR is established statically when the FOR is first executed. That is to say that the NEXT which matches the FOR depends on the order of statements in the program, quite independently of the order of execution. It is not possible, therefore, to have more than one NEXT associated with a given FOR.

FOR loops may be nested.

The value of the control variable is defined once the loop has terminated. It is permissable to terminate a FOR loop by avoiding the NEXT, the value of the control variable is unchanged.

Care must be exercised if fractional step sizes are used, since the effects of rounding may mean that the control variable value can never exactly equal the final value.

Associated Keywords.
NEXT

FRE

Free space measurement. Function

Use
To establish how much free memory remains unused by BASIC. Two forms of the function exist, one causes BASIC to 'garbage collect' before measuring the free space.

Form

`FRE(`<numeric expression>`)` or
`FRE(`<string expression>`)`

Notes.
The value of the argument in a `FRE` is not relevant, so `FRE(Ø)` and `FRE("")` are the recommended forms.

BASIC maintains a heap in the free memory, in which strings are stored. As the heap grows the free space is used up. At the same time areas within the heap area may become free, but cannot be reclaimed by BASIC until a 'garbage collection' is performed. In general BASIC does not 'garbage collect' until the heap has grown to fill all free memory.

`FRE(Ø)` returns the amount of memory left for the heap to grow into.

`FRE("")` forces BASIC to 'garbage collect' so that any unused space within the heap is returned to the free area. The result of the expression is then the maximum free space available. Note that the 'garbage collection' may take a noticeable time.

GET

Get record from random file. Command

Use
To read a given record from a random file into the record buffer, where it can be processed by the program.

Form

`GET` [#]<file number expression>[,<integer expression>]

The <file number expression> specifies the file to read from, which must be open for random access.

The <integer expression> specifies the number of the record to be read, and must yield a value in the range 1....32767. If the expression and the comma are omitted, then the record following the one read by the last `GET` or written by the last `PUT` is assumed - if this is the first `GET` or `PUT` then record 1 is read.

Associated Keywords.
`OPEN`, `FIELD`, `INPUT`, `LINE INPUT`

GOSUB

Go to Subroutine. Command

Use
To call a given subroutine.

Form

GOSUB <line number>

Notes.
When a GOSUB is executed the program jumps to the given line number, remembering the position immediately after the GOSUB so that execution may continue there when the subroutine returns.

Subroutines are terminated by a RETURN command. A subroutine may contain more than one RETURN command.

Subroutines may be nested and recurse, the depth being limited only by the stack size.

Associated Keywords.
RETURN

GOTO

Go to line. Command

Use
Unconditional jump to a given line.

Form

`GOTO` <line number>

HEX$

Hexadecimal string. Function

Use
To produce a string of hexadecimal digits representing the value of the given expression.

Form

HEX$(<unsigned integer expression>[,<integer expression>])

The <unsigned integer expression> must yield a value in -32768....65535 This value is treated as an unsigned 16 bit value, to be converted into a string of hexadecimal characters.

The optional <integer expression> gives the minimum size of string to be produced, and must yield a value in the range 0....16.

Notes.
HEX$ always generates as many characters as are required to represent the number (zero generates at least one digit). If the optional <integer expression> is present and the string is too short, then the string is filled to length with leading zeros. The string is not truncated if it is too long.

Associated Keywords.
OCT$, DEC$, STR$

HIMEM

Return the address of highest byte in memory used by BASIC. — Function

Use
The amount of memory used by BASIC may be altered by the `MEMORY` and `CLEAR` commands. The built-in `HIMEM` function may be used to establish the current memory use.

Form

```
HIMEM
```

Notes.
`HIMEM` returns a Single Length value giving the address of the highest byte in memory used by BASIC.

Associated Keywords.
`CLEAR, MEMORY`

IF

Command

Use
To conditionally execute statements.

Form

IF <logical expression> THEN <option part>
[ELSE <option part>]

or

IF <logical expression> GOTO <line number>
[ELSE <option part>]

Where <option part> is: <statement(s)>
or: <line number>

and <statement(s)> is one or more statements on the same line as the IF (separated by colons).

Notes.
The <logical expression> is evaluated and if the result is zero then the THEN or GOTO option is executed. Otherwise the THEN or GOTO option is skipped, until either end of line or a matching ELSE is found. In the latter case the ELSE option is executed.

IF statements may be nested to any depth, limited only by the line length.

ELSE parts are associated with the innermost IF which does not already have an ELSE part.

THEN <line number> and ELSE <line number> are equivalent to THEN GOTO <line number> and ELSE GOTO <line number>, respectively.

Associated Keywords.
WHILE

INKEY$

Input key from keyboard. Function

Use
Input next key, if any, from keyboard (console input). Note that any character input is not reflected to the console output.

Form

```
INKEY$
```

Notes.
If there is a character pending then `INKEY$` returns it (as a one character string). If no characters are pending `INKEY$` returns an empty string.

All characters are passed as read to the program, without reflecting them to the console, except **CTRL** **C** which is ignored.

Associated Keywords.
`INPUT$`

INP

Input from I/O port. Function

Use
Input byte value from the given I/O port.

Form

`INP(`<integer expression>`)`

The <integer expression> must yield a value in the range 0....255.

Associated Keywords.
`OUT, WAIT`

INPUT

Input data from console. Command

Use
To prompt the operator and read data from the console.

Form

`INPUT` [`;`][<quoted string>`;`]<list of: <variable>
`INPUT` [`;`][<quoted string>`,`]<list of: <variable>

Where the <quoted string> is an optional prompt string.

Notes.
The default prompt is a question mark. If an explicit prompt string is given then a question mark is appended to it if the first form is used (with a semi-colon after the prompt string), but not if the second form is used.

When the `INPUT` command is executed the prompt is issued, and a line is read from the console. The line input is treated as a list of items, and BASIC attempts to assign the value of one item to each variable in the list, checking that the item is compatible with the variable. The line read takes the form:

<list of: <item>

Where each <item> may be one of:

a <numeric value>, which may be surrounded by <white space>, which is ignored.

a <quoted string>, which may be surrounded by <white space>, which is ignored. The trailing double quotes may be omitted if the string is the last object on the line.

an <unquoted string>, which may be surrounded by <white space>, which is ignored. An <unquoted string> is between 0 and 255 characters terminated by comma or carriage return.

If there are the right number of items on the line, and their types are all compatible with the corresponding variable, then the values of the items are assigned to the variables.

If there are too few or too many items, or if an item is not compatible with the corresponding variable, then the message '`?Redo from start`' is issued, and the `INPUT` command restarted (so the prompt is repeated).

The optional semi-colon immediately after the INPUT keyword stops BASIC from reflecting the carriage return typed at the end of the input line. This means that the cursor is left at the end of the text just entered.

Associated Keywords.
LINE INPUT, DATA, READ, INPUT #, LINE INPUT #, INKEY$, INPUT$

INPUT

Input data from file. Command

Use
To read data from sequential files, or the random record buffer.

Form

`INPUT #`<file number expression>`,`<list of: <variable>>

The <file number expression> specifies which file to read from. If the file is open for random access, then the data is read from the record buffer.

The <list of: <variable>> specifies where to read data to, and what type of data is to be read.

Notes.
BASIC attempts to read one item from the file for each variable in the list of variables. Each item must be compatible with the variable to which its value is to be assigned. Each item in the file may be one of:

a <numeric value>, which may be preceded by <white space> which is ignored. A numeric item is terminated by <white space>, comma, carriage return or end of file. Trailing <white space> is ignored. Any following comma or carriage return is ignored.

a <quoted string>, which may be preceded by <white space>, which is ignored. All characters after the leading double quotes, including carriage returns and linefeeds, upto the trailing double quotes form the string. The string is terminated by double quotes or end of file. After the trailing double quotes trailing <white space> is ignored. Any following comma or carriage return is ignored.

an <unquoted string>, which may be preceded by <white space>, which is ignored. An <unquoted string> item is terminated by a comma, carriage return or end of file.

(Nulls are ignored throughout. Carriage return followed by linefeed is treated as a carriage return, and vice versa. Strings are limited to 255 characters, and terminate automatically after the 255th.)

If the file is open for random access, then `INPUT#` reads from the record buffer, and will fail (Error 50) if an attempt is made to read beyond the end

of the buffer. An internal pointer is associated with the record, to allow separate INPUT# commands to read along the record. This pointer is reset to the beginning of the buffer each time a record is read (by GET).

Associated Keywords.
INPUT #, LINE INPUT, DATA, READ, LINE INPUT #, INKEY$, INPUT$, GET

INPUT$

Input fixed length string. Function

Use
To input a fixed number of characters either from the keyboard or from a file.

