Chapter 2 PROGRAMMING CONCEPTS

This chapter discusses a variety of concepts which are applicable to programming in MFBASIC. These are presented in logical order, with the most fundamental concepts presented first. Mastery of this information is essential to realizing the full potential capabilities of MFBASIC.

2.1 Program Lines and Statements

All MFBASIC programs are composed of one or more lines, each of which begins with a line number, ends with a RETURN code, and includes one or more commands or statements. Line numbers indicate the order in which program lines are stored in memory and executed, and are referenced when editing a program or changing the flow of program execution with the GOTO or GOSUB statements. Line numbers must be within the range from 0 to 65529. With MFBASIC, a period, or full stop, (".") may be used in the EDIT, LIST, AUTO and DELETE commands to indicate the last line currently contained in memory.

Commands and statements are words in the MFBASIC language which instruct EPSON MFBASIC to perform specific operations. Any number of statements may be included in one program line by separating them with colons; however, the maximum length of a program line is 255 1-byte characters.

2.2 ASCII Characters

The character generator of the QX-10 includes ASCII character sets for the following eight countries.

1. United States
2. England
3. Germany
4. France
5. Italy
6. Spain
7. Denmark
8. Sweden

These character sets can be selected by means of the OPTION COUNTRY command, which also changes assignment of letters to the keyboard to that of the character set for the specified country. Each character of these character sets is identified by a 1-byte ASCII code; these codes are shown in Appendix H, and the keyboard arrangements for each of the eight option countries are shown in Appendix G. Differences in ASCII code assignments for the eight option countries are shown in the table below.

Country Dec. code	U.S.A.	France	Germany	England	Denmark	Sweden	Italy	Spain
35 (23)	#	#	#	£	#	#	#	Pt
36 (24)	$	$	$	$	$	¤	$	$
64 (40)	@	à	§	@	@	É	@	@
91 (5B)	[	°	Ä	[	Æ	Ä	°	¡
92 (5C)	\	ç	Ö	\	Ø	Ö	\	Ñ
93 (5D)	[	§	Ü	]	Å	Å	é	¿
94 (5E)	∧	∧	∧	∧	∧	Ü	∧	∧
96 (60)	‘	‘	‘	‘	‘	é	ù	‘
123 (7B)	{	é	ä	{	æ	ä ˜	à	¨
124 (7C)	¦	ù	ö	¦	ø	ö	ò	ñ
125 (7D)	}	è	ü	}	å	å	è	}
126 (7E)	~	¨	ß	~	~	ü	ì	~

NOTES:

1. *Figures in parentheses are hexadecimal codes.*
2. *Blanks are displayed for some character codes when option style 15 or 16 is selected, or when character style &H0F is selected with the style selection keys.*

2.3 Multiple Font Characters

In addition to the ASCII characters described above, MultiFonts BASIC provides 16 other character fonts which are internally represented in 2-byte code. Examples of MultiFonts characters are shown below. The character sets are illustrated in full in Appendix J.

```
OCR B-FONT
BODONI
OLD ENGLISH
FLASH BOLD
COMMERCIAL SCRIPT
HELVETICA LIGHT
HELVETICA LIGHT ITALIC
HELVETICA MEDIUM ITALIC
BROADWAY
AMERICAN TYPEWRITER MEDIUM
LIGHT ITALIC
HELVETICA MEDIUM
BODONI ITALIC
SANS SERIF SHADED
MICROGRAMA EXTENDED
OLD GERMAN
```

Of the 16 styles, 15 can be input directly from the keyboard by means of the style selection keys at the top right of the keyboard. Each font is numbered as shown in Appendix I, and the settings of the style selection keys correspond to these numbers as follows.

Assuming that each key corresponds to a logical "1" when ON and to a logical "0" when OFF, the settings of the keys correspond to the binary equivalent of the style number as shown below.

				Hex.	Decimal
OFF	OFF	OFF	ON		
SF1	SF2	SF3	SF4	01	1
ON	OFF	ON	OFF		
SF1	SF2	SF3	SF4	0A	10
ON	ON	ON	ON		
SF1	SF2	SF3	SF4	0F	15

MultiFonts characters are provided in two groups, one with codes which range from &HA0A0 to &HAFFF, and another with codes which range from &HB0A0 to &HCFDF. The former group includes characters which correspond to the OPTION COUNTRY character sets; these can be selected using the OPTION COUNTRY and OPTION STYLE commands, the STYLE$ function, or the style selection keys at the top right of the keyboard. The characters in the latter system are not related to any of these, but can be output to the display or printer by direct specification of their 2-byte codes.

Example

```
USASCII
A1C1 = A     ADC0 = @
France
A1C1 = A     ADC0 = à
```

2.4 Control characters

The ASCII character set includes a number of special codes which can be used in programs together with PRINT CHR$ or LPRINT CHR$ for control of the screen or printer. These control codes are used as described below.

