7. As a rule, BASIC ignores spaces. However, spaces are not permitted within variable names and keywords. If a variable is to be followed by a keyword, a space must be inserted to delimit them for correct operation.

PURPOSE Describes briefly how the command or statement functions.
EXAMPLE This shows a simple example of actual input.
REMARKS Hints on the correct use of each command or statement are given, along with a description of their functions.
SAMPLE PROGRAMME Sample programmes using the command or statement are presented for exercise.

In the HX-20, 5 programmes are independently managed in the respective programme areas. Therefore, to avoid the accidental destruction or loss of important data or programmes in the memory, be sure to check the programme area using a STAT or LIST command, before modifying any programme.

CHAPTER 3
Commands and Statements

CLEAR

FORMAT **CLEAR [<character area size>[,<RAM file size>]]**
PURPOSE To initialise variables and to set the size of the character area and the RAM file.
EXAMPLE **CLEAR 200, 256**

REMARKS CLEAR command sets all numeric variables to 0 and all string variables to null. CLEAR also closes all OPENed files and voids all data defined by DEF statements (DEFFIL, DEF FN, DEF USR, DEFINT, etc.). <character area size> is the size, in bytes, of the memory area used by BASIC for character string processing.

<character area size> is set to 200 bytes at each warm start. When executing operations on a large number of character strings, or when using a large character array, specify sufficiently large <character area size>, or an "OS" error will occur.
<RAM file size> sets, in bytes, the area of the memory to be used for RAM file storage. <RAM file size> is set to 256 bytes at each cold start and is not affected by warm start.

When using a CLEAR command to set <RAM file size>, <character area size> cannot be omitted. Also note that when changing the size of a RAM file in which data is stored, if the newly set size is smaller than the previously set size, data in excess of the new size will become void.

SAMPLE PROGRAMME

```
LIST

100 FOR I=1 TO 10
110 A(I)=I : NEXT I
120 GOSUB 200
130 CLEAR
140 GOSUB 200
150 END
200 FOR I=1 TO 10
210 PRINT USING"####";A(I);
220 NEXT I
230 PRINT
240 RETURN
RUN
   1   2   3   4   5   6   7   8   9  10

   0   0   0   0   0   0   0   0   0   0

>
```


CLOSE

FORMAT **CLOSE[[#]<file number>,[#]<file number>...]]**
PURPOSE To close file(s).
EXAMPLE **CLOSE #3**

REMARKS CLOSE closes a file specified by <file number>. <file number> is the number under which the file was opened. 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 statement without <file number> closes all files opened at the time of executing the statement. # before <file number> may be omitted.
Note that END and NEW statements always close all files automatically but a STOP statement does not close any files.
When an output file has been opened, the file must be closed in order to correctly complete the output processing of the data remaining in the buffer.
After CLEAR, LOGIN, NEW, DELETE, WIDTH, LOAD, RUN, or MERGE is executed, or a programme is edited, all the files being open at that time will be closed.

(See END, OPEN and 5.2, Sequential Files.)

*CLS

FORMAT **CLS**
PURPOSE To clear a text screen.
EXAMPLE **CLS**

REMARKS CLS clears only the text screen (virtual screen) on the LCD or external display and returns the cursor to its home position (i.e., the upper left-hand corner of the virtual screen). However, when a CLS is executed with both text and graphic screens being displayed on the LCD, the CLS will also clear the graphic screen. CLS functions the same as PRINT CHR$ (12).

(See GCLS.)

COLOR

FORMAT COLOR [<foreground colour>][,[<background colour>][,<colour set>]]
PURPOSE To specify the screen colours of the external display.
EXAMPLE COLOR 0, 3, 0

REMARKS COLOR is used to set the foreground colour (i.e., display colour of lines and characters) and background colour of the external display. This command is effective only when the HX-20 is connected to an external display (e.g., TV) through the optional display controller.
You can specify colours using colour codes 0 to 3. The foreground colour and the background colour can be specified independently. A total of 8 colours are available for display. A maximum of 4 colours can be output simultaneously and either of two colour sets can be selected using code "0" or "1". The colour codes and corresponding colours are shown below.

	Colour code
Colour set 0	0: Green
	1: Yellow
	2: Blue
	3: Red
Colour set 1	0: White
	1: Cyan
	2: Magenta
	3: Orange

The default values at warm start are as follows.
<Foreground colour>: 1
<Background colour>: 0
<Colour set> : 0

(See SCREEN.)

CONT

FORMAT CONT
PURPOSE To resume the execution of a programme that has been stopped.
EXAMPLE CONT

REMARKS If you press **BREAK** key during programme execution, or when a STOP in a programme is executed, execution of the programme will stop and the message "Break in XXXX" will be displayed. At this point, the programme execution resumes upon input of a CONT command.
CONT is usually used together with a STOP statement for programme debugging.
If you press **BREAK** key during the I/O operation to a printer or a cassette, message "Abort" will be displayed. In this case, programme execution cannot be resumed by a CONT command. This command is also invalid if the programme is edited after it was stopped by **BREAK** key or STOP.

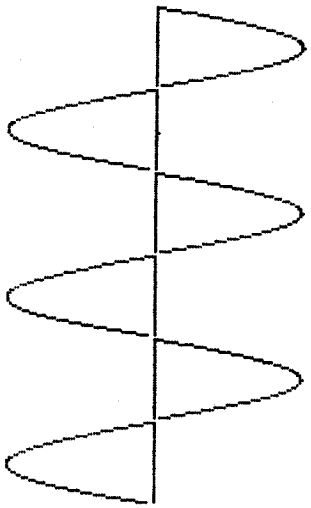
COPY

FORMAT COPY
PURPOSE To output the characters and graphics displayed on the LCD, on the built-in microprinter.
EXAMPLE COPY

REMARKS COPY command, unlike an LPRINT statement, provides hard copy without any space between lines. Therefore, with this command, you can print the graphics drawn using LINE and PSET statements exactly as they are displayed on the screen.

SAMPLE PROGRAMME

```
100 PI=3.141592
110 DX=PI/64
120 CLS
130 PRINTCHR$(23)
140 LINE(60,31)-(60,0),P
SET
150 FOR Y=0 TO 63
160 A=SIN(X)*59
170 X=X+DX
180 LINE-(A+60,Y/2),PSET
190 NEXT Y
200 COPY
210 GOTO 130
220 END
```



DATA

FORMAT DATA <constant>[,<constant>...]

PURPOSE To store the numeric and string constants that are accessed by the READ statement(s).

EXAMPLE DATA HX, 20, EPSON

REMARKS DATA statements are nonexecutable and may be placed anywhere in the programme and any number of DATA statements may be used in a programme.

Any type of constant can be used in <constant> (e.g., single precision, double precision, integer), while no numeric expressions (e.g., 3*4) are allowed. The variable type (numeric or string) given in the READ statement must agree with the corresponding constant in the DATA statement.

Each <constant> in a DATA statement must be separated by commas. String constants must be surrounded by double quotation marks if they contain commas, colons or significant leading or trailing spaces. Otherwise, quotation marks are not needed.

(See READ and RESTORE.)

SAMPLE PROGRAMME

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 1,2,3,4,5,6,7,8,
9,10
40 READ A,B,C
50 PRINT A;B;C
60 READ A,B,C
70 PRINT A;B;C
80 END
```

```
 1  2  3
 4  5  6
 7  8  9
```

DEFFIL

FORMAT DEFFIL <record length>, <relative address>

PURPOSE To define the relative address of record 0 in a RAM file and the length of a single record.

EXAMPLE DEFFIL 20, 200

REMARKS <relative address> is the number of bytes from the starting address of the RAM file area specified by a CLEAR statement. <record length> defines the length, in bytes, of a single record in the RAM file area to be used.
When BASIC is warm-started, <record length> is set at 255 bytes and <relative address> at 0. Execution of a CLEAR statement will also void the <record length> and <relative address> previously defined by the DEFFIL statement and the default values become 255 and 0 respectively at the time of a warm start.

(See GET%, PUT% and 5.1, RAM Files.)

SAMPLE PROGRAMME

```
LIST

100 DEFINT I,J,K
110 DEFFIL 2,0
120 FOR I=0 TO 15
130 PUT% I,I
140 PRINT USING "  & &";HEX$(I);
150 NEXT I:PRINT
160 DEFFIL 2,1
170 FOR J=0 TO 15
180 GET% J,K
190 PRINT USING "  & &";HEX$(K);
200 NEXT J
210 END
RUN
  0    1    2    3    4    5    6    7
  8    9    A    B    C    D    E    F

  0    100  200  300  400  500  600  700
  800  900  A00  B00  C00  D00  E00  F00
>
```


DEF FN

FORMAT DEF FN<name>[(<parameter>[,<parameter>....])]=
<function definition>

PURPOSE To define a function created by the user.

EXAMPLE DEF FNZ(X, Y)=X*2+Y*3+A

REMARKS <name> is the name of a function and must begin with alphabetic characters except reserved words (and is subject to the same restriction as variable names). The list of parameters comprises those variable names in the function definition that are to be replaced when the function is called. The items in the list are separated by commas. <function definition> is an expression that performs the operation of the function. It is limited to one line.
Variable names that appear in this expression serve only to define the function; they do not affect programme variables that have the same name. If a variable name used in <function definition> does not appear in the argument list, the current value of the variable is used.
User-defined functions may be numeric, string or a combination of both. However, the type of the defined argument must match that of the variable actually called. A DEF FN statement must be executed to define a function before it is called.
If a function is called before it is defined, a UF (undefined user function) error will occur.

SAMPLE PROGRAMME

