LOAD

Format LOAD <file descriptor>[,R]

Purpose This command loads a program file into memory from a disk drive, disk image RAM, the RS-232C interface, or "CMOS:" file.

Remarks Specify the device name, primary file name, and extension under which the file was SAVEd in <file descriptor>. If the device name is omitted, the currently active drive is assumed; if the extension file name is omitted, ".BAS" is assumed.

When a LOAD command is executed, all files currently open are closed and all variables and program lines currently resident in memory are cleared before the specified program is loaded. However, if the "R" option is appended to the LOAD command, all data files are left open and the program is RUN as soon as loading is completed; therefore, LOAD with the "R" option may be used to chain programs (or segments of the same program).

When programs are chained in this manner, information may be passed betweem them by storing it in disk data files.

See also CHAIN, MERGE, RUN, SAVE

Example 1 LOAD "LNINPT"

Example 2 LOAD "B:LNINPT.BAS"

LOCATE

Format LOCATE [<X>][,[<Y>][,<cursor switch>]]

Purpose This statement moves the cursor to a specific position on the display screen.

Remarks The LOCATE statement moves the cursor to the character screen coordinates specified by <X>, <Y>.In the WIDTH 80 mode, the value specified for <X> must be in the range from 1 to 80, with 1 indicating the screen column on the far left; in the WIDTH 40 mode, the value specified for <horizontal position> must be in the range from 1 to 40. In either mode, the value specified for <Y> must be in the range from 1 to 20, with 1 indicating the top line of the screen.

Specifying 0 in the <cursor switch> option turns off the cursor, and specifying 1 turns the cursor on.

The cursor switch is forcibly set to 0 when MFBASIC returns to the command level; further, regardless of the setting of the cursor switch, the cursor is displayed when INPUT or LINE INPUT statements or the INPUT$ function are encountered during program execution.

Example

```
10 CLS
20 A$="ABCDEFG"
30 LOCATE 5,2
40 PRINT A$
50 LOCATE 10,3
60 PRINT A$
70 LOCATE 15,4
80 PRINT A$
90 END
```

```
    ABCDEFG
        ABCDEFG
            ABCDEFG
Ok
```

LPRINT/LPRINT USING

Format

LPRINT [<list of expressions>]
LPRINT USING <"format string">; <list of expressions>

Purpose

These statements are used to print data on the printer connected to the QX-10.

Remarks

These statements are used in the same manner as the PRINT and PRINT USING statements, but output is directed to the printer instead of the display screen.

See also

PRINT, PRINT USING

LSET/RSET

Format

LSET <string variable> = <string expression>
RSET <string variable> = <string expression>

Purpose

These statements move data into a random file buffer to prepare it for storage in a random access file with the PUT statement.

Remarks

If the length of <string expression> is less than the number of bytes in the random file buffer which were FIELDed to <string variable>, the LSET statement left-justifies the string data in the field and the RSET statement right-justifies it. Extra spaces in the random file buffer are padded with blanks. If the length of <string expression> is greater than the number of bytes FIELDed to <string variable>, characters are dropped from the right end of <string expression> when it is moved into the buffer.

Numeric values must be converted to strings before they are LSET or RSET. See the explanations of the MKI$, MKS$, and MKD$ functions in Chapter 4 for conversion of numeric values to strings.

See also

FIELD, GET, OPEN, PUT

Example

```
10 A$=STRING$(20," ")
20 N$="EPSON"
30 RSET A$=N$
40 PRINT N$
50 PRINT A$
Ok
RUN
EPSON
                EPSON
Ok
```

NOTE:
The LSET and RSET statements can also be used to left or right justify a string in a non-fielded string variable. For example, the following program lines right justify string N$ in a 20-character field prepared in variable A$. This can be very useful for formatting printed output.

MERGE

Format MERGE <file descriptor>

Purpose This command merges a program from a disk drive, disk image RAM, RS-232C interface, or the "CMOS:" file with the program currently in memory.

Remarks Specify the device name, primary file name, and extension under which the file was SAVEd in <file descriptor>. If the device name is omitted, the currently active drive is assumed; if the extension is omitted, ".BAS" is assumed. The file being merged must have been SAVEd in ASCII format. (Otherwise, a "Bad file mode" error will occur.)

If any lines of the program being merged have the same numbers as lines of the program in memory, the merged lines will replace the corresponding lines in memory. Thus, MERGEing may be thought of as "inserting" program lines from a storage device into the program in memory.