Form

`INPUT$(`<integer expression>[`,`[`#`]<file number expression]`)`

The <integer expression> gives the number of characters to be read, and must yield a value in the range 1....255.

The <file number expression> gives the number of the file from which to read. The file must be open for Random or Input Access. If the file number part is omitted, `INPUT$` reads from the console.

Notes.
When reading from the console INPUT$ puts all characters to the string as input. No characters are reflected to the console output. CTRL **C** is ignored.

When reading from a file open for Random Access it is an error to attempt to read beyond the end of the current record.

Associated Keywords.
`INKEY$, INPUT #`

INSTR

Search for substring in string. Function

Use
To search from a given point in a given string for the first occurrence of another given string.

For

```
INSTR([<integer expression>,]
                    <string expression>,<string expression>)
```

The <integer expression> gives the character position within the searched string at which to start searching, and must yield a value in the range 1....255. If omitted then the search starts at the first character of the searched string.

The first <string expression> gives the string to be searched.

The second <string expression> gives the string to search for.

Notes.
If the searched for string is found, `INSTR` returns the position within the searched string of the start of the first occurrence of the searched for string. Otherwise `INSTR` returns zero.

If there are no characters in the searched string from the given (or default) starting position, then `INSTR` always returns zero.

If the searched for string is null, then it is immediately found, unless the previous rule applies.

INT

Number to integer. Function

Use
To round a given value to the nearest smaller integer (round toward minus infinity). Note that this does not convert floating point numbers to integer format, but simply removes any fraction part.

(For positive numbers this is equivalent to `FIX`. For negative numbers the value returned is one less than `FIX` would return, unless the value was already integral.)

Form

`INT(`<numeric expression>`)`

Associated Keywords.
`CINT, FIX, ROUND`

KILL

Kill file. Command

Use
To erase a file.

Form

KILL <string expression>

The <string expression> yields a string giving the name of the file to be erased.

Notes.
If the <string expression> contains 'wild card' characters, then all matching files are erased.

Associated Keywords.
ERA

LEFT$

Extract left hand part of string. Function

Use
To extract a given number of characters from the left hand end of a given string.

Form

LEFT$(<string expression>,<integer expression>)

The <string expression> gives the string from which to extract characters.

The <integer expression> gives how many characters to extract, and must yield a value in the range 0....255.

Notes.
If the given string is shorter than the required length, then the entire string is returned. Otherwise the left hand end of the string is returned, giving a string of the required length.

Associated Keywords.
MID$, RIGHT$

LEN

Determine length of string. Function

Use
To determine the length of a given string.

Form

`LEN(`<string expression>`)`

Notes.
`LEN` returns an integer in the range 0....255. 0 indicates that the string is null.

All characters in the string are counted, including any non-printing ones.

LET

Preface an assignment. Command

Use
The LET command is not very useful, it is really a hangover from the earliest BASICs, and may be ignored.

Form

LET <variable>=<expression>

Notes.
The expression is evaluated and the resulting value assigned to the variable, unless the two are incompatible.

The LET keyword is, in fact, wholly redundant.

LINE INPUT

Input complete line from console. Command

Use
To prompt the operator and read a complete line of text from the console to a given string variable.

Form

```
LINE INPUT [;][<quoted string>;]<string variable>
LINE INPUT [;][<quoted string>,]<string variable>
```

Where the <quoted string> is an optional prompt string.

The <string variable> gives the name of the variable to which to assign the string created.

Notes.
The default prompt is a question mark. If an explicit prompt string is given then a question mark is appended to it if the first form is used (with a semi-colon after the prompt string), but not if the second form is used.

When the `LINE INPUT` command is executed the prompt is issued, and a line is read from the console and assigned to the given string. The line is terminated by carriage return (or after 255 characters, whichever is the sooner).

If a semi-colon immediately follows the `LINE INPUT` then BASIC will not reflect the carriage return typed at the end of the input line. This means that the cursor is left at the end of the text just entered.

Associated Keywords.
`DATA, READ, INPUT #, LINE INPUT #, INKEY$, INPUT$`

LINE INPUT

Input complete line from file. Command

Use
To read a complete line from a given file to a given string.

Form

LINE INPUT #<file number expression>,<string variable>

Notes.
The file must be open for Input or Random Access.

Associated Keywords.
LINE INPUT

LOAD

Load a program into memory. Command

Use
To read a program from disc into memory, replacing any existing program. The program may optionally be run immediately.

Form

`LOAD` <string expression>[`,R`]

The <string expression> gives the name of the file from which to read the program. If no file type extension is given '`.BAS`' is assumed.

Notes.
The optional '`,R`' indicates that the program is to be run immediately it is loaded.

Any existing program, user functions and variables are deleted from memory. `DEFINT`, `DEFSNG`, `DEFDBL`, `DEFSTR` and `OPTION BASE` settings are reset. Unless the '`,R`' option is specified all files are closed.

In Run Only versions the '`,R`' is required, otherwise BASIC will terminate and return to system level after loading the given program.

LOC

Current Location in file. Function

Use
To establish the current record number in a given file.

Form

`LOC(`<file number expression>`)`

Where <file number expression> gives the number of a file which is open (for any form of access).

Notes.
If the file is open for Random Access `LOC` returns the record number of the record referenced in the last `PUT` or `GET` operation. If there have been no `PUT` or `GET` operations since the file was opened then 0 is returned.

If the file is open for Input or Output Access then `LOC` returns the number of 128 byte records read or written.

LOF

Length of file. Function

Use
To establish the number of logical records in the current physical extent of a given file.

Form

`LOF(`<file number expression>`)`

Where <file number expression> gives the number of a file which is open (for any form of access).

Notes.
See documentation for the underlying operating system.

LOG

Natural logarithm. Function

Use
To calculate natural logarithms.

Form

LOG(<numeric expression>)

Where the result of the <numeric expression> must be greater than zero.

Notes.
While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. LOG returns a Single Length result.

Associated Keywords.
EXP, LOG10

LOG10

Logarithm, base 10. Function

Use
To calculate logarithms to base 10.

Form

LOG10(<numeric expression>)

Where the result of the <numeric expression> must be greater than zero.

Notes.
While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. LOG10 returns a Single Length result.

Associated Keywords.
EXP, LOG

LOWER$

Convert string to lower case. Function

Use
To create a new string which is a copy of another, with all upper case alphabetic characters converted to lower case equivalents.

Form

LOWER$(<string expression>)

Notes.
The result of the <string expression> is returned, with any characters in the range 'A'....'Z' converted to the equivalent character in the range 'a'....'z'.

Associated Keywords.
UPPER$

LPOS

Line printer position. Function

Use
To establish the current print position on the line printer.

Form

`LPOS(<numeric expression>)`

where the <numeric expression> is a dummy argument, so `LPOS(Ø)` is the recommended form.

Notes.
`LPOS` returns a logical position on the line printer, which may bear no relation whatsoever to the current position of the print head!

In calculating the logical position all non-printing characters (those with values < 32) are not counted, except for:

- Back Space (value 8), which counts -1 unless the position is one.
- Tab (value 9), which expands to spaces, each of which is counted. (Tab positions eight characters apart all the way across the printer are implicitly defined.)
- Carriage return (value 13), which sets the position to 1.

The position is also affected by carriage returns inserted by BASIC if the logical position exceeds the width of the printer, as set by `WIDTH LPRINT`.

If the width is set infinite (by `WIDTH LPRINT 255`) then the logical position will grow to 255, but no further.

Associated Keywords.
`WIDTH LPRINT, POS`

LPRINT

Print to the Line Printer. Command

See PRINT, reading Line Printer for Console throughout.

LSET

Set one string to another, left justified. Command

Use
To replace the contents of one string by another, filling with spaces on the right to the same length as the original contents.

Form

LSET <string variable>=<string expression>

Notes.
The <string expression> is evaluated and then forced to the same length as the current value of the given <string variable> - either by padding with spaces at the right hand end, or by discarding characters from the right hand end.

The result replaces the original contents of the <string variable>.

LSET is particularly useful for setting new values into fields of a random record.

Associated Keywords.
FIELD, RSET, MID$, MKI$, MKS$, MKD$

MAX

Determine maximum value. Function

Use
To determine the maximum of a number of values.

Form.

`MAX(`<list of: <numeric expression>`)`

Notes.
`MAX` returns the value of the largest of the <numeric expression>s.

Associated Keywords.
`MIN`

MEMORY

Reset BASIC memory parameters. Command

Use
To change the amount of memory used by BASIC, the maximum number of files and the maximum random record size.

