98072

Microsoft® BASIC

Reference Manual

Part 1

Microsoft Corporation

Contents

Chapter 1

General Information about Microsoft BASIC

1.1 Initialization

The procedure for initialization will vary with different implementations of Microsoft BASIC. Check the *Microsoft BASIC User's Guide* for your machine to determine how Microsoft BASIC is initialized with your operating system.

1.2 Modes of Operation

When Microsoft BASIC is initialized, it displays the prompt "Ok". "Ok" indicates Microsoft BASIC is at command level; that is, it is ready to accept commands. At this point, Microsoft BASIC may be used in either of two modes: direct mode or indirect mode.

In direct mode, Microsoft BASIC statements and commands are not preceded by line numbers. They are executed as they are entered. Results of arithmetic and logical operations may be displayed immediately and stored for later use, but the instructions themselves are lost after execution. Direct mode is useful for debugging and for using Microsoft BASIC as a "calculator" for quick computations that do not require a complete program.

Indirect mode is used for entering programs. Program lines are preceded by line numbers and are stored in memory. The program stored in memory is executed by entering the RUN command.

1.3 Line Format

Microsoft BASIC program lines have the following format (square brackets indicate optional input):

```
nnnnn BASIC statement[:BASIC statement...] <carriage return>
```

More than one BASIC statement may be placed on a line, but each must be separated from the last by a colon.

A Microsoft BASIC program line always begins with a line number and ends with a carriage return. A line may contain a maximum of 255 characters.

It is possible to extend a logical line over more than one physical line by using the <line feed> key. <line feed> lets you continue typing a logical line on the next physical line without entering a <carriage return>.

1.3.1 Line Numbers

Every Microsoft BASIC program line begins with a line number. Line numbers indicate the order in which the program lines are stored in memory. Line numbers are also used as references in branching and editing. Line numbers must be in the range 0 to 65529.

A period (.) may be used in EDIT, LIST, AUTO, and DELETE commands to refer to the current line.

1.4 Character Set

The Microsoft BASIC character set is comprised of alphabetic characters, numeric characters, and special characters.

The alphabetic characters in Microsoft BASIC are the uppercase and lowercase letters of the alphabet.

The Microsoft BASIC numeric characters include the digits 0 through 9.

In addition, the following special characters and terminal keys are recognized by Microsoft BASIC:

Character	Action
	Blank
=	Equals sign or assignment symbol
+	Plus sign
–	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or exponentiation symbol
(	Left parenthesis
)	Right parenthesis
%	Percent
#	Number (or pound) sign
$	Dollar sign
!	Exclamation point
[	Left bracket
]	Right bracket
,	Comma
.	Period or decimal point
;	Semicolon
:	Colon
&	Ampersand
'	Single quotation mark (apostrophe)
?	Question mark
<	Less than
>	Greater than
\	Backslash or integer division symbol
@	At sign
_	Underscore
<rubout>	Deletes last character typed.
<escape>	Escapes edit mode subcommands. See Section 2.16.
<tab>	Moves print position to next tab stop. Tab stops are set every eight columns.
<line feed>	Moves to next physical line.
<carriage return>	Terminates input of a line.

1.4.1 Control Characters

Microsoft BASIC supports the following control characters:

Control Character	Action
Control-A	Enters edit mode on the line being typed.
Control-C	Interrupts program execution and returns to BASIC command level.
Control-G	Rings the bell at the terminal.
Control-H	Backspaces. Deletes the last character typed.
Control-I	Tabs to the next tab stop. Tab stops are set every eight columns.
Control-O	Halts program output while execution continues. A second Control-O resumes output.
Control-R	Lists the line that is currently being typed.
Control-S	Suspends program execution.
Control-Q	Resumes program execution after a Control-S.
Control-U	Deletes the line that is currently being typed.

1.5 Constants

Constants are the values Microsoft BASIC uses during execution. There are two types of constants: string and numeric.

A string constant is a sequence of up to 255 alphanumeric characters enclosed in double quotation marks.