MFBASIC always returns to the command level after execution of a MERGE command.

See also SAVE

Examples
```
MERGE "TEST1"
MERGE "CMOS:TEST1.BAS"
```

MID$

Format MID$ (<string exp1>,n[,m]) = <string exp2>

Purpose This statement replaces a portion of one string with another string.

Remarks In the format above, n and m are integer expressions and <string exp1> and <string exp2> are string expressions. The statement replaces the characters beginning at position n in <string exp1> with the characters of <string exp2>. If the m option is specified, the first m characters of <string exp2> are used in making the replacement; otherwise, all of <string exp2> is used; however, the number of characters replaced cannot exceed the length of the portion of <string exp1> which starts with character n.

Example

```
10 A$="EPSON QX-10 BUSINESS COMPUTER"
20 B$="MFBASIC "
30 PRINT A$
40 MID$(A$,13,8)=B$
50 PRINT A$

RUN
EPSON QX-10 BUSINESS COMPUTER
EPSON QX-10 MFBASIC  COMPUTER
Ok
```

NOTE:
MID$ can also be used as a function to return a substring from a specified string expression; see Chapter 4.

NAME

Format NAME <old filename> AS <new filename>

Purpose This command is used to change the names of files on a flexible disk.

Remarks The file name specified in <old filename> must be that of a currently existing file, and that specified in <new filename> must be a name which is not assigned to any other file on the same disk; otherwise, an error will occur. If <old filename> is not the name of an existing file, a "File not found" error will occur; if <new filename> is already assigned to an existing file, a "File already exists" error will occur. An secondary file name must be specified in both <old filename> and <new filename>.

Results are not assured if the NAME command is executed against a file which is currently open.

This command changes only the name of the specified file; it does not rewrite the file into another area on the disk.

See also FILES

Example

```
NAME"SAMPLE1.BAS"AS"SAMPLE2.BAS"
```

This example assigns the new name SAMPLE2.BAS to the file that was formerly named SAMPLE1.BAS.

NEW

Format NEW

Purpose This command deletes the program currently in memory and clears all variables.

Remarks Enter NEW at the command level to clear the memory before entering a new program. MFBASIC always returns to the command level upon execution of a NEW command.

ON ERROR GOTO

Format ON ERROR GOTO [<line number>]

Purpose This statement causes program execution to branch to the first line of an error handling subroutine when an error occurs.

Remarks Once an ON ERROR GOTO statement has been executed, program execution will jump to the program line specified in <line number> whenever any error (including direct mode errors such as syntax errors) occurs. <line number> should be the first line of a user-written error handling subroutine which directs the course of subsequent processing; e.g., display of error messages.

If <line number> is not specified, the effect is the same as executing ON ERROR GOTO 0.

If subsequent error trapping is to be disabled, execute ON ERROR GOTO 0. If this statement is encountered in an error trapping subroutine, MFBASIC will stop and display the error message for the error which caused the trap. It is recommended that all error handling subroutines include an ON ERROR GOTO 0 statement for errors for which no recovery action is provided.

Use the RESUME statement to resume program execution at the line following that in which the error occurred.

NOTE:
MFBASIC always displays error messages and terminates execution for errors which occur during execution of an error handling subroutine; i.e., error trapping is not performed within an error handling subroutine itself.

See also ERROR, RESUME, ERL/ERR, Appendix A

Example See the explanations of the ERROR statement and the ERR/ERL functions.

ON...GOSUB/ON...GOTO

Format

ON <expression> GOTO <list of line numbers>
ON <expression> GOSUB <list of line numbers>

Purpose

This statement is used to transfer execution to one of the program line numbers specified in <list of line numbers>, depending on the value returned when <expression> is evaluated.

Remarks

The value of <expression> determines which line number in the list will be used for branching. If the value of <expression> is 1, program execution will branch to the first line number in the list; if it is 2, execution will branch to the second line number in the list; and so forth.

If the value of <expression> is a non-integer, it will be rounded to the nearest whole number. An "Illegal function call" error will occur if the value of <expression> is negative or greater than 256. If the value of <expression> is 0 or greater than the number of items in <list of line numbers>, program execution continues with the command or statement following the ON...GOSUB/ON...GOTO statement.

With the ON...GOSUB statement, each program line indicated in <list of line numbers> must be the first line number of a subroutine.