```
100 DEFFNA(B,C,R)=SQR(B^
2+C^2-2*B*C*COS(R/180*3.
1415926))
110 INPUT"SIDE B=";B
120 INPUT"SIDE C=";C
130 INPUT"ANGLE R=";R
140 A=FNA(B,C,R)
150 PRINT "SIDE A= ";A
160 GOTO 110
170 END
```

DEFINT/SNG/DBL/STR

FORMAT DEF{INT, SNG, DBL, STR} <range(s) of letters>

PURPOSE To declare variable types as integer, single precision, double precision, and string.

EXAMPLE DEFSTR A,X–Z

REMARKS DEFINT, DEFSNG, DEFDBL and DEFSTR declare that the variable names beginning with the letter(s) specified by <range(s) of letters> will be integer type, single precision type, double precision type and string type, respectively.

To declare a variable or a range of variables, a single letter or two letters linked with a hyphen will be used as the letter(s) to be specified by <range(s) of letters>. To declare two or more ranges of variables, the letter(s) specifying each range of letters will be separated by a comma. However, a type declaration character always takes precedence over a DEF type statement in the typing of a variable. If no type declaration statements are encountered, BASIC assumes that all variables without declaration characters are single precision variables.

SAMPLE PROGRAMME

```
LIST

100 DEFINT A-C
110 DEFSNG E-G
120 DEFDBL I-K
130 A =123.456789
140 B#=123.456789
150 D =123.456789
160 E =789.123456
170 F%=789.123456
180 H#=789.123456
190 I =0.98765432
200 J%=0.98765432
210 L =0.98765432
220 PRINT A ,B#,D
230 PRINT E ,F%,H#
240 PRINT I ,J%,L
250 END
RUN
 123            123.456789     123.457
 789.124        789            789.123456

 .98765432      1              .987654
>
```

DEF USR

FORMAT DEF USR[<digit>]=<starting address>
PURPOSE To specify the starting address of a machine language subroutine.
EXAMPLE DEF USR6=&H0C00

REMARKS DEF USR specifies the starting address of the machine language routine called by the USR function. <digit> may be any digit from 0 to 9. A total of 10 USR functions are permitted. If <digit> is omitted, 0 is assumed.

(See USR and 5.3, Machine Language Programmes.)

SAMPLE PROGRAMME

```
MEMSET &H0A41

LIST

100 POKE &H0A40;&H39
110 DEFUSR0=&H0A40
120 ' &H39 = RETURN
130 A=USR(0)
140 PRINT A
150 END
RUN
 0
>
```


DELETE

FORMAT DELETE [<starting line number>][-[<ending line number>]]
PURPOSE To delete specified programme lines.
EXAMPLE DELETE 100-200
DELETE 100-
DELETE -200
DELETE 100

REMARKS DELETE command deletes all or part of the programme currently LOGged IN as specified by <starting line number> and <ending line number>. If only <starting line number> is specified, only that programme line is deleted. <starting line number> followed by a hyphen deletes that line and all higher-numbered lines. If <ending line number> preceded by a hyphen is specified, all the lines from the beginning of the programme through that line are deleted. When the DELETE command is executed, all files OPENed at the time of execution are closed.

(See LOGIN.)

SAMPLE PROGRAMME

```
100 'DELETE ELIMINATES
110 'UNEEDED LINES
120 'DELETE LINES 120 TO
 140
130 '***UNEEDED LINE***
140 '****EXCESS LINE***
150 '##IMPORTANT LINE##
160 '##NECESSARY LINE##

DELETE 120-140

100 'DELETE ELIMINATES
110 'UNEEDED LINES
150 '##IMPORTANT LINE##
160 '##NECESSARY LINE##
>
```

DIM

FORMAT DIM<variable>(<maximum subscript value>[,<maximum subscript value>...])[,...]
PURPOSE To declare the size of array variable elements.
EXAMPLE DIM A(40, 10), B$(50)

REMARKS DIM specifies the maximum value for array variable subscripts and allocates storage accordingly.

If an array variable name is used without a DIM statement, the maximum value of its subscript(s) is assumed to be 10. The minimum value for a subscript is always 0, unless specified as 1 with the OPTION BASE statement. If a subscript that is greater than the specified maximum is used, a BS ("Bad subscript") error occurs. The DIM statement sets all the elements of the specified numeric array to an initial value of zero, and those of the specified character array to null.

(See ERASE and OPTION BASE.)

END

FORMAT END
PURPOSE To close all files and terminate programme execution.
EXAMPLE END

REMARKS END terminates programme execution after closing all files and then returns BASIC to command level.
An END statement can appear anywhere and as often as required in a program. The END statement is optional at the end of a programme. In this case, no files will be closed.

ERASE

FORMAT ERASE <array variable>[,<array variable>...]
PURPOSE To eliminate arrays from a programme.
EXAMPLE ERASE A, B

REMARKS ERASE eliminates the previously dimensioned arrays specified by the list of array variables. After arrays have been erased, they can be redimensioned using a DIM statement. If an attempt is made to redimension an array without first erasing it, a DD ("Duplicate definition") error occurs.

(See DIM.)

SAMPLE PROGRAMME

```
LIST

100 OPTION BASE 1
110 D=10
120 FOR I=1 TO 10
130 A(I)=11-I
140 NEXT I
150 GOSUB 300
160 ERASE A
170 D=20
180 DIM A(20)
190 FOR I=1 TO 20
200 A(I)=I
210 NEXT I
220 GOSUB 300
230 END
300 FOR I=1 TO D
310 PRINT USING"####";A(I);
320 NEXT I
330 PRINT
340 RETURN
RUN
  10   9   8   7   6   5   4   3   2   1

   1   2   3   4   5   6   7   8   9  10
  11  12  13  14  15  16  17  18  19  20

>
```

ERROR

FORMAT ERROR<integer expression>
PURPOSE To simulate the occurrence of an error; or to allow error codes to be defined by the user.
EXAMPLE ERROR 225

REMARKS When an ERROR statement is executed without ON ERROR GOTO, the message of the error code corresponding to the <integer expression> will be output and the programme execution will stop.
If an ERROR statement specifies a code for which no error message has been defined, the HX-20 will respond with a message "UP Error". In all cases the value of the error code will be assigned to variable ERR and the line number in which the ERROR statement was encountered will be assigned to variable ERL.
An unused error code can be defined by the user with an ERROR statement, and this code can in turn be used in an error processing programme with an ON ERROR GOTO statement. <integer expression> must be in the range 1 to 255.

(See ON ERROR GOTO, RESUME, ERL/ERR, and APPENDIX A, Error Messages.)

SAMPLE PROGRAMME

```
LIST

100 INPUT "ERROR NUMBER"
;A
110 ERROR A
120 END

RUN
ERROR NUMBER? 78
UP Error In 110

RUN
ERROR NUMBER? 4
OD Error In 110

RUN
ERROR NUMBER? 24
WE Error In 110
2
```


EXEC

FORMAT EXEC [<starting address>]
PURPOSE To start the execution of a machine language subroutine.
EXAMPLE EXEC &H0C00

REMARKS EXEC executes a machine language subroutine from the <starting address>. However, before execution of an EXEC command, the machine language subroutine must be loaded into the memory by executing a LOADM command or similar statement.

When EXEC is used within a programme, the BASIC programme execution continues with the BASIC statement immediately following the EXEC statement after the machine language subroutine has terminated.
When <starting address> is not specified, execution begins from the <starting address> of the LOADM command or the EXEC command previously executed. BASIC command are ineffective while the machine language subroutine is being executed. The BASIC programme and the RAM file are not protected against careless use by the machine language subroutine and ample precautions must be taken to avoid their destruction.

(See LOADM, USR and 5.3 Machine Language Programmes.)

FILES

FORMAT FILES["<device name>"]
PURPOSE To display the names of all files residing on a specified memory.
EXAMPLE FILES "CAS1:"

REMARKS The following information is displayed for all files residing in the auxiliary memory specified by <device name>: filename, filetype, classification, recording format. Classification is displayed as a single digit,

0: BASIC programme file
1: Data file
2: Machine language programme file

Recording format is shown with either "A" or "B"

A: ASCII format
B: Binary format

If <device name> is omitted, all of the peripheral devices currently connected are automatically searched and the device with the highest precedence (i.e., default device) is assumed. If no peripheral device is connected, "CAS1:" is assumed. In BASIC, there is a feature for the detection of the end of files, but none for the detection of the end of tapes. For this reason, when "CAS1:" or "CAS0:" is specified as the <device name>, the cassette tape will run indefinitely. You must determine the end of the file, and press the **BREAK** key at the appropriate moment to stop execution.

FOR...TO...STEP–NEXT

FORMAT FOR <variable>=<initial value> TO <final value>
[STEP <increment>]
.
.
.
NEXT[<variable>[,<variable>...]]
PURPOSE To allow a series of instructions between FOR and NEXT statements to be performed in a loop a given number of times.
EXAMPLE FOR I=0 TO 100 STEP 5
.
.
.
NEXT I

REMARKS <variable> is used as a counter. The counter is first set to the value specified by <initial value>. BASIC executes the programme lines following the FOR statement until the NEXT statement is encountered. Then the counter is incremented by the <increment> amount specified by STEP and a check is performed to see if the value of the counter is greater than <final value>. If it is not greater, BASIC returns to the programme line following the FOR statement to repeat the same processing. If it is greater, execution continues with the statement following the NEXT statement.

Numeric expressions may be used for <initial value>, <final value> and <increment>. If STEP is not specified, <increment> is assumed to be one. If <increment> is negative, the <final value> of the counter is set to be less than the <initial value>.

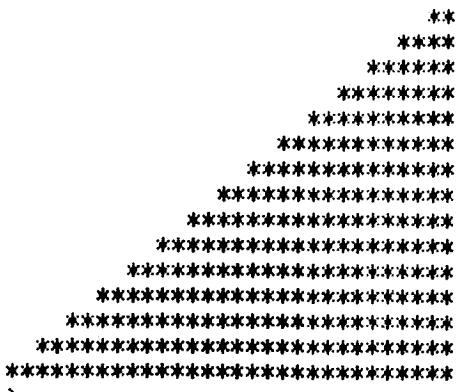
If <increment> is positive and the <initial value> of the counter is greater than the <final value>, or if <increment> is negative and the <initial value> of the counter is not greater than the <final value>, no FOR...NEXT loop will be executed. However, the <initial value> will be assigned to the <variable>.
FOR...NEXT loops may be nested, that is, a FOR...NEXT loop may be placed within the context of another FOR...NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop.
If a NEXT statement is encountered before its corresoponding FOR statement, a "NEXT without FOR" error message is issued and programme execution is terminated.

If nested loops have the same end point, a single NEXT statement may be used for all of them, by separating the variables to be represented with commas (e.g., NEXT I,J,K).

SAMPLE PROGRAMME