2.4.1 Screen control codes

PRINT CHR$(5);
This sequence erases the screen from the current position of the cursor to the end of the line (same as CTRL and E).

PRINT CHR$(7);
This sequence sounds the speaker built into the QX-10.

PRINT CHR$(8);
This sequence moves the cursor one column to the left (same as the BS key or CTRL and H).

PRINT CHR$(9);
This sequence moves the cursor to the next tab position on the screen (same as the TAB key or CTRL and M).

PRINT CHR$(10);
This sequence moves the cursor to the next line on the screen.

PRINT CHR$(11);
This sequence moves the cursor to the upper left corner of the screen (same as the HOME key or CTRL and K).

PRINT CHR$(12);
This sequence clears the display screen (same as the CLS key or CTRL and L).

PRINT CHR$(13);
This sequence performs the same function as the RETURN key or CTRL and M).

PRINT CHR$(26);
This sequence erases the screen from the current cursor position to the end of the screen.

PRINT CHR$(27);
This sequence makes it possible to enter special escape codes for screen control. See Appendix C for the escape codes which can be used with MFBASIC.

PRINT CHR$(28);
This sequence moves the cursor one column to the right (same as the →| key).

PRINT CHR\$(29);
This sequence moves the cursor one position to the left (same as the ⌐←⌐ key).

PRINT CHR\$(30);
This sequence moves the cursor up to line preceding that on which it is currently located (same as the [↑] key).

PRINT CHR\$(31);
This sequence moves the cursor down to the line following that on which it is currently located (same as the [↓] key).

2.4.2 Printer control codes

LPRINT CHR\$(7);
This sequence sounds the buzzer built into the printer.

LPRINT CHR\$(10);
This sequence advances the paper by one line.

LPRINT CHR\$(12);
This sequence advances the paper to the top of the next page.

LPRINT CHR\$(13);
This sequence returns the print head to the left side of the paper (without making a line feed).

LPRINT CHR\$(27);
This sequence makes it possible to enter special escape codes for printer control. See Appendix C for the escape codes which can be used with MFBASIC.

2.5   Constants

Constants are fixed values which are written into a program and which are used by the program during its execution. These values may consist of either characters or numbers; in the former case they are referred to as string constants, and in the latter as numeric constants. A string constant is any sequence of alphanumeric characters which is enclosed in quotation marks. Some examples of string constants are shown below.

```
"HELLO"
"My name is Ralph. What's your's?"
"$25,000.00"
"The quick brown fox jumped over the lazy yellow dog."
```

The length of a string constant cannot exceed the maximum length of a program line (255 bytes, or 255 1-byte characters).

Numeric constants are positive or negative numbers. There are five types of numeric constants as follows.

(1)   Integer constants

Integer constants are whole numbers in the range from -32768 to +32767.

(2)   Fixed point constants

Fixed point constants are positive or negative real numbers which include a decimal fraction.

(3)   Floating point constants

Floating point constants are positive or negative numbers which are represented in exponential form. A floating point constant consists of an integer or fixed point constant, followed by the letter E (denoting an implicit base of 10) and an exponent. Either the fixed-point part or the exponent may be preceded by a minus ("-") or plus ("+") sign to indicate that it is positive or negative; if no sign is present, it is assumed that that portion of the constant is positive. The exponent must be in the range from -38 to +38.

Example $235.988E\text{-}7 = 235.988 \times 10^{-7} = .0000235988$
$2359E6 = 2359 \times 10^{6} = 2359000000$

(With double precision floating point constants, the letter "D" is used to indicate the implicit base (10) instead of "E." See below for a discussion of single and double precision numeric constants.)

(4)　Hexadecimal constants

Hexadecimal constants are numbers which are formed using the numeral set consisting of 0 through 9 and A through F. In this system, the characters A through F are equivalent to the decimal numbers from 10 to 15. Such constants are identified by the prefix "&H". The decimal equivalents of hexadecimal numbers can be calculated as shown below.

Example

$$\&H76 = 7 \times 16^1 + 6 \times 16^0 = 118$$
$$\&H32F = 3 \times 16^2 + 2 \times 16^1 + 15 \times 16^0 = 815$$

(5)　Octal constants

Octal constants are numbers which are formed using the numeral set consisting of 0 through 7. Such constants are identified by the prefix "&" or "&O". The decimal equivalents of octal numbers can be calculated as shown below.

Example

$$\&O347 = 3 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 = 231$$
$$\&1234 = 1 \times 8^3 + 2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 668$$

2.6   Single and Double Precision Numeric Constants

MFBASIC allows use of both single and double precision numbers. Single precision numbers are handled internally as seven significant digits, and are rounded to 6 digits for display or printout. Double precision numbers are handled internally as 16 significant digits, and are also printed or displayed as 16 digits (with leading zeroes suppressed).