See also

GOSUB...RETURN, GOTO

Example

```
10 INPUT I
20 ON I GOSUB 100,200,300,400,500
30 GOTO 10
100 PRINT "ONE"
110 RETURN
200 PRINT "TWO"
210 RETURN
300 PRINT "THREE"
310 RETURN
400 PRINT "FOUR"
410 RETURN
500 PRINT "FIVE"
510 RETURN

RUN
? 3
THREE
? 2
TWO
? 5
FIVE
? 2
TWO
? 6
? 1
ONE
?
```

OPEN

Format

OPEN ''<mode>'',[#]<file number>,<file descriptor>,[<reclen>]

Purpose

The OPEN statement is used to open a disk file or other device for input or output.

Remarks

In this statement, <mode> is a string expression whose first character is one of the following.

- O Specifies the sequential output mode.
- I Specifies the sequential input mode.
- R Specifies the random input/output mode.

Any mode may be specified for a disk file or a file in disk image RAM. For other devices, only the ''I'' and ''O'' modes may be specified.

<file number> is an integer expression from 1 to 15 which specifies the number by which the file is to be referenced in I/O statements while it is open. <file descriptor> is a string expression which conforms to the rules for naming files (see section 2.8). <reclen> is an integer expression which, if specified, sets the record length for random access files. If not specified, the record length is set to 128 bytes.

A file must be OPENed before any I/O operation can be performed for that file. A disk file can be OPENed for sequential input or random access with more than one file number at a time.

Example

```
10 CLS
20 OPEN "o",#1,"NAMEOFCO"
30 LINE INPUT "NAME OF COMPANY?";A$
40 PRINT #1,A$
50 CLOSE #1
60 OPEN "I",#1,"NAMEOFCO"
70 LINE INPUT #1,A$
80 PRINT A$
90 CLOSE #1
100 END

NAME OF COMPANY?EPSON
EPSON
Ok
```

OPTION BASE

Format

OPTION BASE |0|
|1|

Purpose

This statement is used to declare the minimum value of array subscripts.

Remarks

When MFBASIC is started, the minimum value of array subscripts is set to 0; however, in certain applications it may be more convenient to use variable arrays whose subscripts have a minimum value of 1. Specifying 1 in this statement makes it possible to set the minimum subscript base to one.

Once the option base has been set by executing this statement, it cannot be reset until a CLEAR statement has been executed; executing a CLEAR statement restores the option base to 0. Further, OPTION BASE 1 cannot be executed if any values have previously been stored in any array variables. A "Duplicate Definition" error will result if the OPTION BASE statement is executed under either of these conditions.

Example

```
10 OPTION BASE 1
20 DIM N$(5,2)
30 FOR I=1 TO 3
40 PRINT "INPUT ITEM NAME FOR STOCK NO.
";:PRINT USING "_0_0_0#";I:INPUT A$
50 N$(I,1)=A$
60 PRINT "INPUT UNIT COST FOR STOCK NO.
";:PRINT USING "_0_0_0#";I:INPUT A$
70 N$(I,2)=A$
80 NEXT I
90 PRINT "STOCK NO.";TAB(20);"ITEM NAME"
;TAB(40);"UNIT COST"
100 FOR I=1 TO 3
110 PRINT USING "_0_0_0#";I;:PRINT TAB(2
0);N$(I,1);:PRINT TAB(40);:PRINT USING "
$******.##";VAL (N$(I,2))
120 NEXT I
```

```
RUN
INPUT ITEM NAME FOR STOCK NO. 0001
? CAMERA
INPUT UNIT COST FOR STOCK NO. 0001
? 350.99
INPUT ITEM NAME FOR STOCK NO. 0002
? TELESCOPE
INPUT UNIT COST FOR STOCK NO. 0002
? 250.50
INPUT ITEM NAME FOR STOCK NO. 0003
? LENS PAPER
INPUT UNIT COST FOR STOCK NO. 0003
? .50
STOCK NO.        ITEM NAME        UNIT COST
0001             CAMERA           $**350.99
0002             TELESCOPE        $**250.50
0003             LENS PAPER       $****0.50
Ok
```

OPTION COUNTRY

Format OPTION COUNTRY <character string>

Purpose This statement is used to select one of the international character sets for keyboard input/output, CRT display, and output to the printer.

Remarks When this statement is executed, the character set selected is that of the country which corresponds to the first character of <character string>. The characters corresponding to the character sets for the various countries are as follows.