```
LIST

100 FOR I=30 TO 0 STEP -2
110 FOR J=1 TO I
120 PRINT" ";
130 NEXT J
140 FOR K=1 TO 30-I
150 PRINT"*";
160 NEXT K
170 PRINT
180 NEXT I
190 END
RUN
```



```
>
```


GCLS

FORMAT GCLS
PURPOSE To clear a graphic screen.
EXAMPLE GCLS

REMARKS GCLS clears only the graphic screen being displayed on the LCD or external display. If both text and graphic screens are displayed on the LCD, the text screen will be left as is.

(See CLS.)

GET%

FORMAT GET% <record number>,<variable name>[,<variable name>...]
PURPOSE To read data from a RAM file into variables.
EXAMPLE GET% O, A!, B#, C$

REMARKS GET% statement reads data from the record in the RAM file specified by <record number> into the variables specified by the list of variable names. <record numbers> start from 0.
Before executing a GET% statement, the length of a single record and the position of the file containing that record must be defined by a DEFFIL statement.
The types of the variable names in the GET% statement must correspond one-to-one with the types of the variable names in the PUT% statement. Also, when using string variable names, they should be added at the end of the variable list. When GET% and PUT% are to be used with multiple DEFFIL statements, careful attention must be paid to the length and record number of each record, as well as to the method of storing variable values.

(See DEFFIL, PUT%, and 5.1, RAM Files.)

GOSUB...RETURN

FORMAT GOSUB <line number>
.
.
.
RETURN

PURPOSE To branch to and return from a subroutine.

EXAMPLE **GOSUB 500**

REMARKS GOSUB causes an unconditional break in programme execution by transferring control to a subroutine specified by <line number> which is the first line of the subroutine. Programme control returns to the line following the most recent GOSUB statement when a RETURN statement in the subroutine is executed.
Subroutines are the independent portions of a main programme each of which contains a RETURN statement at the end of the programme.
Any of these subroutines may be called any number of times in a programme by a GOSUB statement. A subroutine may also be called from within another subroutine. Such nesting of subroutines is limited only by available memory.
A subroutine may contain more than one RETURN statement, if logic dictates a return at different points in the subroutine and if each RETURN statement corresponds correctly to a GOSUB.

SAMPLE PROGRAMME

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 Z=X
40 GOSUB 200
50 X1=ANS
60 Z=Y
70 GOSUB 200
80 X2=ANS
90 PRINT "X=";X,"Y=";Y
100 PRINT "(2*X+3)^2+(2*
Y+3)^2"
110 PRINT "=";X1+X2
120 END
200 ANS=(2*Z+3)^2
210 RETURN
```


GO TO/GOTO

FORMAT (1) **GO TO <line number>**, or
(2) **GOTO <line number>**

PURPOSE To branch programme execution to a specified line number.

EXAMPLE **GOTO 300**

REMARKS When a GOTO statement is executed, programme control branches unconditionally to a specified line number.
Formats (1) and (2) provide exactly the same function.

IF...THEN...ELSE/IF...GOTO...ELSE

FORMAT IF <logical expression>
THEN <statement> | <line No.>
GOTO <line No.>
[ELSE <statement> | <line No.>]

PURPOSE To choose a particular route for programme execution based on conditions established in a logical expression.

EXAMPLE IF A >10 THEN A=0 ELSE 200

REMARKS IF controls programme execution according to the conditions established in <logical expression>. If the result of <logical expression> is not zero (i.e., true), the THEN or GOTO statement is executed. If the result is zero (i.e., false), the ELSE statement is executed. If the ELSE statement is omitted, the execution continues with the next executable statement. Within the IF . . . THEN . . . ELSE statement, separate IF statements may be nested to create a multiplexed statement, limited to one line.

SAMPLE PROGRAMME

```
100 DEFSTR A-Z
110 DEF FNS=RIGHT$(TIME$
,1)
120 DEF FND=MID$(TIME$,7
,1)
130 CLS
140 OD=FND
150 OS=FNS
160 IF OS=FNS THEN 160
170 LOCATE 0,0,0
180 PRINT TIME$
190 IF OD<>FND THEN SOUN
D 12,4 ELSE SOUND 19,1
200 GOTO 140
210 END
```


INPUT

FORMAT INPUT ["<prompt string>" | ; | ,] <variable>[,<variable>...]

PURPOSE To allow input from the keyboard into a specified variable during programme execution.

EXAMPLE INPUT"NAME"; A$

REMARKS When an INPUT statement is encountered, execution of the programme pauses and the HX-20 waits for input from the keyboard. If <prompt string> is followed by a semicolon, the <prompt string> is displayed on the LCD followed by a question mark "?" with a significant 1-digit space.
When <prompt string> is followed by a comma, it appears on the LCD by itself with no following punctuation.
Data is input by pressing **RETURN** key and is assigned to a specified variable. When multiple variables, separated by commas, are specified, the data must also be separated by commas for input and must match the variables in terms of number and type. If they do not match, the system will display a "?Redo" message and return to a wait state for data input.
String constants need not be enclosed in double quotation marks for input unless they contain commas, colons or significant spaces.
INPUT is invalid in the direct mode.

(See LINE INPUT.)

SAMPLE PROGRAMME

```
100 INPUT"STRING";A$
110 INPUT"NUMBER";A
120 PRINT"THE STRING AND
 THE NUMBER ARE ";A$A

STRING? HX
NUMBER? 20

>
THE STRING AND THE NUMBE
R ARE HX 20
```

INPUT#

FORMAT INPUT# <file number>, <variable>[,<variable>...]
PURPOSE To read data items from a specified file and assign them to programme variables.
EXAMPLE INPUT#1, A, B, C$

REMARKS Other than the facts that data is read from a specified file and that no question mark is output, INPUT# functions essentially the same as an INPUT statement.
<file number> must be the number used when the file was OPENed for input. The data items in the file should be as those required in an INPUT statement. When you use an INPUT# statement to read data from a data file, data must be already prepared in the file. If the execution of the INPUT# statement continues after all the data in the file has been read, an IE ("Input past end") error occurs. If you observe this point, all the data which has been written into the data file using several PRINT# statements can be read with a single INPUT# statement.

(See INPUT, LINE INPUT#, OPEN and PRINT#.)

KEY

FORMAT KEY <key number>, <string>
PURPOSE To define the programmable function keys.
EXAMPLE KEY 1,"LIST"

REMARKS The keyboard of the HX-20 is equipped with 5 "programmable function keys" so that each key may be assigned to the function described in any string. With these keys combined with the SHIFT mode, a total of 10 strings can be defined. The length of character string including a control code must be a maximum of 15 characters. Characters that cannot be input from the keyboard can be specified using the function CHR$ appended to the string with a plus (+) sign.

(See KEY LIST.)

KEY LIST/KEY LLIST

FORMAT (1) KEY LIST
(2) KEY LLIST
PURPOSE To output the strings assigned to the programmable function keys on the screen and the microprinter, respectively.
EXAMPLE KEY LLIST

REMARKS (1) KEY LIST causes a complete list of strings assigned to the function keys to be displayed along with the key numbers. Each control code is displayed by typing the upper arrow "^" and a letter.
(2) KEY LLIST is the same as KEY LIST except that it causes a complete list of strings to be output on the built-in microprinter. When BASIC is started, the following strings are assigned to the respective function keys.

PF1	AUTO	PF6	?DATE$:?TIME$^M
PF2	LIST^M	PF7	LOAD
PF3	LLIST^M	PF8	SAVE
PF4	STAT	PF9	TITLE
PF5	RUN^M	PF10	LOGIN

LET

FORMAT [LET]<variable>=<expression>
PURPOSE To assign the value of an expression to a variable.
EXAMPLE LET A=3.141592

REMARKS LET assigns a value or the value of <expression> to <variable>. The actual assignment is performed by the equal "=" sign and LET can be omitted. <expression> may be a numeric or string constant. However, assignment of a string variable to a numeric variable and the reverse are not permitted. When assigning unmatched types of numeric variables, the variable type is to the right of the equal sign is converted into the one to the left of the equal sign before the assignment is performed.