Examples

```
"HELLO"
"$25,000.00"
"Number of Employees"
```

Numeric constants are positive or negative numbers. Microsoft BASIC numeric constants cannot contain commas. There are five types of numeric constants:

1. Integer constants — Whole numbers between -32768 and 32767. Integer constants do not contain decimal points.

2. Fixed-point constants — Positive or negative real numbers, i.e., numbers that contain decimal points.

3. Floating-point constants — Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant consists of an optionally signed integer or fixed-point number (the mantissa) followed by the letter E and an optionally signed integer (the exponent). The allowable range for floating-point constants is 10^{-38} to 10^{+38}.

   Examples

   ```
   235.988E-7 = .0000235988
   2359E6 = 2359000000
   ```

   (Double precision floating-point constants are denoted by the letter D instead of E. See Section 1.5.1.)

4. Hex constants — Hexadecimal numbers, denoted by the prefix &H.

   Examples

   ```
   &H76
   &H32F
   ```

5. Octal constants — Octal numbers, denoted by the prefix &O or &.

Examples

```
&O347
&1234
```

Note

The 8K version of Microsoft BASIC does not support hexadecimal or octal constants.

1.5.1 Single and Double Precision Form for Numeric Constants

Numeric constants may be either single precision or double precision numbers. Single precision numeric constants are stored with 7 digits of precision, and printed with up to 6 digits of precision. Double precision numeric constants are stored with 16 digits of precision and printed with up to 16 digits.

A single precision constant is any numeric constant that has one of the following characteristics:

1. Seven or fewer digits.
2. Exponential form using E.
3. A trailing exclamation point (!).

Examples

```
46.8
-1.09E-06
3489.0
22.5!
```

A double precision constant is any numeric constant that has one of these characteristics:

1. Eight or more digits.
2. Exponential form using D.
3. A trailing number sign (#).

Examples

```
345692811
-1.09432D-06
3489.0#
7654321.1234
```

1.6 Variables

Variables are names used to represent values used in a BASIC program. The value of a variable may be assigned explicitly by the programmer, or it may be assigned as the result of calculations in the program. Before a variable is assigned a value, its value is assumed to be zero.

1.6.1 Variable Names and Declaration Characters

Microsoft BASIC variable names may be any length. Up to 40 characters are significant. Variable names can contain letters, numbers, and the decimal point. However, the first character must be a letter. Special type declaration characters are also allowed—see below.

A variable name may not be a reserved word, but embedded reserved words are allowed. Reserved words include all Microsoft BASIC commands, statements, function names, and operator names. If a variable begins with FN, it is assumed to be a call to a user-defined function.

Variables may represent either a numeric value or a string. String variable names are written with a dollar sign ($) as the last char-

acter. For example: A$ = "SALES REPORT." The dollar sign is a variable type declaration character; that is, it "declares" that the variable will represent a string.

Numeric variable names may declare integer, single precision, or double precision values. The type declaration characters for these variable names are as follows:

%	Integer variable
!	Single precision variable
#	Double precision variable

The default type for a numeric variable name is single precision.

Examples of Microsoft BASIC variable names:

PI#	Declares a double precision value.
MINIMUM!	Declares a single precision value.
LIMIT%	Declares an integer value.
N$	Declares a string value.
ABC	Represents a single precision value.

There is a second method by which variable types may be declared. The Microsoft BASIC statements DEFINT, DEFSTR, DEFSNG, and DEFDBL may be included in a program to declare the types for certain variable names. These statements are described in detail in Section 2.12.

1.6.2 Array Variables

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. An array variable name has as many subscripts as there are dimensions in the array. For example, V(10) would reference a value in a one-dimension array, T(1,4) would reference a value in a two-dimension array, and so on. The maximum number of dimensions for an array is 255. The maximum number of elements per dimension is 32,767.

1.6.3 Space Requirements

The following table lists only the number of bytes occupied by the values represented by the variable names. Additional requirements may vary according to implementation.