Form.

`MEMORY` [<address expression>][,[<address expression>]
[,[<address expression>][,<address expression>]]]

The first <address expression> gives the address of the highest byte in memory which may be used by BASIC. If omitted the current setting is retained.

The second <address expression> gives the number of bytes that BASIC should use for stack. If omitted the current setting is retained. If present, the expression must yield a value of at least 256.

The third <address expression> gives the maximum number of files that may be handled at once. If omitted the current setting is retained.

The fourth <address expression> specifies the maximum random record size. If omitted the current setting is retained.

Notes.
The optional parameters allow the setting of the total memory size, the maximum number of files and the maximum random record size in much the same way as in the initial command line (see Appendix A).

Changing the stack size causes BASIC to forget any active `FOR`, `WHILE` or `GOSUB` commands.

Reducing the maximum number of files automatically closes any files associated with file numbers greater than the new maximum.

Changing the maximum random record size automatically closes all active files.

Associated Keywords.
`CLEAR`

MID$

Replace part of a string. Command
Return a part of a string. Function

Use

MID$** specifies part of a string (a sub-string) which can be used either as the destination of an assignment (**MID$ as a Command) or as an argument in a string expression (**MID$** as a Function).

Form.

MID$(<string>**,**<integer expression>[**,**<integer expression>]**)**

For **MID$** as a Command <string> must be a <string variable>, part of which is to be altered.

For **MID$** as a Function <string> is a <string expression>, part of which will be returned as the function's value.

The first <integer expression> specifies the position of the character in <string> which is to be the first character of the sub-string. The <integer expression> must yield a value in the range 1....255.

The second <integer expression> specifies the length of the sub-string. If omitted the sub-string extends to the end of the original string. The <integer expression> must yield a value in the range 1....255.

Notes.

The sub-string specified in **MID$** is defined by a starting character position and a length. The first character of the original string is at position 1. The length defaults to all characters after the starting position of the sub-string. If the starting position specified is beyond the end of the string, the sub-string is null. Regardless of the length specified the sub-string always ends at the end of the original string.

When **MID$** is used as a Command it appears on the left hand side of an assignment. On the right hand side must be a <string expression>. The sub-string specified by the **MID$** is replaced by the value of the <string expression>. If the <string expression> yields a string that is shorter than

the sub-string, then the extra characters in the sub-string are unaffected. If the <string expression> yields a string that is longer than the sub-string, then the excess characters are discarded.

Associated Keyword.
LSET, RSET, LEFT$, RIGHT$, FIELD

MIN

Determine minimum value. Function

Use
To determine the minimum of a number of values.

Form.

`MIN(`<list of: <numeric expression>`)`

Notes.
`MIN` returns the value of the smallest of the <numeric expression>s.

Associated Keywords.
`MAX`

MKD$

Make Double Length String. Function

Use
To convert a Double Length value into a eight byte string, so that the value may be stored in that form. This is particularly useful for storing Double Length values in random records.

Form.

`MKD$(`<numeric expression>`)`

Notes.
If the expression does not yield a Double Length result, then it is converted to Double Length.

It is not recommended that the string be processed in any way, other than to assign it to another string. The function `CVD` converts the string back to a Double Length value.

This function does not perform any binary to decimal conversion, so the string is an exact representation of the Double Length value.

Associated Keywords.
`CVD, MKI$, MKS$`

MKI$

Make Integer String. Function

Use
To convert an Integer value into a two byte string, so that the value may be stored in that form. This is particularly useful for storing Integer values in random records.

Form.

`MKI$(`<numeric expression>`)`

Notes.
If the expression does not yield an Integer result, then it is rounded to Integer.

It is not recommended that the string be processed in any way, other than to assign it to another string. The function `CVI` converts the string back to an Integer value.

This function does not perform any binary to decimal conversion.

Associated Keywords.
`CVI, MKS$, MKD$`

MKS$

Make Single Length String. Function

Use
To convert a Single Length value into a four byte string, so that the value may be stored in that form. This is particularly useful for storing Single Length values in random records.

Form.

`MKS$(`<numeric expression>`)`

Notes.
If the expression does not yield a Single Length result, then it is converted to Single Length.

It is not recommended that the string be processed in any way, other than to assign it to another string. The function `CVS` converts the string back to a Single Length value.

This function does not perform any binary to decimal conversion, so the string is an exact representation of the Single Length value.

Associated Keywords.
`CVS, MKI$, MKD$`

NAME

Rename disc file. Command

Form

NAME <string expression> AS <string expression>

The first <string expression> gives the name of the file whose name is to be changed.

The second <string expression> gives the new name for the file.

Notes.
The file must exist. No file with the new name may exist.

Associated Keywords.
REN

NEXT

Step FOR control variable to next value. Command

Use
Delimits the end of a FOR loop. The NEXT command may be anonymous, or may refer to its matching FOR.

Form

NEXT [<list of: <variable>]

Where:
NEXT <variable> , <list of: <variable>
is equivalent to:
NEXT <variable> : NEXT <list of: <variable>

Notes.
The NEXT command defines the end of a FOR loop. The way in which FOR and NEXT are tied together is described under FOR. When a NEXT command is encountered BASIC knows which (if any) FOR it must be associated with. If the NEXT is followed by a variable name, then that name be the same as the name of the control variable in the matching FOR command.

Associated Keywords.
FOR

OCT$

Octal string. Function

Use
To produce a string of octal digits representing the value of the given expression.

Form

OCT$(<unsigned integer expression>[,<integer expression>])

The <unsigned integer expression> must yield a value in -32768....65535.

This value is treated as an unsigned 16 bit value, to be converted into a string of octal characters.

The optional <integer expression> gives the minimum size of string to be produced, and must yield a value in the range 0....16.

Notes.
OCT$ always generates as many characters as are required to represent the number (zero generates at least one digit). If the optional <integer expression> is present and the string is too short, then the string is filled to length with leading zeros. The string is not truncated if it is too long.

Associated Keywords.
HEX$, DEC$, STR$

ON <expression> GOSUB
ON <expression> GOTO

Computed GOSUB & GOTO. Command

Use
To choose one of a number of subroutines to call or a number of lines to jump to, depending on the result on an expression.

Form

ON <integer expression> GOSUB <list of: <line number>
ON <integer expression> GOTO <list of: <line number>

The <integer expression> must yield a value in 0....255.

Notes.
The result of the <integer expression> selects a line number from the list. 1 selects the 1st, 2 the 2nd, and so on. The subroutine starting at the selected line number is called, or the selected line number is jumped to. Zero, or any result greater than the number of line numbers in the list, will do nothing.

Null line number entries in the list are legal. Selecting a null entry is not (Error 2).

Associated Keywords.
GOTO, GOSUB

ON ERROR GOTO

Set Error Trap. Command

Use
When BASIC detects an error while obeying a command it can either take the default action, which is to generate an error message and return to system level, or it can invoke an error handling subroutine in the program. `ON ERROR GOTO` sets BASIC into one of these states.

Form

`ON ERROR GOTO` <line number>

The <line number> specifies the line to which control is to be transferred in the event of an error.

Notes.
Specifying a non-zero line number in an `ON ERROR GOTO` enables error trapping. `ON ERROR GOTO 0` disables error trapping - note special effect of this during error processing, see below.

If an error occurs while error trapping is enabled control is transfered to the line number specified in the `ON ERROR GOTO` command. BASIC is now in Error Processing Mode. The variables `ERR` and `ERL` are set to indicate which error has occurred, and which line it occurred in.

`ON ERROR GOTO 0` in Error Processing Mode not only disables error trapping, but also causes BASIC to take the default error action for the error that was trapped. Error processing routines may, therefore, cope with those errors for which they are competent and invoke the default action for the rest.

If an error is detected during Error Processing Mode BASIC immediately takes the default action.

The `RESUME` command is only legal during Error Processing Mode, and allows the program to resume execution in one of three ways :

- at the statement in which the error occured
- at the statement after the one in which the error occured
- at a specified line number

OPEN

Open file. Command

Use
To open a disc file and associate it with a file number, so that the file may be accessed.

Form

OPEN <mode>,[#]<file number expression>,
<filename>[,<reclen>]

<mode> is a <string expression> specifying the mode of access to the file. The first character of the resulting string must be one of:

'I' - for sequential Input
'O' - for sequential Output
'R' - for Random access

<file number expression> gives the file number with which the file is to be associated, and by which it will be referenced. It is an error (Error 55) to attempt to use a file number which is currently in use.

<file name> is a <string expression> giving the name of the file to be opened.