"U"	or	"u"	...	U.S.A.
"F"	or	"f"	...	France
"G"	or	"g"	...	Germany
"E"	or	"e"	...	England
"D"	or	"d"	...	Denmark
"W"	or	"w"	...	Sweden
"I"	or	"i"	...	Italy
"S"	or	"s"	...	Spain

Executing the OPTION COUNTRY statement changes the keyboard layout to that of the character set for the specified country. All subsequent output to the CRT or printer, and all input from the keyboard is made using this character set. Also, the currency symbol output by the PRINT USING statement is changed to that of the specified country.

Example
```
OPTION COUNTRY "U"
OPTION COUNTRY "england"
```

OPTION CURRENCY

Format OPTION CURRENCY <character expression>

Purpose This statement changes the currency symbol.

Remarks When the first two characters of the format string of a PRINT USING statement are characters which correspond to code decimal 36 ($$ in USASCII), this statement changes the currency symbol which is output by the PRINT USING statement to that specified in <character expression> (or to the character code equivalent of that character if another character set is subsequently selected).

For example, executing OPTION CURRENCY "@" while the U.S. character set is selected causes the symbol "@" to be output as the currency symbol by PRINT USING statements including format strings which begin with "$$". If the character set for Germany is subsequently selected, the "§" symbol will be output as the currency symbol.

Example

```
10 OPTION COUNTRY "U"
20 OPTION CURRENCY "@"
30 PRINT USING "$$#####";100
40 OPTION COUNTRY "G"
50 PRINT USING "$$#####";100

RUN
  @100
  §100
Ok
```

NOTE:
The formatting characters used in PRINT USING statements are based on the USASCII keyboard. With the characters sets of other countries, use characters with the corresponding internal codes as shown below.

Hex.	Dec.	U.S.A.	England	France	Germany	Spain	Italy	Denmark	Sweden
40	64	@	@	à	§	@	@	@	É
26	38	&	&	&	&	&	&	&	&
23	35	#	£	#	#	Pt	#	#	#
24	36	$	$	$	$	$	$	$	¤
2E	46	.	.	.	.	.	.	.	.
2C	44	,	,	,	,	,	,	,	,
2B	43	+	+	+	+	+	+	+	+
2D	45	–	–	–	–	–	–	–	–

OPTION STYLE

Format OPTION STYLE <numeric expression>

Purpose This statement changes the character font of 1-byte characters displayed on the CRT screen in the WIDTH 40 mode.

Remarks Any value from 1 to 16 can be specified in <numeric expression>, allowing any of 16 character fonts to be selected. This statement only affects the display font for 1-byte characters; 2-byte characters input when any of keys SF1 to SF4 are pressed are not affected.

Example 1

```
10 WIDTH 40
20 FOR I=1 TO 16
30 OPTION STYLE I
40 PRINT TAB((I-1)*10);"EPSON";
50 NEXT I
60 END
```

```
EPSON     EPSON     EPSON     EPSON
EPSON     EPSON     EPSON     EPSON
EPSON     EPSON     EPSON     EPSON
EPSON     EPSON     EPSON     EPSON
Ok
```

Example 2

```
OPTION STYLE 1     BODONI
OPTION STYLE 2     OLD ENGLISH
OPTION STYLE 3     FLASH BOLD
OPTION STYLE 4     COMMERCIAL SCRIPT
OPTION STYLE 5     HELVETICA LIGHT
OPTION STYLE 6     HELVETICA LIGHT ITALIC
OPTION STYLE 7     HELVETICA MEDIUM ITALIC
OPTION STYLE 8     BROADWAY
OPTION STYLE 9     AMERICAN TYPEWRITER MEDIUM
OPTION STYLE 10    LIGHT ITALIC
OPTION STYLE 11    HELVETICA MEDIUM
OPTION STYLE 12    BODONI ITALIC
OPTION STYLE 13    SANS SERIF SHADED
OPTION STYLE 14    MICROGRAMA EXTENDED
OPTION STYLE 15    OLD GERMANY
OPTION STYLE 16    OCR B-FONT
```

NOTE:
Certain symbols are not displayed when option style 16 is selected.

OUT

Format OUT <integer expression 1>,<integer expression 2>

Purpose This statement is used to output data to a machine output port.

Remarks The data to be output is specified in <integer expression 2> and the port to which it is to be output is specified in <integer expression 1>. Both values must be in the range from 0 to 255.