A single precision constant is any numeric constant that fulfills one of the following conditions.
(1)   Consists of seven or fewer digits
(2)   Is represented in exponential form with "E", or
(3)   Has a trailing exclamation point (!).

A double precision constant is any numeric constant that fulfills one of the following conditions.
(1)   Consists of eight or more digits
(2)   Is represented in exponential form with "D", or
(3)   Has a trailing number sign (#).

Example

Single Precision Constants	Double Precision Constants
46.8	345692811
-7.09E-06	-1.09432D-06
3489.0	3489.0 #
22.5!	7654321.1234

2.7 Variables

Variables are named locations in memory which are used to hold values during execution of MFBASIC programs. Names are assigned to variables by the programmer, and the values stored in variables are either assigned by the user during program execution or assigned as a result of program execution itself.

The two general types of variables used in MFBASIC are numeric variables and string variables. The former are used to store numeric values and the latter are used to store character strings. The initial values of all variables are 0 or null.

2.7.1 Variable names and type declaration characters

Variable names may consist of up to 40 characters (including all letters, the decimal point, and all numerals), followed by a type declaration character; however, the first character of each name must be a letter. Reserved words may not be used as variable names. (Reserved words are the keywords used in specifying MFBASIC commands and function names.) Further, the letters "FN" must not be used at the beginning of a variable name (MFBASIC will interpret names beginning with FN as calls to a user-defined function).

The names of string variables must end with a dollar sign ($); this is the type declaration character which indicates that a variable is used to hold string data.

Numeric variable names may end with type declaration characters which indicate the type of numeric data which they contain. The type declaration characters for numeric variables are as follows.

      %   Integer variable declaration character
      !   Single precision variable declaration character
      #   Double precision variable declaration character

A single precision numeric variable is assumed if no type declaration character is specified.

Examples of variable names are shown below.

      PI #   Double precision variable
      MINIMUM!   Single precision variable
      LIMIT%   Integer variable
      CATEGORY$ String variable

Variables may be defined in advance as string, integer, single precision, or double precision by means of the DEFINT, DEFSTR, DEFSNG, and DEFDBL statements. When variable types are specified using this method, type declaration characters are not required. See the explanations of these statements in Chapter 3 for details.

2.7.2 Array variables

An array is a group of variables which is referred to by a common name. Each variable of an array is identified by one or more subscripts, each of which is specified as an integer or integer expression. The number of subscripts corresponds to the number of dimensions of the array; thus, P(x) refers to a specific variable in a one-dimensional array (where x is the integer or integer expression which identifies the individual variable), P(x,y) refers to a specific variable in a two-dimensional array, and so forth. A one-dimensional array can be thought of as a one-column list which contains a certain number of items; the number of items depends on the maximum value of x. Likewise, a two-dimensional array can be thought of as a table containing a certain number of rows and columns; the number of rows depends on the maximum value of x, and the number of columns depends on the maximum value of y. Theoretically, an array can have any number of dimensions; however, in practice the number of dimensions and the size of the array are limited by the amount of memory available. The DIM statement is used to define the number of dimensions of an array and the size of each dimension. See the explanation of the DIM statement in Chapter 3 for details.

2.7.3 Type conversion of numeric values

MFBASIC automatically converts numeric values from one type to another as necessary. This section describes the rules governing numeric type conversion for various types of operations.

(1) Type conversion upon storage in variables

If a numeric constant of one type is assigned to a numeric variable of another type, it is stored after being converted to the type declared for that variable name. For example, if an integer type numeric constant is assigned to a single precision variable, it is automatically converted to a single precision value at the time it is stored. Note that a certain amount of error may be introduced by the process of conversion.

Examples

```
10 A%=12.34
20 PRINT A%
RUN
 12
Ok
```

```
10 A#=12.34
20 PRINT A#
RUN
 12.340000152587891
Ok
```

(2) Conversion in arithmetic and relational operations

In an arithmetic or relational expression which includes numeric operands of different types, all operands are converted to the same degree of precision (that of the operand with the highest degree of precision). Further, the results of arithmetic expressions are returned to the degree of precision of the most precise operand. Note that error may be introduced when constants are converted from one precision to another. (Note: Relational operations are described in paragraph 2.8.2 below.)

Example

```
10 A#=6#/7.1#
20 PRINT A#
RUN
 .8450704225352113
Ok
```

In the example above, arithmetic is performed using double precision numbers and the result is returned in double precision.

Example

```
10 A#=6#/7.1
20 PRINT A#
run
 .8450704338862249
Ok
```

Here, the single precision value 7.1 is converted to double precision for arithmetic and the result is returned as a double precision number.