LINE

FORMAT LINE[(<horizontal coordinate 1>, <vertical coordinate 1>)]–(<horizontal coordinate 2>, <vertical coordinate 2>), | PSET / PRESET | [,<colour>]

PURPOSE To draw a straight line between two specified points.

EXAMPLE LINE(0,0)–(119,31),PSET

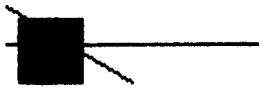
REMARKS This statement is used to draw a straight line between any two points on the graphic screen specified by the SCREEN statement. When using the optional display controller, you can specify colour(s) also by setting a graphic screen on the external display with a SCREEN statement. In this case, you must pay attention to the range within which you can specify coordinates, as the screen configuration of the LCD is different from that of the external display.

PRESET is used to erase a line between two points. With the LCD screen, the line is simply erased. However, with the graphic screen set on the external display, the line is drawn in the background colour specified by the COLOR statement.
If coordinates 1 are omitted, coordinates 2 of the previous LINE statement or the coordinates specified by PSET or PRESET are assumed.
NOTE: LINE Statements may not work if the value specified for either of the two coordinates is more than 4000.

(See COLOR, PRESET and PSET.)

SAMPLE PROGRAMME

```
10 CLS
20 FOR I=5 TO 30
30 LINE (5,I)-(30,I),PSE
T
40 NEXT I
50 LINE (0,0)-(50,30),PS
ET
60 LINE (0,15)-(100,15),
PSET
70 COPY
80 END
```



LINE INPUT

FORMAT LINE INPUT["<prompt string>";]<string variable>

PURPOSE To input an entire line to a string variable.

EXAMPLE LINE INPUT "WHAT?";A$

REMARKS LINE INPUT inputs an entire line of up to 255 characters without the use of delimiters from the keyboard and assigns it to <string variable>. <prompt string> is a string literal that is displayed on the LCD screen before input. A question mark is not displayed unless it is part of the prompt string. All key inputs from the end of the prompt string to RETURN key are assigned to <string variable>. Therefore, you can input delimiters such as commas, double quotation marks, etc., which are not usually permitted by an INPUT statement.
A LINE INPUT may be escaped by pressing BREAK key and BASIC will return to command level. In this case, input a CONT command to resume programme execution.

(See INPUT.)

SAMPLE PROGRAMME

```
10 INPUT "NO. OF CUSTOME
RS";N
20 FOR I=1 TO N
30 LINE INPUT "CUSTOMER
DATA?";C$(I)
40 PRINT C$(I)
50 NEXT I
60 END

RUN
NO. OF CUSTOMERS? 2
CUSTOMER DATA? TOM
JONES, FIELDING AVE.
 TOM  JONES, FIELDING AV
E.
CUSTOMER DATA? DAVID
COPPERFIELD, DICKENS
ST.
 DAVID COPPERFIELD, DICK
ENS ST
```

LINE INPUT#

FORMAT LINE INPUT#<file number>,<string variable>
PURPOSE To read an entire line from a sequential data file to a string variable.
EXAMPLE LINE INPUT #1, A$

REMARKS LINE INPUT# inputs an entire line of characters (255 characters max.) up to a carriage return from a sequential file without the use of delimiters and assigns it to <string variable>.
<file number> is the number under which the file was OPENed by an OPEN statement. <string variable> is the variable name to which the line will be assigned.
LINE INPUT# is especially useful if each line of a data file has been broken into fields, or if a BASIC programme saved in ASCII mode is being read as data by another programme.

(See INPUT#.)

LIST/LLIST

FORMAT (1) LIST[<starting line number>][–[<ending line number>]]
(2) LLIST[<starting line number>][–[<ending line number>]]
PURPOSE To output a programme list (1) on the LCD or external display or (2) on the microprinter.
EXAMPLE
```
LIST 100–200
LIST –200
LIST 100–
LIST 200
LIST.
LIST
```

REMARKS LIST outputs all or part of the programme currently LOGged IN as specified by <starting line number> and <ending line number>. If only <starting line number> is specified, only that programme line will be output. If <starting line number> followed by a hyphen is specified, that line and all higher-numbered lines are listed. If <ending line number> preceded by a hyphen is specified, all lines from the beginning of the programme through that line are listed. If both line numbers are omitted, the entire programme is listed. After the occurrence of an error, or after programme editing, the line number can be replaced with a period. Listing can be interrupted by pressing **BREAK** key, or temporarily halted with **PAUSE** key. A LIST command may be written in a programme for listing, but the command following the LIST command will not be executed and the system will return to command level.
The usage of LLIST is the same as that of LIST except that the list output is on the microprinter.

(See LOGIN.)

LIST <file descriptor>

FORMAT LIST <file descriptor>[,[<line number>][–[<line number>]]]
PURPOSE To output a programme list into a specified file.
EXAMPLE LIST "COM0:

REMARKS When a cassette is specified as the device name, this command functions the same as SAVE in ASCII format. If <file descriptor> is specified with a string expression or string variable, it always begins with double quotation marks (").
The line numbers are specified in the same manner as LIST except that line numbers must always be preceded by a comma.

(See LIST.)

*LIST "COM0:"

FORMAT LIST"COM0:[(<BLPSC>)]"[, [<line number>][–[<line number>]]]
PURPOSE To specify the interface conditions of the RS-232C port and execute LIST.
EXAMPLE LIST"COM0:(2701B)"

REMARKS The usage of this command is the same as that for LIST except that the output device is the RS-232C port.
For details of <BLPSC> which specifies the interface conditions, refer to OPEN"COM0:".

LOAD

FORMAT LOAD[<file descriptor>[,R]]
PURPOSE To load a programme file into the memory.
EXAMPLE LOAD"CAS1:PROG1.ASC"

REMARKS This command loads the programme file specified by <file descriptor> into the memory. When LOAD is executed, all open files are closed and all current variables are deleted. However, if the <R> option (i.e., load and run) is used with LOAD, all open files are left open and the programme is run immediately after it is loaded.
If <file descriptor> is omitted, the first file of the default device is loaded.
LOAD retains the programmes currently residing in the memory until the specified file is found and actual loading begins.
Before executing LOAD, use the STAT command to check the area currently LOGged IN.
Attempting to LOAD in an area which has been named by a TITLE statement, will result in the occurrence of a PP ("Protected programme") error.

(See LOGIN, SAVE, STAT and TITLE.)

*LOAD COM0:"

FORMAT LOAD"COM0:[(<BLPSC>)]"
PURPOSE To specify the interface conditions of the RS-232C port and execute LOAD.
EXAMPLE LOAD"COM0:(68N2B)"[,R]

REMARKS The usage of this command differs from the normal LOAD command only in that a programme is read through the RS-232C port. However, the transmitted programme must be in ASCII format or an error will occur and the programme will not be read. For <BLPSC> which specifies the interface conditions, refer to OPEN"COM0:"

LOADM

FORMAT LOADM<file descriptor>[,[offset value][,R]]
PURPOSE To load a machine language programme file into the memory.
EXAMPLE LOADM"CAS1:ABC"

REMARKS The file to be LOADed should be a machine language programme file created by the monitor function or the SAVEM command. If <file descriptor> is omitted, the first file of the default device is loaded.
<offset value> is added to the top address specified by the SAVEM command and loading begins at the resulting address.

If the <R> option is specified, after the machine language subroutine is LOADed into the memory, programme execution begins at <execution starting address> specified by the SAVEM command. If <R> is not specified, the HX-20 returns to BASIC command level after the machine language programme has been LOADed.
However, with a SAVEM command, the contents of the memory can be created as a file. If <R> is used with a LOADM command in a case other than machine language programme file, the CPU will interpret this file as a machine language programme file and will execute loading accordingly. If this happens, the BASIC programmes and RAM files may be destroyed. Please be careful when specifying <R> option.
<offset value> cannot be specified when the machine language subroutine is not relocatable, even if it is loaded at a location different from that at which it was saved. However, as the CPU performs no check to determine whether <offset value> is permitted or not, there may be cases in which a file is moved to a different address by the <offset value> even though the file contains the memory data.
With LOADM, "COM0:" cannot be specified as a device name.

(See SAVEM and 5.3, Machine Language Programmes.)

*LOAD?

FORMAT LOAD?[<file descriptor>]
PURPOSE To check files.
EXAMPLE LOAD?"CAS1:PROG1.ASC"

REMARKS LOAD? checks whether or not the data file specified by <file descriptor> has been correctly recorded. If <file descriptor> is omitted, "CAS0:" is assumed when the microcassette is connected to the HX-20, and "CAS1:" when it is not connected.
LOAD? command performs CRC check while reading the file (programmes and data) output to the auxiliary memory to confirm that the data file has been correctly recorded. If an error is found in the CRC check, an IO ("Device I/O") error will be displayed. As the data file is not actually loaded during the CRC check, there is no danger that the programs stored in the memory will be lost. This command is not intended to verify the contents of the cassette tape with the contents of the memory. For this reason, LOAD? can be executed even if there is no target programme in the area currently logged in.