Variables	Type	Bytes
	Integer	2
	Single Precision	4
	Double Precision	8
Arrays	**Type**	**Bytes**
	Integer	2 per element
	Single Precision	4 per element
	Double Precision	8 per element
Strings		
3 bytes overhead plus the present contents of the string.		

1.7 Type Conversion

When necessary, Microsoft BASIC will convert a numeric constant from one type to another. The following rules and examples should be kept in mind.

1. If a numeric constant of one type is set equal to a numeric variable of a different type, the number will be stored as the type declared in the variable name. (If a string variable is set equal to a numeric value or vice versa, a "Type mismatch" error occurs.)

   Example

```
10 A% = 23.42
20 PRINT A%
RUN
 23
```

2. During expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision, i.e., that of the most precise operand. Also, the result of an arithmetic operation is returned to this degree of precision.

   Examples

```
10 D# = 6#/7
20 PRINT D#
RUN
 .8571428571428571
```

The arithmetic was performed in double precision and the result was returned in D# as a double precision value.

```
10 D = 6#/7
20 PRINT D
RUN
 .857143
```

The arithmetic was performed in double precision and the result was returned to D (single precision variable), rounded, and printed as a single precision value.

3. Logical operators (see Section 1.8.3) convert their operands to integers and return an integer result. Operands must be in the range -32768 to 32767 or an "Overflow" error occurs.

4. When a floating-point value is converted to an integer, the fractional portion is rounded.

   Example

```
10 C% = 55.88
20 PRINT C%
RUN
 56
```

5. If a double precision variable is assigned a single precision value, only the first seven digits (rounded) of the converted number will be valid. This is because only seven digits of accuracy were supplied with the single precision value. The absolute value of the difference between the printed double precision number and the original single precision value will be less than 6.3E-8 times the original single precision value.

   Example

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
RUN
 2.04 2.039999961853027
```

1.8 Expressions and Operators

An expression may be a string or numeric constant, a variable, or a combination of constants and variables with operators which produces a single value.

Operators perform mathematical or logical operations on values. The Microsoft BASIC operators may be divided into four categories:

1. Arithmetic
2. Relational
3. Logical
4. Functional

Each category is described in the following sections.

1.8.1 Arithmetic Operators

The arithmetic operators, in order of precedence, are:

Operator	Operation	Expression
^	Exponentiation	X^Y
-	Negation	-X
*,/	Multiplication, Floating-point Division	X*Y X/Y
+,-	Addition, Subtraction	X+Y

To change the order in which the operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operations is maintained.

Here are some sample algebraic expressions and their Microsoft BASIC counterparts.

Algebraic Expression	BASIC Expression	
X+2Y	X+Y*2	
$X-\frac{Y}{Z}$	X-Y/Z	
$\frac{XY}{Z}$	X*Y/Z	
$\frac{X+Y}{Z}$	(X+Y)/Z	
$(X^2)^Y$	(X^2)^Y	
X^{Y^Z}	X^(Y^Z)	
X(-Y)	X*(-Y)	Consecutive operators are separated by parentheses.

1.8.1.1 Integer Division and Modulus Arithmetic

Two additional operators are available in Microsoft BASIC: integer division and modulus arithmetic.

Integer division is denoted by the backslash (\). The operands are rounded to integers (must be in the range -32768 to 32767) before the division is performed, and the quotient is truncated to an integer.

Example

```
10\4 = 2
25.68\6.99 = 3
```

Integer division follows multiplication and floating-point division in order of precedence.

Modulus arithmetic is denoted by the operator MOD. Modulus arithmetic yields the integer value that is the remainder of an integer division.

Example