Caution Use of this statement requires sound knowledge of the QX-10 firmware. For information on the QX-10 firmware, refer to the QX-10 Programer's Guide.

PAINT

Format PAINT [STEP](X,Y)[,[<area color>][,<border color>]]

Purpose This statement paints the area enclosed by a border of a specified color with another specified color.

Remarks The area which is painted when this statement is executed is that which includes the coordinates specified by (X,Y) and which is surrounded by a border of the specified <border color>; the color with which this area is painted is that specified in <area color>.

Relative point coordinates are assumed if the STEP option is specified.

<area color> and <border color> are both specified as color code numbers from 0 to 7; if <area color> is omitted, the current foreground color is assumed, and if <border color> is omitted, the color assumed is the same as that used for <area color>.

The meanings of the color codes from 0 to 7 are as follows.

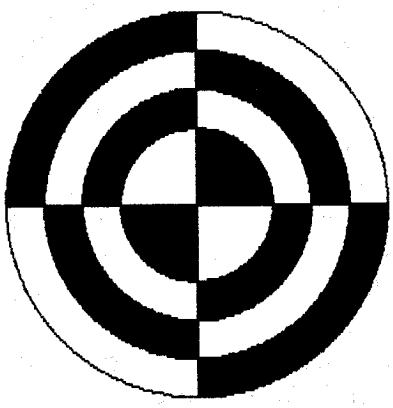
Color mode	Black and white mode
0 ... Black	0 ... Black
1 ... Blue	1 ... White
2 ... Red	2 ... White
3 ... Violet	3 ... White
4 ... Green	4 ... White
5 ... Light blue	5 ... White
6 ... Yellow	6 ... White
7 ... White	7 ... White

Nothing is done by this statement if the point specified by (X,Y) is already the color specified for <border color>. Further, an "Illegal function call" error will result if the point specified by (X,Y) lies outside the range of graphic coordinates of the display.

Example

```
10 CLS
20 FOR I=60 TO 150 STEP 30
30 CIRCLE (300,200),I
40 NEXT I
50 LINE (300,50)-(300,350)
60 LINE (150,200)-(450,200)
70 PAINT (295,55):PAINT (295,115)
80 PAINT (305,265):PAINT (305,325)
90 PAINT (205,205):PAINT (245,205)
100 PAINT (305,195):PAINT (395,195)

Ok
```



NOTE:
After execution of the PAINT statement, the last reference pointer (LRP) is updated to the values specified for (horizontal position 2, vertical position 2).

PEN

Format

PEN |ON|
|OFF|

Purpose

This statement turns ON or OFF input from the light pen.

Remarks

Executing PEN ON makes it possible to use the PEN functions, while executing PEN OFF disables the PEN functions. Therefore, the PEN ON statement must be executed before using the PEN functions. (It is recommended that the PEN OFF statement be executed to disable light pen interrupts when the light pen is not being used.)

Example

```
10 PEN ON
20 A=PEN(0)
30 PRINT A

RUN
 0
Ok
```

POKE

Format POKE <integer expression 1>,<integer expression 2>

Purpose The POKE statement is used to write a data byte into memory.

Remarks The address into which the data byte is to be POKEd is specified in <integer expression 1>, and the value which is to be POKEd into that address is specified in <integer expression 2>. The value specified in <integer expression 2> must be in the range from 0 to 255.

The complement of the POKE statement is the PEEK function, which is used to check the contents of specific addresses in memory. Used together, the POKE statement and PEEK function are useful for accessing memory for data storage, writing machine language programs into memory, and passing arguments and results between BASIC programs and machine language routines.

See also PEEK

Example

```
10 CLEAR ,&HBFFF
20 FOR N=&HC000 TO &HC006
30 POKE N,X
40 X=X+10
50 NEXT N
60 FOR N=&HC000 TO &HC006
70 I=PEEK(N)
80 PRINT I;
90 NEXT N
100 END

RUN
 0  10  20  30  40  50  60
Ok
```

NOTE:
Since this statement changes the contents of memory, the work area used by BASIC may be destroyed if it is used carelessly. This can result in erroneous operation, so be sure to check the memory map to confirm that the address specified is in a usable area.

PRESET

Format PRESET [STEP](X,Y)[,<color code>]

Purpose This statement resets the dot at the specified graphic display coordinates.