When executing LOAD? against files (CAS1: CAS0:) stored on cassette tape, the tape will be searched until the file specified by <file descriptor> is found and then actual check begins. After the execution of this command, the position of the tape recorder head will be at the end of the specified file; namely, at the beginning of the next file.

*LOCATE

FORMAT LOCATE<horizontal coordinate>,<vertical coordinate>
[,<cursor switch>]
PURPOSE To specify the cursor position on the screen.
EXAMPLE LOCATE 10, 10, 0

REMARKS LOCATE positions the cursor at a specified position on the virtucal screen. Both <horizontal coordinate> and <vertical coordinate> must be specified. The cursor cannot be positioned outside the bounds of the virtual screen specified by a WIDTH command.
If the cursor is positioned outside the physical screen by executing a LOCATE command, the physical screen will automatically move to follow the cursor, so that the cursor is positioned at the upper left-hand corner of the screen. However, if the physical screen is at the right-hand end or the trailing end of the virtual screen, the cursor may not be at the upper left-hand corner. Even if the physical screen has been fixed at the left-hand end of the virtual screen (mode 0), the execution of a LOCATE command will cause the physical screen to move from that position (mode 1). (When the system returns to command level after the execution of a programme, the physical screen is put in mode 0.)
The cursor is turned off if the option <cursor switch> is specified as "0" and is turned on if specified as "1".

(See SCROLL.)

*LOCATES

FORMAT LOCATES <horizontal coordinate>, <vertical coordinate>
PURPOSE To specify the position of the physical screen.
EXAMPLE LOCATES 0, 0

REMARKS LOCATES moves the physical screen so that its upper left-hand corner will be in the location on the virtual screen specified by <horizontal coordinate> and <vertical coordinate>. The physical screen cannot leave the bounds of the virtual screen.
As with a LOCATE command, after the execution of a LOCATES command, the screen is always put in mode 1. Even after the execution of a LOCATES command, the cursor position on the physical screen will remain unchanged.

(See LOCATE)

*LOGIN

FORMAT LOGIN <expression>[,R]
PURPOSE To switch the programme areas.
EXAMPLE LOGIN 3

REMARKS In BASIC, the memory space is divided into five areas, each capable of storing separate programmes. A LOGIN command specifies the programme areas to be used (for programme execution, programme editing, etc.)
<expression> must be an integer between 1 and 5.
All commands for programme execution and modification (NEW, LIST, LOAD, SAVE, etc.) are effective only in the areas which have been specified by LOGIN commands. If the <R> option is specified, the programme execution begins soon after the area for the programme was switched. After the execution of a LOGIN command, all variables are cleared.

(See STAT.)

*MEMSET

FORMAT MEMSET [<bottom address of memory>]
PURPOSE To specify the lower limit of the memory.
EXAMPLE MEMSET &HOD00

REMARKS In BASIC, programmes written in machine language are placed before the BASIC programme text area. To enable the storage of machine language and BASIC programmes at the same time, the lower limit of the memory to be used by BASIC must be set using a MEMSET command. This also sets the memory locations for machine language programmes.
At cold start, <bottom address of memory> is set at &H0A3F. Warm start will not affect this <bottom address of memory>.
If <bottom address of memory> is omitted, &H0A3F is assumed. Upon execution of a MEMSET command, all variables are cleared.
Addresses 0 through &H0A3F are allocated as the I/O and system areas and are not permitted for the storage of machine language programmes.

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 200
110 FOR I=&H0A3F TO &H0B
00
120 POKE I,0
130 NEXT I
140 END
200 PRINT "BASIC PROGRAM
ME AREA. TO PROTECT BASI
C, POKE CANNOT BE EXECUT
ED."
210 RESUME 140
MEMSET &H0AFF
RUN

BASIC PROGRAMME AREA. TO
 PROTECT BASIC, POKE CAN
NOT BE EXECUTED.
```

MERGE

FORMAT MERGE [<file descriptor>[,R]]
PURPOSE To merge a specified programme file into the programme currently in memory.
EXAMPLE "CAS1:PROG3.ASC"

REMARKS MERGE command merges the programme file specified by <file descriptor> into the programme in the memory area currently logged in. The specified file must have been saved in ASCII format. If not, a BF ("Bad file mode") error occurs.
If <file descriptor> is omitted, the first file of the default device will be read. If any lines in the file have the same line numbers as lines in the programme in the memory, the lines in the file will replace the corresponding lines in the memory. If the option R is specified, the merged programme will be executed after the MERGE operation. BASIC always returns to command level after executing a MERGE command. When a MERGE command is executed, all files open at that time are closed and all variables are cleared. However, if <R> is specified, MERGE is executed with the files being left open.

(See SAVE.)

*MERGE "COM0:"

FORMAT MERGE "COM0:[(<BLPSC>)"[,R]]
PURPOSE To specify the interface conditions of the RS-232C port and to execute MERGE.
EXAMPLE MERGE"COM0":(68N2B)",R

REMARKS This command is essentially the same as the normal MERGE command except that the programme is read via the RS-232C port.
For details of <BLPSC> which specifies the interface conditions of the RS-232C port, see OPEN "COM0:"

MID$

FORMAT MID$ (<string exp 1>,<n>[,<m>])=<string exp 2>
where n and m are integer expressions and <string exp 1> and <string exp 2> are string expressions.
PURPOSE To replace a portion of one string with another string.
EXAMPLE MID$(A$,2)="BASIC"

REMARKS MID$ replaces the characters in <string exp 1>, beginning at position <n>, by the characters in <string exp 2>. The optional <m> efers to the number of characters from <string exp 2> that will be used in the replacement. If <m> is omitted, all of <string exp 2> is used. However, regardless of whether <m> is omitted or included, the replacement of characters never goes beyond the original length of <string exp 1>.
As the length of <string exp 1> never changes, the value of <n> cannot exceed the number of characters in (string exp 1), nor can <n> be a negative value. <string exp 1> cannot be a null string.

(See MID$ function)

SAMPLE PROGRAMME

```
100 FOR I=1 TO 20 STEP 4
110 A$="----------------
----"
120 MID$(A$,I)="*"
130 PRINT A$:NEXT
140 END
```

```
*-------------------
----*---------------
--------*-----------
------------*-------
----------------*---
```

MON

FORMAT MON
PURPOSE To transfer programme control to the machine language monitor.
EXAMPLE MON

REMARKS MON is used to transfer programme control from BASIC to the built-in machine language monitor. The machine language monitor commands are as follows:

S<address> Changes the contents of the memory. Input of a period (".") terminates this command.

D<address> Displays the contents of the memory.

G<execution address>,<breakpoint> Causes a programme to be executed up to <breakpoint>. The HX-20 then returns to Monitor.

K<string>^:@ Writes a menu number at the beginning of <string>, then writes a list of strings up to ^:@. When the power is switched on, the program specified by the menu number will be automatically selected and the same operations will be performed as if the character strings were input from the keyboard. <string> can be a maximum of 17 characters.
This function is cancelled upon input of K^:@, (which defines a null string for <string>).

B Causes HX-20 to escape from Monitor mode.

X Changes the contents of the registers. Each time **RETURN** key is pressed, the content of each register is displayed. By pressing **RETURN** key following the input of a numeric value, the contents of the register can be changed. This command is terminated upon input of a period.

R<device>,<filename>, R Loads a specified file into the memory from a specified device. If R is used with this command, the programme is run after loading the specified file.

V<device>,<filename> Performs CRC check of a specified file in a specified device.

W<device>,<filename> Saves a specified file in a specified device.

A Specifies addresses when R, V or W command is used.

Each time **RETURN** key is pressed, 2-byte data is requested in the following order.

T Top address of memory
L Bottom address of memory
O Offset
E Execution starting address

For <device>, specify C (Cassette), M (Micro-cassette), or P (ROM cartridge). However, with W command, P (ROM cartridge) cannot be specified. <filename> consists of a filename of eight or less characters and a three-character filetype.
<filename> = <filename>. <filetype>
For details, refer to Chapter 10, How to Use the Monitor in the HX-20 Operation Manual.

MOTOR

FORMAT MOTOR [<switch>]
PURPOSE To turn ON/OFF the motor of the external audio cassette.
EXAMPLE MOTOR ON

REMARKS MOTOR turns on or off the Remote terminal of the external audio cassette connected to the HX-20. By specifying <switch> as either ON or OFF, the motor of the external audio cassette can be controlled. If <switch> is omitted, the motor will reverse its ON/OFF state. In other words, if the Remote terminal is in the OFF state, the motor will reverse from OFF to ON and vice versa.

*NEW

FORMAT NEW
PURPOSE To delete the programme in the memory and clear all variables.
EXAMPLE NEW

REMARKS NEW deletes all programmes in the programme area currently LOGged IN. When this command appears in a programme, the programme in that area is cleared and BASIC returns to command level. If the programme is in the area that has been named by a TITLE statement, execution of a NEW command will result in a PP ("Protected programme") error. Execution of a NEW command causes all files currently open to be closed.

(See LOAD, LOGIN, and TITLE.)

ON ERROR GOTO

FORMAT ON ERROR GOTO <line number>
PURPOSE To enable error trapping and specify the first line of the error handling subroutine.
EXAMPLE ON ERROR GOTO 1000

REMARKS An ON ERROR GOTO statement transfers programme control to a specified error handling subroutine if an error occurs during the execution of a program. If an error is detected with this statement being executed, BASIC will execute a programme beginning with the line specified by <line number> and will not display an error message. Therefore, by enabling error trapping with that programme, you can prevent programme execution from being halted due to the occurrence of an error.
To disable error trapping, execute an ON ERROR GOTO 0. If an error is encountered for which there is no recovery action, execute an ON ERROR GOTO 0 in an error handling subroutine and an error message will be displayed and programme execution will be terminated.

(See RESUME, ERL/ERR, and APPENDIX A, Error Messages.)

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 200
110 INPUT "A=",A
120 IF A<0 THEN ERROR 25
0
130 IF FIX(A)<>A THEN ER
ROR 255
140 B#=1#
150 FOR I=2 TO A
160 B#=B#*I:NEXT I
170 PRINT A;"!"
180 PRINT "IS ";B#
190 GOTO 100
200 '***ERROR***
210 IF ERR=255 AND ERL=1
30 GOTO 250
220 IF ERR=6 AND ERL=160
 GOTO 270
230 IF ERR=250 AND ERL=1
20 GOTO 290
240 ON ERROR GOTO 0
250 PRINT "INPUT AN INTE
GER "
260 RESUME 110
270 PRINT "VALUE TOO LAR
GE "
280 RESUME 110
290 PRINT "NEG NUMBER NO
T ACCEPTED "
300 RESUME 110
310 END

RUN
A=10
 10 !
IS  3628800
A=-30
NEG NUMBER NOT ACCEPTED
A=17
 17 !
IS  355687428096000
A=45
VALUE TOO LARGE
```