```
10.4 MOD 4=2          (10/4=2 with a remainder 2)
25.68 MOD 6.99=5      (26/7=3 with a remainder 5)
```

Modulus arithmetic follows integer division in order of precedence.

1.8.1.2 Overflow and Division by Zero

If, during the evaluation of an expression, division by zero is encountered, the "Division by zero" error message is displayed, machine infinity with the sign of the numerator is supplied as the result of the division, and execution continues. If the evaluation of an exponentiation operator results in zero being raised to a negative power, the "Division by zero" error message is displayed, positive machine infinity is supplied as the result of the exponentiation, and execution continues.

If overflow occurs, the "Overflow" error message is displayed, machine infinity with the algebraically correct sign is supplied as the result, and execution continues.

1.8.2 Relational Operators

Relational operators are used to compare two values. The result of the comparison is either "true" (-1) or "false" (0). This result may then be used to make a decision regarding program flow. (See "IF" statements, Section 2.26.)

The relational operators are:

Operator	Relation Tested	Example
=	Equality	X=Y
< >	Inequality	X< >Y
<	Less than	X<Y
>	Greater than	X>Y
<=	Less than or equal to	X<=Y
>=	Greater than or equal to	X>=Y

(The equal sign is also used to assign a value to a variable. See "LET," Section 2.30.)

When arithmetic and relational operators are combined in one expression, the arithmetic is always performed first. For example, the expression

```
X+Y<(T-1)/Z
```

is true if the value of X plus Y is less than the value of T-1 divided by Z.

Examples

```
IF SIN(X)<0 GOTO 1000
IF I MOD J<>0 THEN K=K+1
```

1.8.3 Logical Operators

Logical operators perform tests on multiple relations, bit manipulation, or Boolean operations. The logical operator returns a bitwise result which is either "true" (not zero) or "false" (zero). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined as shown in Table 1. The operators are listed in order of precedence.

Table 1. Microsoft BASIC Relational Operators Truth Table

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

Just as the relational operators can be used to make decisions regarding program flow, logical operators can connect two or more relations and return a true or false value to be used in a decision (see "IF" statements, Section 2.26).

Example

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
```

Logical operators work by converting their operands to 16-bit, signed, two's complement integers in the range -32768 to 32767. (If the operands are not in this range, an error results.) If both operands are supplied as 0 or -1, logical operators return 0 or -1. The given operation is performed on these integers in bitwise fashion, i.e., each bit of the result is determined by the corresponding bits in the two operands.

Thus, it is possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator may be used to "mask" all but one of the bits of a status byte at a machine I/O port. The OR operator may be used to "merge" two bytes to create a particular binary value. The following examples will help demonstrate how the logical operators work.

63 AND 16=16	63=binary 111111 and 16=binary 10000, so 63 AND 16=16.
15 AND 14=14	15=binary 1111 and 14=binary 1110, so 15 AND 14=14 (binary 1110).
-1 AND 8=8	-1=binary 1111111111111111 and 8=binary 1000, so -1 AND 8=8.
4 OR 2=6	4=binary 100 and 2=binary 10, so 4 OR 2=6 (binary 110).
10 OR 10=10	10=binary 1010, so 1010 OR 1010=1010 (decimal 10).
-1 OR -2=-1	-1=binary 1111111111111111 and -2=binary 1111111111111110, so -1 OR -2=-1. The bit complement of sixteen zeros is sixteen ones, which is the two's complement representation of -1.
NOT X=-(X+1)	The two's complement of any integer is the bit complement plus one.

1.8.4 Functional Operators

A function is used in an expression to call a predetermined operation that is to be performed on an operand. Microsoft BASIC has "intrinsic" functions that reside in the system, such as SQR (square root) or SIN (sine). All Microsoft BASIC intrinsic functions are described in Chapter 3.

Microsoft BASIC also allows "user-defined" functions that are written by the programmer. See "DEF FN," Section 2.11.

1.8.5 String Operators

Strings may be concatenated by using +.

Example

```
10 A$ = "FILE" : B$ = "NAME"
20 PRINT A$ + B$
30 PRINT "NEW" + A$ + B$
RUN
FILENAME
NEW FILENAME
```

Strings may be compared using the same relational operators that are used with numbers:

```
=    <>    <    >    <=    >=
```

String comparisons are made by taking one character at a time from each string and comparing the ASCII codes. If all the ASCII codes are the same, the strings are equal. If the ASCII codes differ, the lower code number precedes the higher. If during string comparison the end of one string is reached, the shorter string is said to be smaller. Leading and trailing blanks are significant.