If the <mode> is 'R' then a record length may be specified. <reclen> is an integer expression, which must yield a value in the range 1....<max>, where <max> is set when BASIC is first loaded (default value 128) or by a CLEAR or MEMORY command. If no <reclen> is specified 128 is assumed.

Associated Keywords.
CLOSE

OPTION BASE

Set the base value for array subscripts. Command

Use
Array subscripts may start from zero or one. The `OPTION BASE` command chooses between these alternatives.

Form

`OPTION BASE` <integer expression>

Where the <integer expression> must yield a value in the range 0....1.

Notes.
Attempting to obey more than one `OPTION BASE` command will generate an error (Error 10).

Attempting to obey an `OPTION BASE` command while any array exists will generate an error (Error 10).

The default base value is 0.

OUT

Output a value to a processor output port. — Command

Use
To send a byte to a given I/O port.

Form

`OUT` <integer expression>`,`<integer expression>

The first <integer expression> gives the I/O address to which the data is to be sent. The expression must yield a value in the range 0....255.

The second <integer expression> gives the data to be sent to the given output port. The expression must yield a value in the range 0....255.

Associated Keywords.
`INP, WAIT`

PEEK

Peek at memory location. Function

Use
To read a given byte of the machine's memory.

Form

PEEK(<address expression>)

Notes.
PEEK returns a value in the range 0....255.

Associated Keywords.
POKE

POKE

Poke a value into machine memory. Command

Use
Allows direct write access to machine memory.

Form

POKE <address expression>,<integer expression>

The <address expression> gives the address of the byte to be written to.

The <integer expression> gives the value to be written to the byte at the given address. The expression must yield a value in the range 0....255.

Associated Keywords.
PEEK

POS

Console position. Function

Use
To establish the current print position on the console.

Form

`POS(<numeric expression>)`

where the <numeric expression> is a dummy argument, so `POS(Ø)` is the recommended form.

Notes.
`POS` returns a logical position on the console, which may bear no relation whatsoever to the current position of the cursor!

In calculating the logical position all non-printing characters (those with values < 32) are not counted, except for:

- Back Space (value 8), which counts -1 unless the position is one.
- Tab (value 9), which expands to spaces, each of which is counted. (Tab positions eight characters apart all the way across the console are implicitly defined.)
- Carriage return (value 13), which sets the position to 1.

The position is also affected by carriage returns inserted by BASIC if the logical position exceeds the width of the console, as set by `WIDTH`.

If the width is set infinite (by `WIDTH 255`) then the logical position will grow to 255, but no further.

Associated Keywords.
`WIDTH, LPOS`

PRINT

Print to console. Command

Use
To print data, numeric or string, to the console.

Form

PRINT [<print list>][<using clause>][<separator>]

<print list> is: <print item>[<separator><print item>]*
where <print item> is: <expression>
or: SPC(<integer expression>)
or: TAB(<integer expression>)

<using clause> is: USING <string expression>;[<using list>]

Where <using list> is: <expression>[<separator><expression>]*

<separator> is: comma or semi-colon.

(Note that the [....]* metalanguage construct means that the object is optional, but may repeated any number of times.)

1.Effect.
The <print items>, if any, in the <print list> are evaluated and printed in 'free format'. A comma following a <print item> causes BASIC to advance to the beginning of the next 'print zone' after the item is printed. A semi-colon serves only to separate the items, and has no effect on the output.

If a <using clause> is present, then the <string expression> defines a template which controls the format in which the values of the expressions in the <using list> are printed. Commas or semi-colons may be used to separate the items in the <using list>, neither has any effect on the output.

When all the arguments have been processed a carriage return is issued, unless the PRINT command ends in a <separator>.

2. Free Format Printing.
The <print items> are evaluated in turn and their result printed. Numeric results are printed as described below. String results are sent to the console character by character. The effects of SPC & TAB are described below.

A comma following a <print item> causes BASIC to space forward to the start of the next print zone before processing the next expresssion. A semi-colon simply separates expressions.

2.1 Print Zones and Console Width.

BASIC divides the console line into zones, whose width is set by the `ZONE` command, and which by default are 15 characters wide. When `PRINT` spaces forward to the next print zone BASIC will start a new line if there are less than 'zone width' characters to the edge of the console.

Before printing the result of an expression BASIC checks that there is enough room for it between the current position and the edge of the console (as defined by the current `WIDTH`), unless the current position is the beginning of the line. If there is not enough room, a new line is started before the result is printed.

2.2 Free Format Number Printing.

Positive numbers are preceded by a space; negative numbers by a minus sign.

All numbers are printed with the minimum of characters. No decimal point is printed if there are no significant fraction digits. At least one digit is printed before any decimal point - so numbers may include a leading zero.

Single length numbers are printed in scaled format (with an exponent) if they cannot be accurately represented by a number with seven or less digits (excluding a possible leading zero).

Double length numbers are printed in scaled format (with an exponent) if they cannot be accurately represented by a number with sixteen or less digits (excluding a possible leading zero).

All numbers are followed by a space.

2.3 SPC Print Function.

Use

SPC prints a given number of spaces.

Form

`SPC(`<integer expression>`)`

If <integer expression> yields a negative value, zero is assumed. If the

value is greater than the device width, then it is reduced to the range 1....device width by a suitable number of subtractions of the device width. The resulting value gives the number of spaces to print.

Notes.
The given number of spaces are printed, starting unconditionally at the current position. `SPACE$` is not quite equivalent, since BASIC would check that the string fitted on the current line, and possibly issue a carriage return before the spaces.

`SPC` need not be followed by a comma or semi-colon, a semi-colon is assumed (including when `SPC` is the last item in the print command).

2.4 TAB Print Function.

Use
`TAB` prints spaces to move to the the given print position.

Form

`TAB(`<integer expression>`)`

If <integer expression> yields a value less than 1, 1 is assumed. If the value is greater than the device width, then it is reduced to the range 1....device width by a suitable number of subtractions of the device width. The resulting value gives the print position to which to move.

Notes.
If the required print position is greater than or equal to the current position, spaces are printed until the the required position is reached (this may print nothing at all).

If the required print position is less than the current position, BASIC issues a carriage return followed by spaces to reach the required position on the new line.

`TAB` need not be followed by a comma or semi-colon, a semi-colon is assumed (including when `TAB` is the last item in the print command).

3. USING Formatted Printing.

The expressions in the expression list are evaluated in turn, and printed according to the format template. As each expression is processed the template is processed to find a suitable format specification for its result. If the template string is exhausted while searching for a format specification the process is restarted from the beginning of the string. If no suitable format is found an error (Error 5) is generated.

3.1 Format Template.

The format template is a string which is interpreted character by character to control the way in which the result of each expression is printed. The meaning of characters in the template is described below. The following characters are recognised in format specifications :

```
! \ & # . + - * $ ^ , _
```

Any other character will be printed immediately it is encountered while processing the format template against the expression list. Characters in this list will be printed, if found out of context. The "_" character is not printed, but causes the following character to be printed 'as is'.

3.2 Format Specifications suitable for Strings.

'!' Only the first character of the string is printed.

'\<spaces>\'
The first 'n' characters of the string are printed, where the string '\<spaces>\' is 'n' characters long.

'&' The complete string is printed 'as is'.

3.3 Format Specifications suitable for Numbers.

Numbers may be printed with (scaled) or without (unscaled) an exponent part. If an exponent part is specified a number of the other options are inoperable. The template for a number may not exceed 24 characters, not counting the exponent and trailing sign options.

Numbers are rounded to the number of digits printed.

The body of the number:

'#' Each '#' specifies a digit position.

'.' Specifies the position of the decimal point. There may be at most one '.' in a number format specification.

'**,**' May appear before '**.**'. Specifies a digit position, and also requests that digits before the decimal point be divided into groups of three, separated by commas.

(The latter is suppressed if an exponent part is specified)

Leading Dollar and Asterisk options:

The following are mutually exclusive options. If an exponent part is specified they have no effect, except to specify digit positions. These options must be specified immediately before the body of the number :

'**$$**' Specifies two digit positions. Specifies that a '**$**' sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the '**$**' will occupy one of the 'digit' positions.

'******' Specifies two digit positions. Specifies that any leading spaces be replaced by '*****'s.

'****$**' Specifies three digit positions. Acts as the previous two options combined.

Sign Options:

The default is for '**-**' to be printed immediately before the number (and any leading '**$**') if the number is negative, and no sign at all if the number is positive. The '**-**' will occupy one of the digit positions before the decimal point.It is possible to specify that '**+**' be printed for positive numbers, and that the sign is to follow the number.