*OPEN"COM0:"

FORMAT OPEN "<mode>", [#]<file number>, "COM0:[(<BLPSC>)]"
PURPOSE To specify the interface conditions for the RS-232C port and execute OPEN.
EXAMPLE OPEN "0", #1,"COM0:(68N2B)"

REMARKS This command is essentially the same as the normal OPEN command except that it specifies the interface conditions of the RS-232C port. <BLSPC> consists of 5 characters each specifying one of the interface conditions of the RS-232C port as follows.

B (Bit rate)
Numerics 0 to 6 are used to specify the bit rate (data transfer rate).

0: 110 bps
1: 150 bps
2: 300 bps
3: 600 bps
4: 1,200 bps
5: 2,400 bps
6: 4,800 bps

L (Word length)
Either 7 or 8 is used to specify the word length of 1-character data.

7: 7 bits/character
8: 8 bits/character

P (parity)
N, E, or O is used to specify the method of parity check.

N: No parity check
E: Even parity check
O: Odd parity check

S (Stop bits)
Either 1 or 2 is used to specify the stop bit length.

1: 1-bit length
2: 2-bit length

C (Control line active)
Active control (signal) lines are determined using hexadecimal digits 0 through F (corresponding to 4 binary bits, with each bit corresponding to one control line). In the following chart, active signal lines are represented by "○" and inactive lines (to be ignored) by "×". For the RTS signal, "+" sign indicates that the positive signal potential is active and "−" sign indicates that the negative signal potential is active.

Hexadecimal number \ Control line	CTS	DSR	RTS	CD
0	○	○	−	○
1	○	○	−	×
2	○	○	+	○
3	○	○	+	×
4	○	×	−	○
5	○	×	−	×
6	○	×	+	○
7	○	×	+	×
8	×	○	−	○
9	×	○	−	×
A	×	○	+	○
B	×	○	+	×
C	×	×	−	○
D	×	×	−	×
E	×	×	+	○
F	×	×	+	×

<BLPSC> can be omitted. If omitted, the HX-20 defaults to the values set last time. At warm start, HX-20 is set in the following conditions.

Bit rate: 4,800 bps
Word length: 8 bits/character
Parity: No parity check
Stop bit length: 2 bits
CTS: Ignore
DSR: Active
RTS: + potential is active.
CD: Ignore

Using the RS-232C port, two files (for input and output) can be opened at the same time. In this case, different interface conditions cannot be specified for the files to be opened. The conditions under which the first file was opened remain in effect.

OPTION BASE

FORMAT OPTION BASE [0 | 1]

PURPOSE To declare the minimum value for array variable subscripts.

EXAMPLE **OPTION BASE 1**

REMARKS OPTION BASE is used to declare the minimum value for array variable subscripts as either 0 or 1. The default base is 0. However, if an OPTION BASE 1 is executed, the minimum value for a subscript is set to 1.
Thereafter, if an array element is referenced with a subscript specified as 0, a BS ("Bad subscript") error will occur. This declaration cannot be made after array variables have been declared in a programme or referenced. Also, once declared, this minimum value cannot be altered by redeclaration, which will cause a DD ("Duplicate definition") error to occur.
Once an OPTION BASE statement has been executed, the declaration by the OPTION BASE can be neither changed nor cancelled until a RUN or CLEAR command is executed.

(See CLEAR and DIM.)

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 160
110 OPTION BASE 1 : DIM
A(9)
120 FOR I=0 TO 9
130 A(I)=I:PRINT A(I);
140 NEXT I
150 OPTION BASE 0:END
160 IF ERR=9 THEN PRINT
"MINIMUM SUBSCRIPT IS 1"
 : RESUME 140
170 IF ERR=10 THEN PRINT
 "OPTION BASE CANNOT BE
CHANGED"
180 RESUME NEXT
RUN
MINIMUM SUBSCRIPT IS 1
 1  2  3  4  5  6  7  8
 9
OPTION BASE CANNOT BE CH
ANGED
```


*PCOPY

FORMAT **PCOPY <expression>**

PURPOSE To copy BASIC programme into another programme area.

EXAMPLE **PCOPY 3**

REMARKS PCOPY command copies the programmes in the currently logged-in programme area into another programme area specified by <expression>. <expression> must be within the range of 1 to 5. If there is already a programme in the programme area specified by <expression> or if no programme exists in the currently logged-in programme area, execution of a PCOPY will cause an FC ("Illegal function call") error. If the programme to be copied is too large, an OM ("Out of memory") error occurs and PCOPY is not executed.

(See LOGIN, STAT, and TITLE.)

*POKE

FORMAT POKE <address>, <numeric expression>
PURPOSE To write a byte into a specified memory location.
EXAMPLE POKE &HOCOO, &H39

REMARKS POKE command writes one byte (8 bits) of data specified as <numeric expression> into a memory location specified by <address>.
<numeric expression> must be a one-byte integer expression in the range of 0 to 255 (&HO to &HFF). POKE rounds off less significant digits to the nearest integer (e.g., 1.5 to 2 and 1.4 to 1).
<address> must be a two-byte integer expression in the range of 0 to 65535 (&HO and &HFFFF). POKE rewrites the data currently in the memory.
Note that careless use of this command can be a source of malfunctions. Since &H0 to &H4D comprise a special area allocated for input/output, an overrun may occur just by reading it.
BASIC programmes are stored in the area above the address specified by a MEMSET command. For this reason, any attempt to execute a POKE statement against the abovementioned special area will cause an FC error to occur. To rewrite any of these addresses, you must write &H80 into address &H7E. In the area between addresses &H4E and &HA3F, execution of a POKE statement will not cause any error.
However, as this area is used as a work area by BASIC, rewriting any of the values in these memory locations may result in malfunctioning of BASIC. Therefore, please restrict execution of POKE statements to the machine language area between address &HA40 and the address specified as the bottom address of memory by a MEMSET statement. The complementary function to POKE is PEEK, a function which reads <numeric expression> from a specified memory location.
POKE and PEEK are useful for efficient data storage, loading machine language subroutines, and passing arguments and results to and from machine language subroutines.

(See PEEK, USR, and 5.3 Machine Language Subroutines.)

Character set selection

Character set selection can be made using the POKE and EXEC statements.
To select the character set, POKE the address &H7F and write the corresponding data shown below:

Character set of country to be selected	Data to be written
Spain	&H10
Italy	&H11
Sweden	&H12
Denmark	&H13
England	&H14
Germany	&H15
France	&H16
U.S.A.	&H17

The starting address for the EXEC statement is &HFF6A.

In order to return to the default character set (i.e., character set selected by the DIP switch), execute the following statements.