Examples

```
"AA"<"AB"
"FILENAME" = "FILENAME"
"X&">"X#"
"CL">"CL"
"kg">"KG"
"SMYTH"<"SMYTHE"
B$<"9/12/78"  where
B$ = "8/12/78"
```

Thus, string comparisons can be used to test string values or to alphabetize strings. All string constants used in comparison expressions must be enclosed in quotation marks.

1.9 Input Editing

If an incorrect character is entered as a line is being typed, it can be deleted with the <RUBOUT> key or with Control-H. Rubout surrounds the deleted character(s) with backslashes. Control-H has the effect of backspacing over a character and erasing it. Once a character(s) has been deleted, simply continue typing the line as desired.

To delete a line that is in the process of being typed, type Control-U. A carriage return is executed automatically after the line is deleted.

To correct program lines for a program that is currently in memory, simply retype the line using the same line number. Microsoft BASIC will automatically replace the old line with the new line.

More sophisticated editing capabilities are provided. See "EDIT," Section 2.16.

To delete the entire program currently residing in memory, enter the NEW command. (See Section 2.41.) NEW is usually used to clear memory prior to entering a new program.

1.10 Error Messages

If an error causes program execution to terminate, an error message is printed. For a complete list of Microsoft BASIC error codes and error messages, see Appendix A.

Chapter 2

Microsoft BASIC Commands and Statements

Microsoft BASIC commands and statements are described in this chapter. Each description is formatted as follows:

Syntax Shows the correct syntax for the instruction. See the "Introduction" to this manual for syntax notation.

Purpose Tells what the instruction is used for.

Remarks Describes in detail how the instruction is used.

Example Shows sample programs or program segments that demonstrate the use of the instruction.

Note Describes special cases or provides additional pertinent information.

2.1 AUTO

Syntax AUTO [<line number>[,<increment>]]

Purpose To generate a line number automatically after every carriage return.

Remarks AUTO begins numbering at <line number> and increments each subsequent line number by <increment>. The default for both values is 10. If <line number> is followed by a comma but <increment> is not specified, the last increment specified in an AUTO command is assumed.

If AUTO generates a line number that is already being used, an asterisk is printed after the number to warn the user that any input will replace the existing line. However, typing a carriage return immediately after the asterisk will save the line and generate the next line number.

AUTO is terminated by typing Control-C. The line in which Control-C is typed is not saved. After Control-C is typed, Microsoft BASIC returns to command level.

Example

```
AUTO 100,50     Generates line numbers
                100, 150, 200 ....
AUTO            Generates line numbers 10, 20,
                30, 40 ....
```

2.2 CALL

Syntax

CALL <variable name>[(<argument list>)]

Purpose

To call an assembly language subroutine.

Remarks

The CALL statement is one way to transfer program flow to an external subroutine. (See also the USR function, Section 3.41)

<variable name> contains an address that is the starting point in memory of the subroutine. <variable name> may not be an array variable name. <argument list> contains the arguments that are passed to the external subroutine. <argument list> may contain only variables.

The CALL statement generates the same calling sequence used by Microsoft FORTRAN, Microsoft COBOL, and Microsoft BASIC Compilers.

Example

```
110 MYROUT = &HD000
120 CALL MYROUT(I,J,K)
.
.
.
```

Note

In a Microsoft BASIC Compiler program, line 110 is not required because the address of MYROUT will be assigned by the linking loader at load time.

2.3 CHAIN

Syntax

CHAIN [MERGE]<filename>[,[<line number exp>] [,ALL][,DELETE <range>]]

Purpose

To call a program and pass variables to it from the current program.

Remarks

<filename> is the name of the program that is called.