'**+**' Specifies that '**+**' or '**-**' is to be printed, as appropriate.

If the '**+**' appears at the beginning of the format specification, the sign is printed immediately before the number (and any leading dollar).
If the '**+**' appears at the end of the format specification, the sign is printed after the number (and any exponent part).

'**-**' May only appear at the end of a format specification. Causes '**-**' to be printed if the number is negative, and a space to be printed if it is positive. This sign indication is printed after the number (and any exponent part).

Exponent Option:

'**^^^^**' following the body of the number, and preceding any trailing sign indication, enables the exponent option. The four characters reserve space for the four character exponent part.

As noted above, specifying the Exponent option suppresses the effects of the Dollar, Asterisk and Comma options.

The body of the number is printed with the maximum possible number of digits before the decimal point, reserving one digit position for the sign (even if the number is positive) if no other provision is made for it.

Field Overflow:

If a number cannot be printed within the format given, BASIC attempts to get as close as it can to the required format, but precedes the result by a '%' to indicate field overflow.

No number whose absolute value is greater than or equal to 10^16 can be printed in unscaled format. Any attempt to do so causes the number to be printed in free format preceded by '%' to indicate a format failure.

4. Print Continuation.

Terminating the `PRINT` command with a <separator> stops this `PRINT` from issuing a newline, and causes the next `PRINT` command to be treated as a continuation of this one.

Associated Keywords.
`LPRINT, PRINT #, WIDTH, ZONE`

PRINT

Print to file. Command

Use
To put data to a file, formatted in the same way as PRINTing to the console.

Form

PRINT #<filenumber expression>[,<PRINT arguments>]

The <filenumber expression> specifies the file to be printed to. The file must be open for Output or for Random access.

The <PRINT arguments> take precisely the same form as in the PRINT command. If no arguments are given, a carriage return is written.

Notes.
PRINT # is the same as PRINT, except that the output is sent to a file, not to the console, and that lines in a file are thought to be infinitely wide. The handling of the <PRINT arguments> is identical. A logical position is maintained in the same was as in PRINT, so the TAB Print Function will work in PRINT #. Tab characters (value 9), however, are not expanded, and do not affect the logical position.

It is possible to use PRINT # to output to a Random access file. Characters are put to the random record buffer. It is an error to attempt to overfill the random record buffer.

Associated Keywords.
PRINT, WRITE #

PUT

Put data to random access file. Command

Use
To put the contents of the random record buffer to the given record in the given file.

Form

PUT [#]<file number expression>[,<integer expression>]

<file number expression> specifies the file to be written to, which must be open for random access.

<integer expression> specifies which record to write to, and must yield a value in the range 1....32767. If omitted, the record after the one in the last PUT or GET for this file is written - if this is the first PUT or GET since the file was opened, then the first record of the file is written.

Notes.
An internal pointer is associated with the random record buffer, to allow PRINT # et al to put data to the record. The PUT operation resets that pointer.

Associated Keywords.
PRINT #, WRITE #

RANDOMIZE

Randomise the current random number seed.

Command

Use

BASIC's random number generator produces a pseudo random sequence, in which the each number depends completely on the previous number. Starting from a given value the sequence is always the same. `RANDOMIZE` sets a new initial value for the random number generator, either to a given value, or to a value entered by the operator.

Form

`RANDOMIZE` [<integer expression>]

If the expression is omitted then the operator is prompted :

```
Random Number Seed?
```

to get a suitable value, which may be any form of number.

Notes.

Setting the initial random number to the same number will result in the same sequence being generated.

Associated Keywords.

`RND`

READ

Read from DATA statements. Command

Use
READ fetches data from DATA statements and assigns it to variables.

Form

READ <list of: <variable>

Notes.
All the DATA statements in a program, taken in order, form a list of constants. These constants may be read into variables by the READ command. READ steps on to the next constant in the list and assigns it to the next variable in the list of variables given. The constant and the variable to which it is read must be compatible. Attempting to READ past the last DATA constant generates an error (Error 4).

Associated Keywords
DATA, RESTORE

REM

Remark. Command

Use
To put comments into BASIC programs.

Form

`REM` <rest of line>

BASIC ignores the rest of the line after the `REM` command. Note that colon (statement separator) is also ignored.

Notes.
It is permissable to transfer control to a `REM` command, by `GOTO` or by `GOSUB`.

A single quote character in a line (not in a string) is equivalent to `:REM`, and may also prefix comment text. The one exception to this is in a `DATA` command, where a single quote is treated as part of an unquoted string.

REN

Rename file. Command

Use
To change the name of a disc file.

Form

`REN` <new filename>=<old filename>

The <old filename> gives the name of the file whose name is to be changed.

The <new filename> gives the new name for the file.

Notes.
The file must exist. No file with the new name may exist.

`REN` treats the rest of the current line as its arguments, irrespective of any colons or single quotes.

Associated Keywords.
`NAME`

RESET

Reset file system. Command

Use
To reset the file system so that discs may be changed.

Form

```
RESET
```

Notes.
All files are closed.

RESTORE

Restore pointer into DATA list. Command

Use
To move the internal pointer to the next DATA item to a given position (typically back to the first DATA statement).

Form

RESTORE [<line number>]

The <line number> gives the line to which the DATA pointer is to be set. If omitted the pointer is set back to the start of the program.

Notes.
READ commands move a pointer to DATA items through the program. The RESTORE command moves that pointer to a specified position in the program. The next READ command executed will search forward from that point for the next DATA statement, and start reading from there.

Associated Keywords.
READ, DATA

RESUME

Resume execution after processing an error.

Command Legal only in Error Processing Mode

Use
When an error has been trapped by an `ON ERROR GOTO` routine, and has been processed by it, `RESUME` allows normal execution to continue, from a variety of points.

Form

`RESUME` [<line number>]or
`RESUME NEXT`

Notes.
`RESUME` is legal only during Error Processing Mode (ie in an `ON ERROR GOTO` routine).

`RESUME 0`, or `RESUME` where the line number has been omitted, returns control to the beginning of the statement in which the original error was detected.

`RESUME NEXT` returns control to the statement immediately after the statement in which the original error was detected.

`RESUME` <non-zero line number> returns control at the specified line.

Associated Keywords.
`ON ERROR GOTO`

RIGHT$

Extract right hand part of string. Function

Use
To extract a given number of characters from the right hand end of a given string.

Form

RIGHT$(<string expression>,<integer expression>)

The <string expression> gives the string from which to extract characters.

The <integer expression> gives how many characters to extract, and must yield a value in the range 0....255.

Notes.
If the given string is shorter than the required length, then the entire string is returned. Otherwise the right hand end of the string is returned, giving a string of the required length.

Associated Keywords.
MID$, LEFT$

RND

Random Number. Function

Use
To get a random number. This may be the next in the current sequence, the last random number repeated or the first in a new sequence.

Form

`RND[(`⩽numeric expression⩾`)]`

Notes.
`RND` returns a Single Length value, where 0 < value < 1. The pseudo random number generator produces each number by operating on the previous one, so that from the same initial value, the same sequence is produced.

`RND` with the expression omitted, or with an expression yielding a value greater than zero, returns the next random number in the current sequence.

`RND` with an expression yielding the value zero returns a copy of the last random number generated.

`RND` with an expression yielding a value less than zero starts a new sequence loosely, but predictably, based on that value. The first number in the new sequence is returned.

Associated Keywords.
`RANDOMIZE`

ROUND

Round value. Function

Use
To round a value to a given number of decimal places, or to a given power of ten.

Form

`ROUND(`<numeric expression>[`,`<integer expression>]`)`

The <numeric expression> yields the value to be rounded, which may be of any numeric type.

The optional <integer expression> specifies the rounding, and must yield a value in the range −39....+39. If the <integer expression> is omitted, zero is assumed.

Notes.
The value of the <numeric expression> is rounded to the number of decimal places specified by the <integer expression>, as follows :

<integer expression> greater than 0
the value is rounded to the given number of digits after the decimal point.

<integer expression> zero or absent
the value is rounded to an integer

<integer expression> less than zero
the value is rounded to give `ABS(`<integer expression>`)` zeros before the decimal point.

(Rounding means round to nearest, away from zero if equidistant.)

Associated Keywords.
`INT, FIX, CINT`

RSET

Set one string to another, right iustified. — Command

Use
To replace the contents of one string by another, filling with spaces on the left to the same length as the original contents.

Form

RSET <string variable>=<string expression>

Notes.
The <string expression> is evaluated and then forced to the same length as the current value of the given <string variable> - either by padding with spaces at the left hand end, or by discarding characters from the left hand end.