The difference between the result returned in this example and that returned in the preceding example is due to conversion error.

(3) Conversion for logical operations

During logical operations, non-integer operands are converted to integers and the result is returned as an integer. The operands of logical operations must be in the range from -32768 to 32767; otherwise, an "Overflow" error will occur.

2-12

```
10 PRINT 6.34 OR 15
RUN
 15
Ok
```

See paragraph 2.8.3 for a discussion of logical operations.

(4) Type conversion of floating point numbers to integers

When a floating point number is converted to an integer, the decimal fraction is rounded to the nearest whole number.

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

(5) Conversion of single precision numbers to double precision

If a single precision number is assigned to a double precision variable, only the first seven digits of the converted number are significant. This is because only six digits of accuracy are provided by single precision numbers.

```
10 D#=7.12345
20 PRINT D#
RUN
 7.12345027923584
Ok
```

2.8 Expressions and Operations

An expression is any notation within a program which represents a value. Thus, variables and numeric and string constants constitute expressions, either when they appear alone or when combined by operators with other constants or variables.

Operators are symbols which indicate mathematical or logical operations which are to be performed on given values. The types of operations which are performed by MFBASIC may be divided into four categories as follows.
(1) Arithmetic operations
(2) Relational operations
(3) Logical operations
(4) Functional operations

2.8.1 Arithmetic operations

The arithmetic operations performed by MFBASIC include exponentiation, negation, multiplication, division, addition, and subtraction. The precedence of these operations is as shown below.

Operator	Operation	Sample Expression
$\wedge$	Exponentiation	$X \wedge Y$
$-$	Negation (conversion of the sign of a value)	$(-Y)$
$*, /$	Multiplication, division	$X * Y, X/Y$
$\backslash$	Integer division	$X \backslash Y$
MOD	Modulus arithmetic	X MOD Y
$+, -$	Addition, subtraction	$X + Y, X - Y$

The concepts of integer division and modulus arithmetic are explained in (1) and (2) below.

The order in which operations are performed can be changed by including parts of expressions in parentheses according to the normal rules of algebra. When this is done, the operations within parentheses are performed first according to the normal rules of precedence.

Sample algebraic expressions and their equivalents in MFBASIC are shown below.

Algebraic Expression	BASIC Expression
$X + 2Y$	$X + 2 * Y$
$X - Y \div Z$	$X - Y / Z$
$X \times Y \div Z$	$X * Y / Z$
$(X^2)^Y$	$(X \wedge 2) \wedge Y$
$X(Y^2)$	$X \wedge (Y \wedge 2)$
$X \times (-Y)$	$X * (-Y)$

When there are two consecutive operators in an expression, they must be separated by parentheses as shown in the last example above.

(1) Integer division

With integer division, the operands of an expression are rounded to integers, then division is performed and the integer portion of the quotient is returned. The operator for integer division is the backslash (\ ).

Examples PRINT 15\6

   2

Ok

PRINT 17.89\4.32

   4

Ok

When integer division is performed, both operands must be within the range from -32768 to 32767.

(2) Modulus arithmetic

Modulus arithmetic is the arithmetic operation which returns the remainder of integer division as an integer. The operator for modulus arithmetic is MOD. The precedence of modulus arithmetic is just after that of integer division.

Examples PRINT 10.4 MOD 4

   2

Ok

PRINT 25.68 MOD 6.99

   5

Ok

(3) Overflow and division by zero

If division by zero is encountered during evaluation of an expression, the "Division by zero" message is displayed. In this situation, machine infinity is returned as the result of division, then execution continues.

Example

```
10 A#=30/0
20 PRINT A#
30 PRINT "Program line 30"
RUN
Division by zero
 1.701411733192645D+38
Program line 30
Ok
```

The "Division by zero" message is also displayed and machine infinity returned when zero is raised to a negative power.

An overflow error is the condition which occurs when the result of an operation is too large to fit into the memory available for storing the resulting type of numeric value. Whether or not execution continues depends on the type of operation attempted.

Example

```
10 A#=666^666
20 PRINT A#
30 PRINT "Program line 30"
40 A%=66666!\25
50 PRINT A%
60 PRINT "Program line 60"
RUN
Overflow
 1.701411733192645D+38
Program line 30
Overflow in 40
Ok
```

2-16

2.8.2 Relational operations

Operations in which two values are compared are referred to as relational operations. The result returned by such a comparison is either "true" (-1) or "false" (0), and is then used to make a decision regarding subsequent program flow. (See the discussion of the IF...THEN...ELSE and IF...GOTO statement sequences in Chapter 3.)

The relational operators and their meanings are listed below.

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

NOTE:
In a relational expression, the "=" sign has a meaning different than when it is used to assign a value to a variable. See the discussion of the LET statement in Chapter 3 for details on assigning values into variables.