<line number exp> is a line number or an expression that evaluates to a line number in the called program. It is the starting point for execution of the called program. If it is omitted, execution begins at the first line.

The COMMON statement may be used to pass variables (see Section 2.7).

Example 1

```
10  REM THIS PROGRAM DEMONSTRATES CHAIN-
ING USING COMMON TO PASS VARIABLES.
20  REM SAVE THIS MODULE ON DISK AS "PROG1"
USING THE A OPTION.
30  DIM A$(2),B$(2)
40  COMMON A$(),B$()
50  A$(1)="VARIABLES IN COMMON MUST BE
ASSIGNED"
60  A$(2)="VALUES BEFORE CHAINING."
70  B$(1)=" ": B$(2) =" "
80  CHAIN "PROG2"
90  PRINT: PRINT B$(1): PRINT: PRINT B$(2):
PRINT
100 END
```

Example 2

```
10  REM THE STATEMENT "DIM A$(2),B$(2)" MAY
ONLY BE EXECUTED ONCE.
20  REM HENCE, IT DOES NOT APPEAR IN THIS
MODULE.
30  REM SAVE THIS MODULE ON THE DISK AS
"PROG2" USING THE A OPTION.
40  COMMON A$(),B$()
50  PRINT: PRINT A$(1);A$(2)
60  B$(1) = "NOTE HOW THE OPTION OF SPECIFY-
ING A STARTING LINE NUMBER"
70  B$(2) = "WHEN CHAINING AVOIDS THE DIMEN-
SION STATEMENT IN 'PROG1'."
80  CHAIN "PROG1",90
90  END
```

<line number exp> is not affected by a RENUM command.

With the ALL option, every variable in the current program is passed to the called program. If the ALL option is omitted, the current program must contain a COMMON statement to list the variables that are passed. See Section 2.7.

The MERGE option allows a subroutine to be brought into the BASIC program as an overlay. That is, a MERGE operation is performed with the current program and the called program. The called program must be an ASCII file if it is to be MERGEd.

After an overlay is brought in, it is usually desirable to delete it so that a new overlay may be brought in. To do this, use the DELETE option.

Example 3

```
10  REM THIS PROGRAM DEMONSTRATES CHAIN-
ING USING THE MERGE AND ALL OPTIONS.
20  REM SAVE THIS MODULE ON THE DISK AS
"MAINPRG".
30  A$ = "MAINPRG"
40  CHAIN MERGE "OVRLAY1",1010,ALL
50  END
```

```
1000  REM SAVE THIS MODULE ON THE DISK AS
"OVRLAY1" USING THE A OPTION.
1010  PRINT A$; "HAS CHAINED TO OVRLAY1."
1020  A$ = "OVRLAY1"
1030  B$ = "OVRLAY2"
1040  CHAIN MERGE "OVRLAY2",1010,ALL,
DELETE 1000-1050
1050  END
```

```
1000  REM SAVE THIS MODULE ON THE DISK AS
"OVRLAY2" USING THE A OPTION.
1010  PRINT A$;  "HAS CHAINED TO ";B$:"."
1020  END
```

The line numbers in <range> are affected by the RENUM command.

Note

The CHAIN statement with MERGE option leaves the files open and preserves the current OPTION BASE setting.

If the MERGE option is omitted, CHAIN does not preserve variable types or user-defined functions for use by the chained program. That is, any DEFINT, DEFSNG, DEFDBL, DEFSTR, or DEF FN statements containing shared variables must be restated in the chained program.

The Microsoft BASIC Compiler does not support the ALL, MERGE, DELETE, and <line number exp> options to CHAIN. Thus, the statement syntax is CHAIN <filename>. If you wish to maintain compatibility with Microsoft BASIC Compiler, it is recommended that COMMON be used to pass variables and that overlays not be used. The CHAIN statement leaves the files open during CHAINing.

When using the MERGE option, user-defined functions should be placed before any CHAIN MERGE statements in the program. Otherwise, the user-defined functions will be undefined after the merge is complete.