The result replaces the original contents of the <string variable>.

RSET is particularly useful for setting new values into fields of a random record.

Associated Keywords.
FIELD, LSET, MID$, MKI$, MKS$ MKD$

RUN <filename>

Load and start executing a program. Command

Use
To load a program from file and start executing it.

Form

RUN <string expression>[,R]

The <string expression> gives the filename of the file to load from. If no type extension is given '.BAS' is assumed.

Notes.
Any existing program, user functions and variables are deleted from memory. DEFINT, DEFSNG, DEFDBL, DEFSTR and OPTION BASE settings are reset. Unless the ',R' option is specified all files are closed.

RUN <filename> is equivalent to LOAD <filename> followed by RUN.

RUN <filename>,R is equivalent to LOAD <filename>,R.

Associated Keywords:
LOAD

RUN [<line number>]

Start executing current program. Command

Use
To start executing the current program, either at the beginning, or at a given line.

Form

RUN [<line number>]

If the <line number> is omitted, then execution starts at the first line of the program.

Notes.
Any existing program, user functions and variables are deleted from memory. DEFINT, DEFSNG, DEFDBL, DEFSTR and OPTION BASE settings are reset. All files are closed.

SAVE

Save program on disc. Command

Use
To write the program currently in memory to disc. Three formats for program files are supported.

Form

SAVE >string expression>[,<format specifier>]

The <string expression> gives the name of the file to which the program is to be written. If no type extension is specified, '.BAS' is assumed. If a file of that name already exists it is deleted.

The <format specifier> may be one of:

A - specifying ASCII format.
P - specifying Protected format.

The default format is Compressed form.

Notes.
Compressed format saved files contain the BASIC program in a processed form, suitable only for reloading into memory by BASIC.

ASCII format saved files contain the BASIC program in the same form as in a listing. These files may be processed by other programs - for instance text editors.

Protected format saved files are similar to Compressed format files, except that the program is encrypted. BASIC disallows any command that gives access to the program text after loading a protected format file.

SGN

Sign of value. — Function

Use
To determine the sign of a given value.

Form

`SGN(`<numeric expression>`)`

Notes.
`SGN` returns an Integer value :

−1 if <numeric expression> is < 0
0 if <numeric expression> is = 0
+1 if <numeric expression> is > 0

Associated Keywords.
`ABS`

SIN

Sine. Function

Use
To calculate the Sine of a given value, given that the argument is expressed in radians.

Form

`SIN(`<numeric expression>`)`

The <numeric expression> gives the angle in radians, and must yield a value in the approximate range −200,000....200,000.

Notes.
With angles very much greater than 2*PI the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI....+PI. Rather than return a very inaccurate figure, BASIC will not evaluate `SIN` for values much beyond the range given above.

While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `SIN` returns a Single Length value.

Associated Keywords.
`COS, TAN, ATN`

SPACE$

String of spaces. Function

Use
To create a string of spaces of a given length.

Form

`SPACE$(`<integer expression>`)`

Where <integer expression> must yield a value in the range 0....255, and specifies the length required. (A length of zero returns a null string.)

Associated Keywords.
`SPC, TAB, STRING$`

SQR

Square Root. Function

Use
Evaluate the square root of a given value.

Form

`SQR(`<numeric expression>`)`

Where the <numeric expression> must yield a positive value.

Notes.
While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `SQR` returns a Single Length result.

STOP

Stop execution. Command

Use
To stop execution of a program, leave BASIC and return to system level.

Form

```
STOP
```

Notes.
All open files are closed.

Associated Keywords.
END, SYSTEM

STR$

String representation of numeric value. Function

Use
To convert given numeric value to a decimal string representation.

Form

`STR$(`<numeric expression>`)`

Notes.
The value of the <numeric expression> is converted to a decimal string in the same form as used in a `PRINT` command. Note that positive values yield a string with a leading space, where negative values have a leading minus sign.

Associated Keywords.
`VAL, PRINT, DEC$, HEX$, OCT$`

STRIP$

Strip bit 7 from all characters in a string. — Function

Use
To create a string which is a copy of another, except that Bit 7 of every character in the string is set to zero.

Form

`STRIP$(`<string expression>`)`

Notes.
The result of the string expression is returned, with every character forced into the normal ASCII range (values 0 ....127) by setting the top bit (Bit 7) to zero. This may be used to remove any parity setting on characters, or to remove markers which make use of the 'extra' bit.

Associated Keywords.
`LOWER$, UPPER$`

STRING$

String of a particular character. Function

Use
To construct a string consisting of a given character repeated a given number of times. The character may be specified by its numeric value.

Form

`STRING$(`<integer expression>`,`<character specifier>`)`

The <integer expression> gives the required length of the result string, and must yield a value in the range 0....255.

The <character specifier> may be one of:

<integer expression>
specifying `CHR$(`<integer expression>`)`
<string expression>
specifying the first character of the string

Associated Keywords.
`SPACE$`

SWAP

Swap the contents of two variables. Command

Use
To exchange the contents of two variables, without requiring an intermediate variable.

Form

`SWAP` <variable>`,`<variable>

The two <variables> must be of the same type.

SYSTEM

Return to system level. Command

Use
To stop execution of a program, leave BASIC and return to system level.

Form

```
SYSTEM
```

Notes.
All open files are closed.

Associated Keywords.
END, STOP

TAN

Tangent. Function

Use
To calculate the Tangent of a given value, given that the argument is expressed in radians.

Form

`TAN(`<numeric expression>`)`

The <numeric expression> gives the angle in radians, and must yield a value in the approximate range −200,000....200,000.

Notes.
With angles very much greater than 2*PI the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI....+PI. Rather than return a very inaccurate figure, BASIC will not evaluate `TAN` for values much beyond the range given above.

While the <numeric expression> may be of any numeric type, the argument is forced to Single Length before it is processed. `TAN` returns a Single Length value.

Associated Keywords.
`COS, SIN, ATN`

TYPE

Type file to console. Command

Use
To print a given file on the console.

Form

TYPE <filename>

The filename may not contain 'wild card' characters.

Notes.
The file specified is read and written directly to the console.

TYPE takes the rest of the current line as its argument, irrespective of colons or single quotes.

Tabs and carriage returns are treated in the usual way, and console width is enforced. All other characters are sent to the console exactly as read from the file, so control characters and characters outside the normal ASCII range may be included in the file to produce special console effects.

Associated Keywords.
DISPLAY

UPPER$

Convert string to upper case. Function

Use
To create a new string which is a copy of another, with all lower case alphabetic characters converted to upper case equivalents.

Form

UPPER$(<string expression>)

Notes.
The result of the <string expression> is returned, with any characters in the range 'a'....'z' converted to the equivalent character in the range 'A'....'Z'.

Associated Keywords.
LOWER$

UNT

Convert unsigned integer. Function

Use
To convert the given value to an integer in the range 0....65535, that is an unsigned sixteen bit integer.

Form

`UNT(`<address expression>`)`

Notes.
`UNT` returns an Integer value in the range −32,768....32767 (ie the 2's complement equivalent of the unsigned value of the <address expression>).

Associated Keywords.
`INT, FIX, CINT, ROUND`

USR

Invoke external user function. Function

Use
Invokes one of the ten possible external user functions. External functions are defined by `DEF USR` commands, in which the machine address of the function is given.

Form

`USR[ <digit>](<expression>)`

The <digit> may be in `"Ø"....."9"`. If <digit> is omitted, `" Ø "` is assumed. `USR`<digit> must previously have been defined in a `DEF USR` command.

The <expression> may be any form of expression.

Notes.
See Appendix C for a discussion of external functions and subroutines.

Associated Keywords.
`DEF USR, CALL`

VAL

Convert string to numeric value. Function

Use
To take a string representation of a number and convert it to a numeric value.

Form

VAL(<string expression>)

Notes.
VAL converts the given string to a number in the same way as INPUT. The result returned will either be Integer or Double Length (so if there is any fraction part VAL returns the maximum possible precision).

Associated Keywords.
STR$

VARPTR

Get pointer to variable. Function

Use
To get the address of a given variable or file buffer, so that it can be passed to an external function or subroutine.

Form

VARPTR (<variable>)
or VARPTR (#<file number expression>)

The first form returns the address of the given variable. If the variable is subscripted, the address of that element of the array is returned.

The second form returns the address of the buffer for the given file. The file must be open.

Notes.
The value returned is an Integer in the range −32768....32767 (ie the 2's complement equivalent of the actual address.)

See Appendix C for a discussion of external functions and subroutines.