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.

Example

```
10 A=1=1
20 PRINT A
30 B=3>4
40 PRINT B
50 PRINT 3>2
RUN
-1
0
-1
Ok
```

In the example above, line 10 tests for equality between the first and second operands of the relational expression "1 = 1", then stores the result (-1, or true) in variable A. Line 20 then displays the contents of A. Line 30 tests whether the first operand of the relational expression "3 > 4" is greater than the second, then stores the result (0, or false) in variable B. The result is then displayed by the statement on line 40. Line 50 evaluates and displays the result of the relational expression "3 > 2" (-1, or true).

2.8.3 Logical operations

A logical operation uses Boolean arithmetic to define the logical connection between the results (-1 or 0) of relational operations (or to set or reset specific data bits according to the logical connection between the individual bit states of the operands). In any given expression, logical operations are always performed after arithmetic and relational operations. The results of logical operations are determined as shown in the table below; the operators are listed in the table according to their order of precedence.

NOT (Negation)		
X	NOT X	
1	0	
0	1	

AND (Logical product)		
X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (Logical sum)		
X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR (Exclusive − OR)		
X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP (Inclusion)		
X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV (Equivalence)		
X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

Since relational operations can be used to make decisions concerning program flow, logical operators can be used to connect two or more relational operations; this allows decisions to be based on multiple conditions. (See the discussion of the IF...THEN...ELSE and IF...GOTO statement sequences in Chapter 3.)

Example

1) IF D<200 AND F<4 THEN 80
   This statement branches program execution to program line 80 if the contents of variable D are less than 200 and the contents of variable F are less than 4.
2) IF I<10 OR K<0 THEN 50
   This statement branches program execution to program line 50 if the contents of variable I are less than 10 or the contents of variable K are less than 0.

In a logical operation, the operands are converted to signed 16-bit two's complement integers † in the range from -32768 to 32767 before their logical connection is checked according to the operator (an error will result if an operand is not within this range). The specified operation is then performed for corresponding bits of each operand (bits in the same positions) and the result returned is the two's complement integer equivalent of the results for all bits. The following examples may aid your understanding of how logical operators work.

Example

```
10 PRINT 63 AND 16
RUN
 16
Ok
```

In binary notation, the twos complement integer 63 is 111111 and the twos complement integer 16 is 010000; since 1 AND 0 yields 0 and 1 AND 1 yields 1, the result is 010000, or 16.

---

† The first bit of a two's complement integer indicates whether the integer is positive or negative. In binary notation, the two's complement integers from 0 to 32767 are expressed as 0000000000000000 to 0111111111111111. The integers from -1 to -32768 are expressed as 1111111111111111 (-1) to 1000000000000000 (-32768). The value 1111111111111111 is obtained by adding 1 to the complement of 0000000000000001 (i.e., 1111111111111110 + 1 = 1111111111111111). The binary representations of other negative two's complement integers can be obtained in the same manner.

Example	10 PRINT 21 XOR 17
	RUN
	4
	Ok

The two's complement integer 21 is expressed in binary as 10101, while the two's complement integer 17 is expressed as 10001; since 1 XOR 1 and 0 XOR 0 yield 0, while 1 XOR 0 yields 1, the result is 00100, or 4.

Example	10 PRINT -1 OR -2
	RUN
	-1
	Ok

The two's complement integer -1 is expressed in binary as 1111111111111111, while the twos complement integer -2 is expressed as 1111111111111110; since both 1 OR 1 and 1 OR 0 yield 1, the result 1111111111111111, or -1.

Logical operators can be used to test data bytes for a particular bit pattern. For instance, the AND operator can be used to mask all but one bit of a status byte to obtain the status of a device I/O port; or, the OR operator can be used to merge two data bytes to obtain a particular binary value.

2.8.4 Functions

Functions are operations which return a specific value for a single operand. For example, the function SIN(X) returns the sine of the numeric value stored in variable X (where the value in X is in radians). A variety of functions are built into EPSON MFBASIC; these are referred to as intrinsic functions, and are described in Chapter 4.

MFBASIC also makes it possible for the programmer to write user defined functions; see the discussion of the DEF FN statement in Chapter 3 for details.

2.8.5 String operations

String operations involve manipulation of character strings with operators. For example, the " + " operator makes it possible to concatenate (link) strings as shown in the example below.