```
10 POKE &H7F,0
20 EXEC &HFF6A
```

Example: Spain

```
10 POKE &H7F,&H10
20 EXEC &HFF6A
```

PRESET

FORMAT PRESET (<horizontal coordinate>, <vertical coordinate>)
PURPOSE To erase a dot on a graphic screen.
EXAMPLE PRESET (40,25)

REMARKS PRESET resets the colour of any of the dots drawn on the graphic screen by a PSET or LINE statement to the background colour.
On the LCD, this command simply erases the specified dot from the graphic screen. If an external display is connected to the HX-20 using the optional display controller, this command will reset the specified dot to the background colour specified by a COLOR statement when the external display is set in the colour graphic mode by a SCREEN statement. As the LCD and the external display differ from each other in screen configuration, please pay attention to the range within which you can specify <coordinates>.

(See LINE, COLOR, PSET, and SCREEN.)

PRINT/LPRINT

FORMAT PRINT / LPRINT [<expression>[, / ;] <expression>...]]
PURPOSE To output data on the screen or the built-in microprinter.
EXAMPLE PRINT "EPSON"

REMARKS PRINT causes the data specified by <expression> to be output on the screen, while LPRINT command causes the same data to be output on the built-in microprinter. <expression> may be a numeric or string expression. If string expressions are used, character strings resulting from the expressions are output. If numeric expressions are used, the values resulting from the expressions are output. In this case, however, a blank space is left before and after each output value.
If the value is negative, a negative sign "−" is output in the blank preceding the value. When writing a list of expressions, the expressions must be separated by a comma, a semicolon or a blank. Delimiters can be omitted between a variable and a string constant and between one string constant and another. In this case, the effect is the same as when each expression was separated by a semicolon. A question mark ("?") may be used in place of PRINT in a PRINT statement, while LPRINT can neither be replaced nor omitted.

Output Format and Punctuation

- When the items in a list of expressions are separated with blanks or semicolons, the next value will be displayed or printed immediately after the last value. When the list of expressions are separated with commas, BASIC divides the line into display or print zones of 14 spaces each and outputs the value of each expression at each zone.
  If the result of an expression overlaps two or more zones, the next value will be output at the beginning of the zone following the last value.
- If the list of expressions terminates without a comma or semicolon, a carriage return is effected at the end of the line. If a comma or semicolon terminates the list of expressions, the next PRINT statement begins output on the same line, spacing accordingly.
- If the string to be output is longer than one line specified for the terminal, output continues on the next line. If all the strings or values to be output cannot be accommodated on a single line (between the cursor position and the end of the line), a carriage return is effected and output continues on the next line.

(See PRINT USING/LPRINT USING, SPC, and TAB.)

PRINT USING/LPRINT USING

FORMAT PRINT | LPRINT USING <"format string">;[<expression>[; | ,] <expression> ...]]

PURPOSE To output strings or numerics using a specified format.

EXAMPLE PRINT USING "####"; A,B

REMARKS PRINT USING/LPRINT USING determines the field and format of <expression> to be output by <"format string">. PRINT USING is used to display strings or numerics on the screen, while LPRINT USING is used to output the same on the microprinter.

String Fields

! Specifies that only the first character in the given string is to be output.

\ (n Blanks) \ . . . Specifies that (n+2) characters from the beginning of the given string will be output. If the string is longer than the field, the extra characters are ignored.
If the field is longer than the string, the string will be left-justified in the field and padded with spaces on the right. (See Note below.)

& Used to output character strings. When a number of & is specified, the value of a variable assigned to each & will be output. If the number of "&" is greater than the number of strings in the expression list, the extra "&" will be ignored.

Numeric Fields

A number sign is used to represent each digit position. Digit positions are always filled. If the number to be output has fewer digits than positions specified, the number will be right-justified (preceded by spaces) in the field. (See Note below.)

. A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be output as 0. Numbers are rounded as necessary. In the last example, three spaces were inserted at the end of the format string to separate the printed values on the line.

+ A plus sign at the beginning or end of a format string will cause the sign of the number (plus or minus) to be output before or after the number. If two or more plus signs are placed in succession, extra plus signs will be handled as "Characters Other Than Formatting Characters" described later.

– A minus sign at the end of a format string will cause negative numbers to be output with a trailing minus sign. If a minus sign is placed at the beginning of a format string, or if two or more minus signs are placed in succession, the minus signs will be handled as "Character Other Than Formatting Characters" described later.

** A double asterisk at the beginning of a format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

$$ A double dollar sign at the beginning of a format string causes a dollar sign to be output to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format ("^^^^") described below cannot be used with $$. (See Note below.)

$ The **$ at the beginning of a format string combines the effects of the above two symbols ("" and "$$"). Leading spaces will be asterisk-filled and a dollar sign will be output before the number. **$ specifies three more digit positions, one of which is used as the output area for a dollar sign.

, A comma that is to the left of the decimal point in a format string causes a comma to be output to the left of every third digit to the left of the decimal point. If a comma is placed to the right of the decimal point in a format string, a comma is output at the end of the formatted number.

^^^^ Four carets (or up-arrows) may be placed after the digit position ("#") characters to specify an exponential format. (See Note below.)

_ Used to represent any of the abovementioned formatting characters as a literal character. An underscore in a format string causes the next character to be output as a character which has no formatting function.

% If the number to be output is larger than the specified numeric field, a percent sign is output in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

Characters Other Than Formatting Characters

If any characters other than the abovementioned formatting characters (e.g., alphanumerics, graphic symbols, etc.) are placed at the beginning or end of a format string, such characters will be output before or after the formatted number.

NOTE: The formatting characters shown above apply to the ASCII character set. If your selected character set is other than ASCII, some of the formatting characters will be output differently as shown below.

USASCII	France	Germany	England	Denmark	Sweden	Italy	Spain
#	#	#	£	#	#	#	Pt
$	$	$	$	$	¤	$	$
\	ç	Ö	\	ø	Ö	\	Ñ
^	^	^	^	^	Ü	^	^

(See Section 2.4, Character Sets, for detailed information.)

SAMPLE PROGRAMME

```
100 A$="EPSON HX-20"
110 B=123.456
120 C=-123.456
130 D=1234.56
140 E=123456
150 PRINT USING "!";A$
160 PRINT USING "\      \
";A$
170 PRINT USING "THIS IS
 A &";A$
180 PRINT USING "###";B
190 PRINT USING "###.#";
B
200 PRINT USING "+##.#";
B
210 PRINT USING "###.#-"
;C
220 PRINT USING "**#.#";
B
230 PRINT USING "$$#.#";
B
240 PRINT USING "**$.#";
B
250 PRINT USING "###,.#^
^^^^";D
260 PRINT USING "$$#.__"
;D
270 PRINT USING "###";E
280 END
RUN
E
EPSON H
THIS IS A EPSON HX-2
0
123
123.5
%+123.5
123.5-
123.5
%$123.5
%$123.5
 123.5E+01
 123.5E+01
%$1235.-
%123456
>
```

*PRINT#

FORMAT PRINT# <file number>, [<expression>...]
PURPOSE To write data into a sequential file.
EXAMPLE PRINT#1,A,B

REMARKS <file number> is the number used when the file was OPENed for output. <expression> is the numeric or string expression which is to be written into the file. For CAS0: and CAS1:, PRINT#, unlike output to the display screen, automatically delimits the data before writing it into the file. Therefore, irrespective of whether commas or semicolons are used as delimiters in the list of expressions, data will be output in the same format.

(See INPUT#, PRINT/LPRINT USING, and PRINT# USING.)

PRINT# USING

FORMAT PRINT #<file number>, USING<"format string">;
[<expression>[; | ,] <expression>...]]

PURPOSE To write strings and numerics into a sequential file using a specified format.
EXAMPLE PRINT#1, USING"###"; A

REMARKS PRINT# USING writes string or numeric expressions into the sequential file specified by <file number> using the format specified by <"format string">. For <"format string"), refer to PRINT USING.
PRINT# USING outputs data in almost the same format as that for output to the display screen. Therefore, when reading a data file output by a PRINT USING, the data will not be delimited unless you use delimiters; commas for numeric expressions and double quotation marks for string expressions.

(See OPEN, PRINT USING, and PRINT#)

PSET

FORMAT PSET (<horizontal coordinate>, <vertical coordinate>)[,<colour>]
PURPOSE To draw dots on a specified graphic screen.
EXAMPLE PSET (30,20)

REMARKS PSET draws a dot at a specified location on the graphic screen specified by a SCREEN statement. PSET does not affect the dots already drawn on the screen. If the optional display controller is used, please pay attention when you specify coordinates, as the external display is different from the LCD in screen configuration. If you set the external display in colour graphic mode using a SCREEN statement, <colour> can also be specified. In this case, if the colour of the existing dots is different from your selected <colour>, it will change to the newly specified colour.
If <colour> is omitted, the colour of the dot to be drawn on the screen will be the same as the foreground colour specified in a COLOR statement.

(See COLOR, PRESET, and SCREEN.)

SAMPLE PROGRAMME

```
LIST

100 CLS:PI=3.14159
110 DEFINT X,Y
120 FOR I=0 TO 2*PI STEP PI/54
130 X=COS(I)*15+64
140 Y=SIN(I)*15+16
150 PSET(X,Y)
160 NEXT:COPY
>
```

*PUT%

FORMAT PUT% <record number>, <variable>[,<variable>...]
PURPOSE To write the values of variables into a RAM file.
EXAMPLE PUT%0, A!, B#, C$

REMARKS PUT% writes the values of variables into the record specified by <record number>. String variables must be written at the end of the variable list and only one string variable per record is permitted.

If all the values of the string variables cannot be written into the specified record, the excess portion of the string variables will be ignored.
If there is an extra capacity within the record after the last variable has been written, the record will be padded with spaces.

(See DEFFIL, GET% and RAM Files.)

SAMPLE PROGRAMME

```
100 DEFFIL 20,0
110 A=1.2:B%=3:C$="ABCDE
FG"
120 PUT%0,A,B%,C$
130 D=0.123:E$=C$+C$
140 PUT%1,A,B%,D,E$
150 'GET% CAN BE EXECUTE
D EVEN FOR TYPES DIFFERE
NT THAN THE PUT% STATEME
N BUT THE VALUE WILL BE
REFUSED
160 FOR I=0 TO 1
170 PRINT
180 GET% I,J,K%,L$
190 PRINT I;J;K%;L$
200 GET% I,J,K%,M,M$
210 PRINT I;J;K%;M;M$
220 NEXT I
230 END