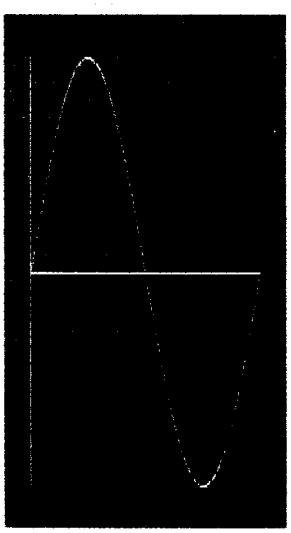
Remarks The graphic coordinates of the dot to be reset are specified by (X,Y). Relative coordinates are used when STEP is included; otherwise, absolute coordinates are used.

When <color code> is specified, the dot is reset to the specified color; if it is omitted, the dot is reset to the current background color.

See also CIRCLE, COLOR, CONNECT, LINE, PSET

Example

```
10 FOR I=80 TO 319
20 LINE (80,I)-(300,I)
30 NEXT
40 LINE (100,200)-(280,200),0
50 LINE (100,100)-(100,300),0
60 PI=3.14159
70 D=PI/180
80 FOR J=1 TO 360
90 X=100+J/2
100 Y=200-SIN(D*J)*100
110 PRESET(X,Y)
120 NEXT J
130 END
```



NOTE:
After execution of the PRESET statement, the last reference pointer (LRP) is updated to the values specified for (X, Y).

PRINT

Format PRINT [<expression>[; | , <expression>...]][; | ,]

Purpose This statement is used to output data to the CRT display screen.

Remarks Executing a PRINT statement without specifying any expressions advances the cursor to the line following that on which it is currently located without displaying anything on that line.

The expressions specified in the PRINT statement may be either string constants, string variables, numeric constants, or numeric variables.

Different types of expressions can be specified in the same PRINT statement. With string expressions, the character string corresponding to the constant or variable specified is displayed; if the statement includes any string constants, they must be specified in quotation marks. If numeric expressions are specified, values displayed are preceded by a space (if positive) or a minus sign (if negative), and are followed by a space.

Each expression specified in a PRINT statement must be separated from the following one by a space, semicolon, or comma. When a space or semicolon is used to separate expressions, the value of each expression is displayed immediately following that of the preceding expression in the list. When expressions are separated with commas, the value of each expression is displayed starting at the beginning of the MFBASIC print zone following the last character displayed for the previous expression (each print zone consists of 14 spaces). Other display formats can be obtained by including the TAB, SPACE$, or SPC functions in the list of expressions.

If a semicolon or comma is included at the end of the list of expressions, the cursor remains on the current display line after the PRINT statement has been executed; values specified in the next PRINT statement will then be displayed starting on the same line. If the list of expressions is concluded without a semicolor or comma, the cursor is moved to the beginning of the next line after the values have been displayed; this is the position at which display of values will begin the next time a PRINT statement is executed.

If the length of the values displayed by the PRINT statement is greater than the line length of the display, display is continued on the

next line.

When typing a PRINT statement, a question mark (?) may be substituted for the word PRINT; you will find this abbreviated form convenient when preparing programs which include many PRINT statments.

See also LPRINT, PRINT USING, TAB, SPACE$, SPC

Example 1

```
10 A=40:B=60
20 PRINT "A=";A,"B=";B
30 PRINT
40 PRINT "A+B=";A+B

RUN
A= 40        B= 60

A+B= 100
Ok
```

Example 2

```
10 PRINT 123,456,789
20 PRINT 123;456;789
30 PRINT "123";"456";"789"
40 PRINT "ABC",
50 PRINT "DEF"
60 PRINT "ABC";
70 PRINT "DEF"

RUN
 123          456          789
 123  456  789
123456789
ABC          DEF
ABCDEF
Ok
```

Example 3

```
10 PRINT "----+----*----+----*----+----*
----+----*"
20 PRINT "A";TAB(5);"B";TAB(10);"C";TAB(
30);"D"
30 PRINT "1";SPC(8);"2";SPC(29);"3"
40 PRINT "A";SPC(3);"B";SPC(3);"C";SPC(3
);"D"

Ok
RUN
----+----*----+----*----+----*
A   B    C              D
1        2                    3
A   B   C   D
Ok
```

Example 4

```
10 FOR I=1 TO 10
20 A(I)=5*I
30 PRINT A(I);
40 NEXT I

RUN
 5  10  15  20  25  30  35  40  45  50
Ok
```

NOTE:

Single precision numbers which can be represented with 6 or fewer digits are displayed in non-exponential format; otherwise they are displayed in exponential format. For example, 10△(-6) is output as .000001, while 10△(-7) is output as 1E-7. With double precision numbers, non-exponential format is used for numbers which can be displayed with 16 or fewer digits, while exponential format is used for numbers which require more than 17 digits. For example, 10△(-16) is output as .0000000000000001, while 10△(-17) is output as lD-17.