Associated Keywords.
CALL, USR, DEF USR, UNT

WAIT

Wait on I/O port. Command

Use
To wait until a given I/O port returns a particular value.

Form

WAIT <integer expression>,<integer expression>
[,<integerexpression>]

The first <integer expression> gives the I/O address of the port to wait on. The expression must yield a value in the range 0....255.

The second <integer expression> gives a value to be ANDed with the value read from the I/O port. The expression must yield a value in the range 0....255.

The third, optional, <integer expression> gives a value to be Exclusive ORed with the value read from the I/O port. The expression must yield a value in the range 0....255. If omitted, zero is assumed.

Notes.
BASIC enters a loop reading from the I/O port, Exclusive ORing and then ANDing the value read with the numbers given, until the result is non zero.

Note that there is no way of interrupting this loop, so BASIC will remain in it indefinitely if the required condition never occurs.

Associated Keywords.
INP, OUT

WEND

Delimit end of a While loop. Command

Use
A While loop repeatedly executes a body of program until a given condition is true. The `WEND` command defines the end of the loop.

Form

```
WEND
```

Notes.
The `WEND` command defines the end of a `WHILE` loop. The way in which `WHILE` and `WEND` are tied together is described under `WHILE`. When a `WEND` command is encountered BASIC knows which (if any) `WHILE` it must be associated with.

Associated Keywords.
`WHILE`

WHILE

While loop. Command

Use
A While loop repeatedly executes a body of program until a given condition is true. The `WHILE` command defines the head of the loop, and gives the condition.

Form

`WHILE` <logical expression>

Notes.
When the `WHILE` command is obeyed the <logical expression> is evaluated. If the result is zero BASIC skips to the matching `WEND`. If the result is not zero, execution continues until the matching `WEND` is met, whereupon BASIC loops back to the `WHILE` command, and the process is repeated.

The `WEND` command which matches a given `WHILE` is established statically when the `WHILE` is first executed. That is to say that the `WEND` which matches the `WHILE` depends on the order of statements in the program, quite independent of the order of execution. It is not possible, therefore, to have more than one `WEND` associated with a given `WHILE`.

`WHILE` loops may be nested.

It is permissable to terminate a `WHILE` loop by avoiding the `WEND`.

Associated Keywords.
`WEND, FOR`

WIDTH

Set width of console. Command

Use
To tell BASIC how wide the console is in characters. This information is used when printing to the console, to allow BASIC to insert carriage returns at the appropriate moments.

Since many consoles automatically insert carriage returns, `WIDTH` may be used to tell BASIC where the console does this, so that it can avoid inserting extra carriage returns.

Form

`WIDTH` <integer expression>[`,`<integer expression>]

The <integer expression> gives the console width, and must yield a value in the range 1....255 (though small values may have curious effects).

The optional second <integer expression> gives the console's automatic carriage return column, and must yield a value in the range 1....255.

Notes.
The initial value of console width is 80. The initial value of automatic carriage return column is 255 (ie. none).

Setting the width to 255 has special meaning. BASIC treats the console as being infinitely wide, so never inserts carriage returns. BASIC maintains a counter giving the logical position on the console. When the logical position reaches 255 the counter is no longer incremented, so all positions greater than 255 are treated as 255.

Setting the automatic carriage return column to 255 causes BASIC to assume that the console never inserts carriage returns. Any other setting gives the column immediately after which the console will insert a carriage return. BASIC will avoid inserting an extra carriage return when the width and automatic carriage return column are equal.

The width setting affects all output to the console.

Associated Keywords.
`WIDTH LPRINT, PRINT, POS`

WIDTH LPRINT

Set width of line printer. Command

Use
To tell BASIC how wide the line printer is in characters. This information is used when printing to the line printer, to allow BASIC to insert carriage returns at the appropriate moments.

Form

`WIDTH LPRINT` <integer expression>

The <integer expression> must yield a value in the range 1....255 (though small values may have curious effects).

Notes.
The initial value of line printer width is 132.

Setting the width to 255 has special meaning. BASIC treats the line printer as being infinitely wide, so never inserts carriage returns. BASIC maintains a counter giving the logical position on the printer. When the logical position reaches 255 the counter is no longer incremented, so all positions greater than 255 are treated as 255.

The width setting affects all output to the line printer.

Associated Keywords.
`WIDTH, LPRINT, LPOS`

WRITE

Write to console. Command

Use
To print the values of a number of expressions to the console, separating them by commas and enclosing strings in double quotes.

Form

`WRITE` [<write list>]

<write list> is: <expression>[<separator><expression>]*

where <separator> may be comma or semi-colon, interchangeably.

Notes.
Write is similar to `PRINT`, except that:

- print zones are ignored
- strings are printed enclosed in double quotes
- commas are added between the printed items
- `WRITE` does not support the trailing separator option

Associated Keywords.
`WRITE #, PRINT`

WRITE

Write to file. Command

Use
To print the values of a number of expressions to the given file separating them by commas and enclosing strings in double quotes. `WRITE #` puts data to the file in a form that `INPUT #` is able to read back.

Form

`WRITE #`<file number expression>`,`[<write list>]

The <file number expression> specifies the file to write to. The file must be open for Output or Random Access.

<write list> is: <expression>[<separator><expression>]*
where <separator> may be comma or semi-colon, interchangeably.

(The [....]* construct indicates an optional part, which, if present, may be repeated any number of times.)

Notes.
`WRITE #` is similar to `PRINT #` except that:

- print zones are ignored
- strings are printed enclosed in double quotes
- commas are added between the printed items
- `WRITE #` does not support the trailing separator option
- If the file is open for Random Access the record is space filled to the record length-1 before the carriage return is inserted.

Associated Keywords.
`WRITE, PRINT #`

ZONE

Set Print Zone Size. Command

Use
To change the width of the Print Zone used in PRINT, LPRINT and PRINT #.

Form

ZONE <integer expression>

The <integer expression> gives the new Print Zone width, and must yield a value in the range 1....255.

Notes.
The default Print Zone width is 15. The width is reset to the default value whenever a new program is loaded, that is by NEW, LOAD, CHAIN and RUN 'file' commands.

Associated Keywords.
PRINT, LPRINT, PRINT #, WIDTH

Appendix A
Initialising BASIC

When BASIC is loaded by the host operating system the command line may include a number of parameters, as follows:

<file name>[/F:<number of files>]
[/M:<address>][/S:<size>]

When BASIC has completed initialisation, the file specified by the <file name> is loaded and execution begins immediately.

The other parameters may be given in any order.

/F: BASIC can handle a limited number of files at once, **/F:** sets this number. The <number of files> may be in the range 0....255 (though it may prove difficult to find enough memory for more than a hundred files). If no **/F:** is specified, the maximum is set to 3.

Each file requires 47 bytes of memory plus the buffer specified by the **/S:** setting, unless that buffer is less than 128 bytes, when a total of 175 bytes are required.

/M: Sets the limit on BASIC's memory use. The <address> gives the address of the last (highest address) byte of memory which may be used by BASIC. If no **/M:** is specified BASIC uses as much memory as it can, as given by the value at locations 6 and 7.

BASIC requires at least 1024 bytes of free space to run at all.

/S: Sets the maximum size of random records. The <size> is expressed in bytes. If no **/S:** is specified, 128 is assumed.

The various numbers may be unsigned decimal integers (maximum value 65535), or may be entered in Octal or Hexadecimal (using **&O** and **&H** notation).

These values may be altered after BASIC is loaded by using the **MEMORY** and **CLEAR** commands.

Appendix B
Error Numbers and Error Messages

All errors generated by BASIC are listed here, in error number order. The messages produced by BASIC are given, as well a brief description of possible causes.

1 `Unexpected NEXT`

A `NEXT` command has been encountered while not in a `FOR` loop, or the control variable in the `NEXT` command does not match that in the `FOR`.

2 `Syntax Error`

3 `Unexpected RETURN`

A `RETURN` command has been encountered when not in a subroutine.

4 `DATA exhausted`

A `READ` command has attempted to read beyond the end of the last `DATA`.

5 `Improper argument`

This is a general purpose error. The value of a function's argument, or a command parameter is invalid in some way.

6 `Overflow`

The result of an arithmetic operation has overflowed. This may be a floating point overflow, in which case some operation has yielded a value greater than 1.7E+38 (approx.). Alternatively, this may be the result of a failed attempt to change a floating point number to a 16 bit signed integer.

7 `Memory full`

The current program or its variables may be simply too big. If the control structure is very deeply nested (nested `GOSUB`s, `WHILE`s or `FOR`s)

then the stack may be too small - the **CLEAR** or **MEMORY** commands allow more stack to be specified.