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

Character strings can also be compared using the same relational operators as are used with numeric values.

```
10 A$="ALPHA"
20 B$="BETA"
30 IF A$<B$ THEN 40 ELSE 60
40 PRINT A$;" IS LOWER THAN ";B$
50 END
60 PRINT A$;" IS NOT LOWER THAN ";B$
RUN
ALPHA IS LOWER THAN BETA
Ok
```

Strings are compared by taking one character at a time from each string and comparing their ASCII codes. The strings are equal if all codes are the same; if the codes differ, the character with the lower ASCII code is regarded as lower. Comparison ends when different characters are encountered in the two strings or when the end of one of the strings is reached; in the former case, the string in which the lower code was encountered is regarded as lower, and in the latter case, the shorter string is regarded as lower. Spaces included in strings are also significant; thus, the message on line 60 will be displayed by the example above if the character string stored in B$ is " BETA" (with a leading space). Further examples are shown below.

"AA" is less than "AB"
"FILENAME" is equal to "FILENAME"
"X&" is greater than "X #"
"CL" is greater than "CL"

"kg" is greater than "KG"
"SMYTH" is less than "SMYTHE"

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

2.9　Files

In general, a file is any set of data records which is output to or input from an external device (such as a disk drive) under a common identifier. This includes text files containing the program lines of MFBASIC programs, machine language program files, and data files. Files are stored on flexible disks; however, they may also be input from and output to other devices. (See Chapter 5 for information on the types of file organizations used by MFBASIC.)

With MFBASIC, files are identified by means of descriptors which consist of a device name and a file name. Together, these are referred to as the "file descriptor" and are specified as follows.

"<device name>:[<filename>]"

(1)　File names
A file name is composed of a primary name of up to 8 alphanumeric characters and an extension consisting of a period (".") and up to 3 alphanumeric characters. With the LOAD, MERGE, RUN, LIST, and SAVE commands, the extension ".BAS" is assumed if only the primary name is specified in the command's operand. With the FILES, KILL, or NAME commands, extensions must be specified.

(2)　Device names
With MFBASIC, the format of all input and output commands is the same regardless of the type of I/O device. I/O devices are distinguished from one another by means of device names; the devices which can be addressed for I/O operations are as follows.

Device name	Device
KYBD:	Keyboard (input only)
SCRN:	Display screen (output only)
LPT0:	Printer (output only)
COM0:	Standard RS-232C interface
COM1: to COM4:	Expansion RS-232C interfaces
A:	Disk drive A
B:	Disk drive B
E:	Disk image RAM (256K byte version only)
CMOS:	CMOS RAM

(3) File numbers

With MFBASIC, a logical file number must be assigned to each file which is read or written by a program (except when a text file is accessed using the LOAD, MERGE, RUN, LIST, or SAVE commands). This is done by means of the OPEN statement, which links the logical file number to the physical I/O device defined in the file descriptor. Unless otherwise specified with the /F: option when MFBASIC is activated, the maximum number of files which can be open at one time is 3. See the discussion of the OPEN statement in Chapter 3 for the procedure for assigning file numbers.

2.10 Display Screen

This section describes the character screen modes supported by MFBASIC (including the graphics mode and screen attributes), the systems of coordinates which are used for specifying the positions of characters and graphics on the display screen, and MFBASIC's color display support feature.

2.10.1 Screen modes

MFBASIC supports two modes of screen operation for the display of characters. These are referred to as the standard mode and the double width mode. The standard mode is that which is effective when MFBASIC is activated, and the double width mode is that which is entered by executing WIDTH 40. To return to the standard mode from the double width mode, execute WIDTH 80.

(1) Standard mode
In this mode, a maximum of 1600 1-byte characters can be displayed on the screen at one time, with 80 characters on each of the screen's 20 lines. Two 1-byte character positions are used for display of each 2-byte character. Such characters are displayed in the style which corresponds to the first byte of their 2-byte code; see section 2.3 above and Appendix J for details.

(2) Double width mode
In this mode, the display width of each character is twice its width in the standard mode; thus, a maximum of 800 1-byte characters can be displayed on the screen at one time, with 40 characters on each of the screen's 20 lines. As in the standard mode, each 2-byte character is displayed using the character positions for two 1-byte characters, and the style in which such characters are displayed is that which corresponds to the first byte of their 2-byte codes.

A unique feature of the double width mode is that any of 16 different character styles can be specified for 1-byte characters with the OPTION STYLE statement (this is not possible in the standard mode); use of the OPTION STYLE statement is described in Chapter 3.

(3) Graphics
Regardless of the mode of character display, the size of the graphic screen is 640 (horizontal) × 400 (vertical) dots. MFBASIC features a variety of statements and functions which allow the settings of these dots to be controlled individually or in groups for display of graphics. Use of these statements/functions is described in detail in Chapters 3 and 4. When the optional color display interface board is installed, any of 8 different colors can be specified for individual dots.

(4) Attributes
Regardless of whether the optional color display interface board is installed, the COLOR statement can be used to specify a variety of attributes for display of characters; these include automatic underlining of characters and reverse display. By

using the COLOR statement to specify the same color for both the background and foreground color of characters, it is also possible to make characters displayed invisible.