2.4 CLEAR

Syntax CLEAR [,[<expression1>][,<expression2>]]

Purpose To set all numeric variables to zero, all string variables to null, and to close all open files; and, optionally, to set the end of memory and the amount of stack space.

Remarks <expression1> is a memory location which, if specified, sets the highest location available for use by Microsoft BASIC.

<expression2> sets aside stack space for Microsoft BASIC. The default is 512 bytes or one-eighth of the available memory, whichever is smaller.

Note Microsoft BASIC allocates string space dynamically. An "Out of string space" error occurs only if there is no free memory left for Microsoft BASIC to use.

Microsoft BASIC Compiler supports the CLEAR statement with the restriction that <expression1> and <expression2> must be integer expressions. If a value of 0 is given for either expression, the appropriate default is used. The default stack size is 512 bytes, and the default top of memory is the current top of memory. The CLEAR statement performs the following actions:

Closes all files.
Clears all COMMON and user variables.
Resets the stack and string space.
Releases all disk buffers.

Examples

```
CLEAR
CLEAR ,32768
CLEAR ,,2000
CLEAR ,32768,2000
```

2.5 CLOAD

Syntax

CLOAD <filename>
CLOAD? <filename>
CLOAD* <array name>

Purpose

To load a program or an array from cassette tape into memory.

Remarks

CLOAD executes a NEW command before it loads the program from cassette tape. <filename> is the string expression or the first character of the string expression that was specified when the program was CSAVEd.

CLOAD? verifies tapes by comparing the program currently in memory with the file on tape that has the same filename. If they are the same, Microsoft BASIC prints "Ok". If not, Microsoft BASIC prints "NO GOOD".

CLOAD* loads a numeric array that has been saved on tape. The data on tape is loaded into the array called <array name> specified when the array was CSAVE*ed.

CLOAD and CLOAD? are always entered at command level as direct mode commands. CLOAD* may be entered at command level or used as a program statement. Make sure the array has been DIMensioned before it is loaded. Microsoft BASIC always returns to command level after a CLOAD, CLOAD?, or CLOAD* is executed. Before a CLOAD is executed, make sure the cassette recorder is properly connected and in the play mode, and the tape is positioned correctly.

See also "CSAVE," Section 2.9.

Note CLOAD and CSAVE are not included in all implementations of Microsoft BASIC.

Example

```
CLOAD "MAX2"
```

Loads file "MAX2" into memory.

2.6 CLOSE

Syntax

```
CLOSE [[#]<file number>[,[#]<file number...>]]
```

Purpose To conclude I/O to a disk file.

Remarks <file number> is the number under which the file was OPENed. A CLOSE with no arguments closes all open files.

The association between a particular file and file number terminates upon execution of a CLOSE statement. The file may then be reOPENed using the same or a different file number; likewise, that file number may now be reused to OPEN any file.

A CLOSE for a sequential output file writes the final buffer of output.

The END statement and the NEW command always CLOSE all disk files automatically. (STOP does not close disk files.)

Example See "Disk File Handling," in the *Microsoft BASIC User's Guide.*

2.7 COMMON

Syntax COMMON <list of variables>

Purpose To pass variables to a CHAINed program.

Remarks The COMMON statement is used in conjunction with the CHAIN statement. COMMON statements may appear anywhere in a program, though it is recommended that they appear at the beginning. The same variable cannot appear in more than one COMMON statement. Array variables are specified by appending "( )" to the variable name. If all variables are to be passed, use CHAIN with the ALL option and omit the COMMON statement.

Example

```
100 COMMON A,B,C,D(),G$
110 CHAIN "PROG3",10
 .
 .
 .
```

Note Microsoft BASIC Compiler supports a modified version of the COMMON statement. The COMMON statement must appear in a program before any executable statements. The current nonexecutable statements are:

```
COMMON
DEFDBL, DEFINT, DEFSNG, DEFSTR
DIM
OPTION BASE
REM
$INCLUDE
```

Array variables used in a COMMON statement must be declared in a preceding DIM statement.