PRINT USING

Format PRINT USING <"format string">;<list of expressions>

Purpose This statement is used to display character strings or numbers in a specific format when displaying lists or tables.

Remarks <"format string"> is comprised of special characters which determine the format in which the expression or expressions in <list of expressions> are to be displayed. <list of expressions> consists of the string or numeric expressions which are to be displayed in the specified format; each expression in this list must be separated from the following one by a semicolon. The characters specified in <"format string"> differ according to whether the expressions included in <list of expressions> are string expressions or numeric expressions; these characters and their functions are as described below.

Format characters for string expressions

! Specifying an exclamation mark in <"format string"> causes the first character of each expression included in <list of expressions> to be displayed in a 1-character field.

Example

```
10 PRINT USING "!";"AAAAA";"BBBBB";"CCCCC"

RUN
ABC
Ok
```

\ n spaces \ Specifying a pair of backslashes with n intervening spaces in <"format string"> causes the first 2+n characters of each expression included in <list of expressions> to be displayed. Two characters will be displayed if no spaces are included between the backslashes, three characters will be displayed if one space is included between the backslashes, and so forth. Extra characters are dropped if the length of a string expression is greater than 2+n; if the length of a string expression is less than 2+n, it is left-justified in the field and padded on the right with spaces.

Example

```
10 A$="1234567"
20 B$="ABCDEFG"
30 PRINT USING "\    \";A$;B$
40 PRINT USING "\          \";A$;B$

RUN
12345ABCDE
1234567    ABCDEFG
Ok
```

& Specifying an ampersand in <"format string"> causes strings corresponding to the expressions specified in <list of expressions> to be displayed exactly as they are.

Example

```
10 A$="Epson"
20 B$="QX-10"
30 PRINT USING "&";A$;" ";B$

RUN
Epson QX-10
Ok
```

Format characters for numeric expressions

The positions in which the digits of numbers are displayed by a PRINT USING statement are determined by specifying # signs in <"format string">, with each # sign corresponding to the position of one digit. When the number of digits to be displayed is fewer than the number of # signs included in format string, the number is right-justified in that field. If the number of digits to be displayed is greater than the number of # signs, a percent sign (%) is displayed in front of the number and all digits are displayed. Minus signs are included in front of negative numbers, but (except as described below) plus signs are not displayed in front of positive numbers. The following is an example of use of the # sign in the format string of a PRINT USING statement.

```
10 PRINT USING "####";1
20 PRINT USING "####";.12
30 PRINT USING "####";12.6
40 PRINT USING "####";12345

RUN
   1
   0
  13
%12345
Ok
```

A number of special characters may be used in combination with the # sign to specify special formats; use of these characters is described below.

A decimal point may be included in the format string to indicate the position in which the decimal point of numbers is to be displayed. If the number to be displayed is less than 1, 0 will be displayed in the position to the left of the decimal point (unless the decimal point is specified as the first character in <"format string">). Digits to the right of the decimal point in the expression are rounded to fit in the number of places specified by the # signs to the right of the decimal point in <"format string">.

Example

```
10 PRINT USING "###.##";123
20 PRINT USING "###.##";12.34
30 PRINT USING "###.##";123.456
40 PRINT USING "###.##";.12
50 PRINT USING "###.##  ";123;12.34;123.456;.12

RUN
123.00
 12.34
123.46
  0.12
123.00  12.34  123.46   0.12
Ok
```

\+ Specifying a plus sign (+) at the beginning or end of the format string causes numbers displayed to be preceded by their sign (plus or minus).

Example

```
10 PRINT USING "+####";123
20 PRINT USING "+####";234

RUN
 +123
 +234
Ok
```

\- Specifying a minus sign at the end of the format string causes negative numbers to be displayed with a trailing minus sign.

Example

```
10 PRINT USING "####-";345
20 PRINT USING "####-";-456

RUN
 345
 456-
Ok
```

******

Specifying a double asterisk at the beginning of the format string causes spaces to the left of numbers displayed to be filled with asterisks. The asterisks also represent the positions of two digits in the field.