2.10.2 Coordinate specification

(1) Character coordinates

Character coordinates are used together with the LOCATE statement and the SCREEN, POS, and CSRLIN functions to specify or determine the positions in which characters are displayed. In the standard mode, horizontal coordinates are specified as numbers from 1 to 80, with 1 corresponding to the column on the left side of the screen and 80 corresponding to the column on the right side. Similarly, in the double width mode, horizontal coordinates are specified as numbers from 1 to 40. In either mode, vertical coordinates are specified as numbers from 1 to 20, with 1 corresponding to the top line of the screen and 20 corresponding to the bottom line. Thus, in the standard mode the character position at the top left corner of the screen is specified as (1,1), that at the bottom right corner is specified as (80,20), and so forth.

(1, 1)   (40 or 80, 1)
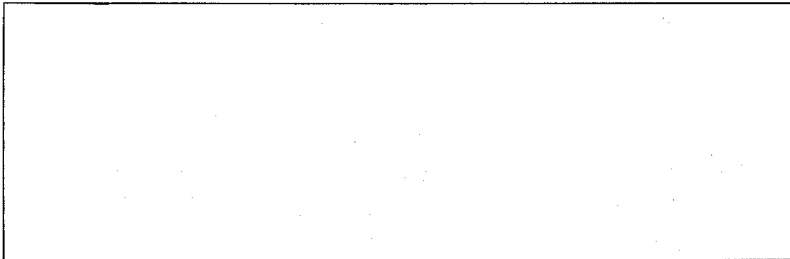
(1, 20)   (40 or 80, 20)

(2)   Graphic coordinates

Graphic coordinates are used to specify the positions of individual dots on the screen, and are used with statements such as PSET, PRESET, and POINT. Horizontal graphic coordinates are specified as numbers from 0 to 639, with 0 corresponding to the column of dots on the left side of the screen and 639 corresponding to the column of dots on the right side. Vertical coordinates are specified as numbers from 0 to 399, with 0 corresponding to the row of dots at the top of the screen and 399 corresponding to the row of dots at the bottom.

(0, 0)   (639, 0)

(0, 399)   (639, 399)
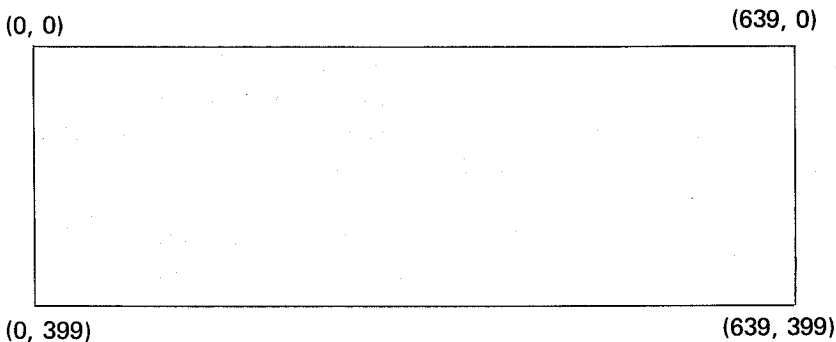
In most cases, graphic coordinates can be specified in either absolute or relative terms. Absolute coordinates are specified directly in the following form.

   (< horizontal position > , < vertical position >)

This form of coordinate specification can be used with all graphic statements and functions.

Relative coordinates are specified in the following form.

   STEP (< horizontal position > , < vertical position >)

Here, the effective position of the point specified is that of the coordinates of the last reference pointer plus the values specified for < horizontal position > and < vertical position >. Relative coordinate specification is possible with the PSET, PRESET, CONNECT, LINE, PAINT, GCURSOR, CIRCLE, and GET@ statements. The coordinates indicated by the last reference pointer are changed whenever one of these statements or the PUT@ statement is executed.

2.10.3 Color support

The statements and functions of QX-10 MFBASIC support color display using the optional color interface board. When this interface board is installed, any of 8 different colors can be specified for each of the screen's 640×400 graphic display dots. The foreground and background colors which are used as the default values for display of characters and graphics are specified with color codes by the COLOR statement. The foreground color is the color which is used for display of characters by statements such as PRINT, and is also the color which is assumed when a color code is not explicitly specified in the PSET, LINE, CONNECT, or CIRCLE statements.

The color codes are also significant when the optional color interface board is not installed. In this case, a color code of 0 corresponds to black (dot off) and the color codes from 1 to 7 correspond to white (dot on).

If the setting of a dot is read with the POINT function at this time, a color code of 7 is returned if the dot is set (on) and a color code of 0 is returned if the dot is reset (off).

However, the size of variable arrays required for pattern storage and display with the GET@ and PUT@ statements differs according to whether the color interface board is installed. When the color interface board is not installed, only one bit is used for each dot of the pattern (since only information concerning whether individual dots are set or reset need be stored); when the color interface board is installed, three bits are required to store the color code of each dot, and therefore about three times as much memory is required.