8 Line does not exist

The line referenced cannot be found.

9 Subscript out of range

One of the subscripts in an array reference is too big or too small.

10 Array already dimensioned

One of the arrays in a **DIM** statement has already been declared, or an **OPTION BASE** command has been issued again or too late.

11 Division by zero

May occur in floating point division, integer division, integer modulus or in exponentiation.

13 Type mismatch

A numeric value has been presented where a string value is required. and vice versa, or an invalidly formed number has been found in **READ** or **INPUT #**.

14 String space full

So many strings have been created that there is no further room available, even after 'garbage collection'.

15 String too long

String exceeds 255 characters in length. May be generated by concatenating strings.

16 String expression too complex

String expressions may generate a number of intermediate string values. When the number of these values exceeds a reasonable limit, BASIC gives up and this error results.

18 Unknown user function

No DEF FN has been executed for the FN just invoked.

19 RESUME missing

The end of the program has been encountered while in Error Processing Mode (ie in an ON ERROR GOTO routine).

20 Unexpected RESUME

RESUME is only valid while in Error Processing Mode (ie in an ON ERROR GOTO routine).

22 Operand missing

BASIC has encountered an incomplete expression.

23 Line too long

26 NEXT missing

Cannot find a NEXT to match a FOR command.

29 WEND missing

Cannot find a WEND to match a WHILE command.

30 Unexpected WEND

Encountered a WEND when not in a WHILE loop, or a WEND that does not match the current WHILE loop.

50 Record overflow

Run out of record in a FIELD command, or any command reading or writing the random record buffer.

52 File number error

An out of range file number has been specified.

53 File not found

Unable to open the required file.

54 File type error

This error occurs when BASIC is reading what it expects is a program file, but cannot recognise the contents.

55 File already open

The file number specified is in use.

57 Disc I/O error

58 File already exists

The new name in a **NAME** or **REN** command specifies a file which already exists.

61 Disc full

62 EOF met

An attempt has been made to read past the end of a sequential file.

63 Record number error

A record number outside the legal range 1....32767 has been specified.

64 File name invalid

66 Direct command found

When loading a program file BASIC has encountered a direct command. May indicate that the file is an ASCII file generated outside BASIC and which is not in a suitable form.

67 Directory full

Appendix C
External Routines

Commands exist to invoke external machine language routines, which must be loaded into machine memory above the highest address used by BASIC. Two forms of routine are supported - USR Functions and CALLed Subroutines.

1. Data Formats.

This is a brief description of the format of the four data types. All are held with the least significant byte first (at the lowest address).

1.1 Integer.

Two's complement 16 bit values.

1.2 String.

Strings are held in two parts, the String Descriptor and the String Body. The String Descriptor is a three byte vector. The least significant byte gives the length of the string, zero implies the string is null and that the rest of the descriptor should be ignored. The next two bytes give the machine address of the string body.

1.3 Single Length.

A three byte mantissa followed by a one byte exponent. Numbers are always held normalised, and the most significant bit of the mantissa is replaced by the sign. The exponent is biased by 128. A zero exponent means that the number is zero, and the mantissa should be ignored. The mantissa is in sign and magnitude form with the binary point to the left of the implied most significant bit.

1.4 Double Length.

As single length, but with four more bytes of mantissa.

2. USR Functions.

Ten names are reserved for external functions. They are USRØ to USR9. The DEF USR command declares the address of an external function and assigns it to the given USR name. USR functions are invoked in the usual

way and take one parameter, which may be of any type. The function must return a value, which may either be of the same type as the parameter or of Integer type.

The entry conditions of the function are as follows:

Register A contains a value indicating the type of the parameter:

2 → Integer Parameter
3 → String Parameter
4 → Single Length Parameter
8 → Double Length Parameter

If the parameter is a string the register pair **DE** contains the address of the String Descriptor, as described above.If the parameter is numeric the register pair **HL** contains the address of the number, except for Double Length where it contains the address of the fifth byte of the mantissa.

The state of other registers is undefined.

Note that there is limited stack available, so functions with large stack requirements should have their own stack area.

The exit conditions are:

All registers and flags may be corrupted.

Either:

A value of the same type as the original returned in the area occupied by the parameter.

Or:

An Integer value returned using the **RETURN INTEGER** subroutine (see below).

While **USR** functions may take a String parameter and return a String value, the function MAY NOT alter the String Descriptor IN ANY WAY.

It is expected that **USR** functions will, in general, require Integer parameters and return Integer values. Two routines are provided by BASIC

to support this pattern of use. They are :

`GET INTEGER`

Force the parameter to an Integer (in the range −32768....+32767), rounding floating point numbers away from zero. Fails if the parameter is a string.

Entry conditions:

None.
(But it is assumed that the parameter has not been altered)

Exit conditions:

`HL` contains the Integer parameter
Other registers and flags corrupt

`RETURN INTEGER`

Set the value returned by the function to be the Integer given.

Entry conditions:

`HL` contains the Integer value to be returned

Exit conditions:

Register A and flags corrupt
Other registers preserved

The addresses of these two routines appear within the BASIC interpreter at the following locations :

`GET INTEGER`	259 Decimal	103 Hexadecimal
`RETURN INTEGER`	261 Decimal	105 Hexadecimal

3. CALLed Subroutines.

The `CALL` command specifies the address of the subroutine, optionally followed by a list of parameters. The address may only be specified by the value of a variable - ie not by an expression. The parameters may only be variables - they may not be expressions. (Note that this includes variables which are array items.)

Parameters are passed by reference, that is the address of the value of each parameter is passed (in the case of a String, the value is the String Descriptor). The number and type of parameters must be agreed between the BASIC program and the subroutine - there is no checking. If possible the addresses are passed in registers, otherwise they are passed in an area of memory.

The entry conditions are:

If there are 3 or less parameters:

HL contains the address of parameter 1 (if any)
DE contains the address of parameter 2 (if any)
BC contains the address of parameter 3 (if any)

If there are 4 or more parameters:

HL contains the address of parameter 1
DE contains the address of parameter 2
BC contains the address of an area of memory containing the addresses of the other parameters, thus:

address of parameter 3	low address
address of parameter 4	
. . . .	
address of parameter n	high address

The contents of other registers are undefined.

The exit conditions are:

All registers and flags corrupt.

Since parameters are passed by reference it is possible for the subroutine to return values. New String values may be returned provided that the subroutine does NOT alter the String Descriptor IN ANY WAY.

Appendix D
BASIC Keywords

The following are the BASIC keywords. They are therefore reserved and cannot be used as variable names. (Note that this list includes keywords for commands not implemented in Run-Only Professional BASIC, this is to ensure that compatability is maintained with the fully interpretive version of Professional BASIC.

```
ABS, ALL, AND, AS, ASC, ATN

BASE

CALL, CDBL, CHAIN, CHR$, CINT, CLEAR,
CLOSE, COMMON, COS, CSNG, CVD, CVI, CVS

DATA, DEC$, DEF, DEFDBL, DEFINT, DEF SEG,
DEFSNG, DEFSTR, DIM, DIR, DISPLAY

ELSE, END, EOF, EQV, ERA, ERASE,
ERL, ERR, ERROR, EXP

FIELD, FILES, FIND$, FIX, FN, FOR, FRE

GET, GOSUB, GOTO

HEX$, HIMEM

IF, IMP, INKEY$, INP, INPUT, INPUT #,
INPUT$, INPW, INSTR, INT

KILL

LEFT$, LEN, LET, LINE, LOAD, LOC,
LOF, LOG, LOG10, LOWER$, LPOS, LPRINT, LSET

MAX, MEMORY, MID$, MIN, MKD$, MKI$, MKS$, MOD

NAME, NEXT, NOT

OCT$, ON, ON ERROR GOTO 0, OPEN, OPTION,
OR, OUT, OUTW
```

PEEK, POKE, POS, PRINT, PRINT #, PUT

RANDOMIZE, READ, REM, REN, RESET, RESTORE,
RESUME, RESUME Ø, RETURN, RIGHT$, RND,
ROUND, RSET, RUN

SAVE, SGN, SIN, SPACE$, SPC, SQR, STEP, STOP,
STR$, STRING$, STRIP$, SWAP, SYSTEM

TAB, TAN, THEN, TO, TYPE

UNT, UPPER$, USING, USR, VAL, VARPTR

WAIT, WAITW, WEND, WHILE, WIDTH, WRITE, WRITE #

XOR

ZONE

Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, England

Printed by Saunders & Williams (Printers) Ltd, Croydon, Surrey