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "******.##";A
40 NEXT
50 DATA 12.35,123.555,555555.88

RUN
**12.35
*123.56
555556.00
Ok
```

$$

Specifying a double dollar sign at the beginning of the format string causes the currency symbol specified with the OPTION CURRENCY statement to be displayed to the immediate left of numbers. The dollar signs also represent the positions of two digits in the field.

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "$$#####.##";A
40 NEXT
50 DATA 12.35,123.555,555555.88

RUN
  $12.35
 $123.56
%$555556.00
Ok
```

$

Specifying $** at the beginning of the format string combines the effect of the dollar sign and the asterisk. Numbers displayed are preceded by a dollar sign, and spaces between the dollar sign and the number are filled with asterisks. The symbols $** also represent the positions of three digits in the field (one of which is used for the dollar sign).

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "$*******.##";A
40 NEXT
50 DATA 12.35,123.555,555555.88

RUN
$****12.35
$***123.56
$555556.00
Ok
```

******

Same as above, except that the dollar sign is displayed immediately to the left of the number and positions to the left of the dollar sign are filled with asterisks.

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "**$*****.##";A
40 NEXT
50 DATA 12.35,123.555,555555.88

RUN
****$12.35
***$123.56
$555556.00
Ok
```

,.
or
,

Specifying a comma to the left of the period in a format string causes commas to be displayed to the left of every third digit to the left of the decimal point. If the format string does not include a decimal point, specify the comma at the end of the format string. If only the comma is specified, numbers are rounded to the nearest integer value for display. The comma in the format string represents the position of an additional digit in the field, and each comma displayed uses one digit position.

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "#######,.##";A;
40 PRINT USING "##########,";A
50 NEXT
60 DATA 12.35,123.555,555555.88

RUN
    12.35      12
   123.56     124
555,556.00  555,556
Ok
```

^^^^

Specifying four carets (exponentiation operators) at the right end of the format string causes numbers to be displayed in exponential format. The four carets reserve space for display of E+XX; any number of decimal places may be specified. Unless a leading + or trailing + or - is specified, the first space in the field is always blank when positive numbers are displayed, and contains a minus sign when negative numbers are displayed. Significant digits are displayed starting in the second position in the field, with the exponent and fixed point constant adjusted as necessary to allow the number to be displayed in the number of positions in the field.

Example

```
10 PRINT USING "##/##/##";12;34;56

RUN
12/34/56
Ok
```

```
10 FOR I=1 TO 5
20 READ A(I)
30 PRINT USING "###.##^^^^  ";A(I);
40 PRINT USING "+###.##^^^^  ";A(I);
50 PRINT USING "###.##^^^^-";A(I)
60 NEXT I
70 DATA 123.45,12.345,1234.5,123.45,-12.3456

RUN
 12.35E+01  +123.45E+00  123.45E+00
 12.35E+00  +123.45E-01  123.45E-01
 12.35E+02  +123.45E+01  123.45E+01
 12.35E+01  +123.45E+00  123.45E+00
-12.35E+00  -123.46E-01  123.46E-01-
Ok
```

```
10 PRINT USING "###_@";123

RUN
123@
Ok
```

– Specifying an underscore mark in the format string causes the character which follows to be output as a literal together with the number.

Example

```
10 FOR I=1 TO 3
20 READ A
30 PRINT USING "##########    ";A;
40 PRINT USING "######_#_@";A
50 NEXT
60 DATA 12.35,123.555,555555.88

RUN
     12      12#@
    124     124#@
  555556  555556#@
Ok
```

Other characters:

If characters other than those described above are placed at the beginning or end of a format string, those characters will be displayed before or after the formatted number. Operation varies from case to case if other characters are included within a format string; however, in general including other characters in the format string has the effect of dividing the string up into sections, with formatted numbers displayed in each section together with the dividing characters.

Example

```
10 READ A,B,C
20 PRINT USING "######    ";A;B;C
30 RESTORE
40 READ A,B,C
50 PRINT USING "##,##,##";A;B;C
60 RESTORE
70 READ A,B,C
80 PRINT USING "##/##/##";A;B;C
90 RESTORE
100 READ A,B,C
110 PRINT USING "(+######)    ";A;B;C
120 DATA 11,22,33

RUN
   11     22     33
    11     22     33
11/22/33
(   +11)  (   +22)  (   +33)
Ok
```