See the explanation of the COLOR statement in Chapter 3 for a full description of the meanings of the color codes.

2.11   Input/Output Device Support

MFBASIC supports data input/output (I/O) for a variety of peripheral devices. These include the built in flexible disk drives; a user memory bank referred to as disk image RAM (which can be used in exactly the same manner as the disk drives, but which provides much faster access); sequential access devices such as the display screen and RS-232C interface; and several special function devices.

2.11.1   Random access devices

Random access devices supported by the QX-10 are the built-in flexible disk drives and disk image RAM. These devices can be opened in either the random ("R") or sequential ("O") access modes, and can be used with all file input/output statements and functions.

The device names of the random access devices are as follows; these device names are included in the file descriptor with the file name as described in section 2.9.

Flexible disk drives ---- A: and B:
Disk image RAM   E:

2.11.2   Sequential access devices

Sequential access devices are devices which can be opened as files for input (the "I" mode), output (the "O" mode), or both. An OPEN statement includes a mode specification ("I" or "O"), a file number, and a file descriptor as described in section 2.9. However, unlike the random access devices, no file name is specified in the file descriptor. The device names under which these devices are opened as files, the I/O modes in which they can be used, and applicable input/output statements are as follows. Note that the colon must be specified following the device name even though it is not followed by a file name.

KYBD:	Keyboard; input only, using INPUT #, LINE INPUT #, and INPUT$.
SCRN:	Display screen; output only, using LIST, SAVE, PRINT #, PRINT # USING, POS, and WRITE #.
LPT0:	Line printer; output only, using the same statements as are used with SCRN:.
COM0:	Standard RS-232C communications interface; both input and output, using LOAD, RUN<file descriptor>, MERGE, INPUT #, LINE INPUT #, INPUT$, EOF, LOF, SAVE, LIST, PRINT #, PRINT # USING, WRITE #, and POS.
COM1: to COM4:	Same as above for up to four optional RS-232C interface cards.
CMOS:	CMOS RAM memory; both input and output, using the same statements as are used with COM0:.

NOTE:
The format for file specification of the RS-232C interfaces is slightly different from that of the other sequential access devices. See Chapter 6 for details.

2.11.3 Other devices

Other peripheral devices supported by MFBASIC include the optional light pen, the console speaker, and the built-in clock.

The light pen is a light sensitive device which makes it possible to read in the coordinates of characters or graphics displayed on the screen without using the keyboard, allowing the operator to devote undivided attention to the screen.

No device names are assigned to these devices, but they can be used with the following statements and fucntions.

```
Light pen  ------------  PEN ON/OFF
                         PEN functions
Speaker    -------------  BEEP
                         SOUND
Clock      -----------------  TIME/TIME$
                         DATE/DATE$
                         DAY
```

2.11.4 Commands, statements, and functions usable with I/O devices

The table below shows correspondence between the input/output commands, statements, and functions of MFBASIC and the devices with which they can be used.

General I/O

STATEMENT / DEVICE	KYBD:	SCRN:	LPT0:	COM0: to COM4:	CMOS:	DISK A, B, E
OPEN	○	○	○	○	○	○
CLOSE	○	○	○	○	○	○
INPUT #	○	×	×	○	○	○
LINE INPUT #	○	×	×	○	○	○
INPUT #	○	×	×	○	○	○
PRINT #	×	○	○	○	○	○
PRINT # USING	×	○	○	○	○	○
LOAD	×	×	×	○	○	○
SAVE	×	○	○	○	○	○
LIST	×	○	○	○	○	○
RUN	×	×	×	○	○	○
MERGE	×	×	×	○	○	○
EOF	×	×	×	○	○	○
WRITHE #	×	○	○	○	○	○
LOF	×	×	×	○	○	○
POS	×	○	○	○	○	○
PUT	×	×	×	×	×	○
GET	×	×	×	×	×	○
LOC	×	×	×	×	×	○
ACCESS MODE						
I	○	×	×	○	○	○
O	×	○	○	○	○	○
R	×	×	×	×	×	○

2.12　Error Messages

Error messages are displayed if errors are detected during execution of MFBASIC commands, statements or functions. If such errors occur during program execution, execution stops and MFBASIC returns to the command level. (However, it is possible to prevent this by including error processing routines which use the ON ERROR statement and the ERR and ERL functions in programs; see the explanations in Chapters 3 and 4 for details.)

Any of three languages (English, French, or German) can be specified for display of error messages with the /E:n option of the MFBASIC command when MFBASIC is started; for details, see "Starting MFBASIC" in Chapter 1.

A complete list of the MFBASIC error messages and their corresponding codes is provided in Appendix A.