 0  1.2  3
ABCDEFG
 0  1.2  3  8.22735E-20
EFG

 1  1.2  3
)( mABCDEFGABC
 1  1.2  3  .123
ABCDEFGABC
```


RANDOMIZE

FORMAT RANDOMIZE [<expression>]
PURPOSE To reseed the random number generator.
EXAMPLE RANDOMIZE

REMARKS RANDOMIZE changes the sequence of random numbers by supplying the random number generator with a new seed. The random number seed must be specified by <expression> within the range −32768 to 32767. If <expression> is omitted, a "Seed?" message will be displayed upon execution of RANDOMIZE, asking for the value of a random number seed. If the value input is outside the above random number seed range, an error will occur. RANDOMIZE statements cannot be executed in direct mode.

READ

FORMAT READ<variable>[,<variable>...]
PURPOSE To read values from a DATA statement and assigning them to variables.
EXAMPLE READ A, I, C$

REMARKS A READ statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree, a "Syntax error" will result.
If the number of variables in the list of variables in the READ statement(s) exceeds the number of elements in the DATA statement(s), an OD ("Out of data") error will occur.
If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

(See DATA and RESTORE.)

REM

FORMAT REM[<remark>]
PURPOSE To allow explanatory remarks to be inserted in a program.
EXAMPLE REM COMMENT MESSAGE

REMARKS A REM statement is a nonexecutable statement that is output exactly as it was entered in the programme.
In a REM statement, keyword "REM" can be replaced with an apostrophe ("'").
In a REM statement, a colon (":") is recognized as part of a <remark>. Therefore, no statement can be placed after the REM statement.

RENUM

FORMAT RENUM [<new line number>][,[<old line number>][,[<increment>]]
PURPOSE To renumber programme lines.
EXAMPLE RENUM

REMARKS <new line number> is the first line number to be used in the new sequence. The default is 10. <old line number> is the line in the current programme where renumbering is to begin. The default is the first line of the program. <increment> is the increment to be used in the new sequence. The default is 10.
RENUM also changes all line number references following GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB and ERL statements to reflect the new line numbers. If a line number nonexistent in the programme appears after one of these statements, the error message "Undefined Line Number nnnn in mmmm" is output. The incorrect line number reference (nnnn) is not changed by RENUM, but line number mmmm may be changed.
A RENUM command cannot be used to change the order of programme lines or to create line numbers greater than 64000. An FC ("Illegal function call") error will result if RENUM is used contrary to the above rules. In this case, the programme line numbers will remain unchanged. RENUM is valid only in the programme area currently LOGged IN.

RESTORE

FORMAT RESTORE [<line number>]
PURPOSE To allow DATA statements to be reread from a specified point.
EXAMPLE RESTORE 1000

REMARKS RESTORE causes the next READ statement to access the first item in the DATA statement at the line specified by <line number>. If <line number> is omitted, the next READ statement will read from the first DATA statement in the programme.
If there is no DATA statement at the line specified by <line number>, the READ statement will read from the DATA statement at the lowest line number after the specified <line number>.
If the line specified by <line number> does not exist in the programme, a UL ("Undefined line number") error occurs.

(See DATA and READ.)

SAMPLE PROGRAMME

```
LIST

100 DATA "*************","   HC -",2  0
110 READ A$
120 PRINT A$
130 READ A$,B%
140 PRINT A$;B%
150 RESTORE 100
160 READ A$
170 PRINT A$
180 END
RUN
*************
   HC - 20
*************
>
```

RESUME

FORMAT RESUME [| NEXT | <line number> |]

PURPOSE To continue programme execution after an error recovery procedure has been performed.

EXAMPLE **RESUME 100**

REMARKS With RESUME, you can specify the statement or programme line at which programme execution is to resume after an error recovery procedure has been completed. The three format options are as follows.

RESUME	Execution resumes at the statement which caused the error.
RESUME NEXT	Execution resumes at the statement immediately following the one which caused the error.
RESUME<line number>	Execution resumes at the line specified by <line number>.

(See ON ERROR GOTO.)

RUN

FORMAT (1) **RUN [<line number>]** or
(2) **RUN <file descriptor>[,R]**

PURPOSE To start programme execution.

EXAMPLE (1) **RUN 300**
(2) **RUN"CAS0:PROG4.ASC"**

REMARKS In format (1), if <line number> is specified, execution of the programme currently in the memory begins on that line. If <line number> is omitted, execution starts at the lowest line number.
In format (2), a programme specified by <file descriptor> is loaded into the memory and then programme execution begins.
Execution of a RUN command clears all variables and closes all open files before loading the designated programme. However, if the <R> option is used with this command, all data files remain OPEN.
Before executing a RUN command in format (2), use a STAT command to check the programme area currently LOGged IN.

(See LOGIN and STAT.)

RUN"COM0:"

FORMAT **RUN"COM0:[(<BLPSC>)]"[,R]**

PURPOSE To specify the interface condition of the RS-232C port and execute RUN.

EXAMPLE **RUN"COM0:(68N2B)"**

REMARKS Except for the fact that the programme is loaded into memory via the RS-232C port, this command is essentially the same as the normal RUN command. However, the programme to be transmitted through the port must be in ASCII format.
For details of <BLPSC>, which specifies the interface conditions of the RS-232C port, refer to OPEN"COM0:".

SAVE

FORMAT SAVE <file description>[,|A|]]
|V|

PURPOSE To save an EPSON BASIC programme on a specified file.
EXAMPLE **SAVE"CAS0:ABC"**

REMARKS This command is used to save BASIC programmes on the file specified by <file descriptor>. If a file already exists under the specified filename, the programme can be saved on another location under the same filename. If this is done, two different programmes saved cannot be distinguished from each other as they are under the same filename. Be careful not to use the same filename.
The <A> option saves a file in ASCII format. If <A> is not specified, the file is saved in a compressed binary format. ASCII format requires more space than the binary format but some file access requires that the file be in ASCII format (e.g., when using the MERGE command for programme editing). Also, a file saved in ASCII format may be read as a data file.

The <V> option may be used when a microcassette recorder is being used as an auxiliary memory. In this case, after the SAVE is executed, the tape will automatically be rewound to the beginning of the programme and program verification (CRC check) will be performed. If <device> other than the microcassette recorder is specified, the <V> option will be ignored.

(See LOAD and MERGE.)

*SAVE"COM0:"

FORMAT **SAVE"COM0:[(<BLPSC>)]",A**
PURPOSE To specify the interface conditions of the RS-232C port and execute SAVE.
EXAMPLE **SAVE"COM0: (68E13)",A**

REMARKS Except for the fact that the programme is output via the RS-232C port, this command is essentially the same as the normal SAVE command.
If A is omitted, no error will occur. However, since the data in a compressed binary format cannot be read properly by the RS-232C port, an attempt to transmit such data will be meaningless.
For details of <BLPSC>, which specifies the interface conditions of the RS-232C, refer to OPEN"COM0:".

*SAVEM

FORMAT **SAVEM <file descriptor>,<top address>,<bottom address>, <execution starting address>[,V]**
PURPOSE To save the memory contents on a specified file.
EXAMPLE **SAVEM"CAS1:ABC", &H0B00, &H0C00, &H0B00**

REMARKS SAVEM saves a machine language programme or memory contents on a specified file.
<top address> and <bottom address> indicate the range of the memory contents to be saved on the specified file.
If the <V> option is specified, programme verification (CRC check) is performed after SAVEM is executed. If a device other than the microcassette recorder is used, the <V> option will be ignored.
Execution of machine language programme loaded into the memory by a LOADM command will begin at <execution starting address>. Even if the data saved is not a machine language programme, <execution starting address> cannot be omitted and the same value as the <top address> must be set.

(See LOADM.)

*SCREEN

FORMAT **SCREEN <text>, <graphic mode>**
PURPOSE To specify the text or graphic screen mode.
EXAMPLE **SCREEN 0,2**

REMARKS SCREEN sets the LCD or external display in the text screen mode and/or graphic screen mode. (The optional display controller is required to connect the HX-20 to an external display.)
The value of <text> must be either 0 or 1. Specify 0 to display a text screen (physical screen) on the LCD and 1 to display a text screen on the external display.

The value of <graphic mode> must be in the range of 0 to 2. Specify 0 to display a graphic screen on the LCD. If 1 is specified, the external display is set in colour graphic mode (128×64 dots, 4-colour). If 2 is specified, the external display is set in high-resolution mode (128×96 dots, black and white). At the time of warm start, the LCD is set in both text and graphic screen mode.

SCREEN 0,0 ... Displays both text and graphic screens on the LCD.
SCREEN 0,1 ... Displays a text screen on the LCD and a graphic screen in the colour mode on the external display.
SCREEN 0,2 ... Displays a text screen on the LCD and a graphic screen in high-resolution mode on the external display.
SCREEN 1,0 ... Displays a text screen on the external display and a graphic screen on the LCD.

Due to the limited capability of the display controller, both text and graphic screens cannot be displayed simultaneously on the external display.

*SCROLL

FORMAT **SCROLL[<speed>][,[<mode>][,<scroll step X>,<scroll step Y>]]**
PURPOSE To specify the SCROLL function of the physical screen.
EXAMPLE **SCROLL 9,0,10,4**

REMARKS <speed> specifies the speed when the LCD screen scrolls using a value in the range of 0 to 9, with 9 indicating the highest scrolling speed and 0, the lowest. However, the scrolling speed of the external display is fixed and not adjustable.

<mode> specifies the manner of movement of the physical screen and its value must be either 0 or 1.
If you specify <mode> as 0, the physical screen is fixed at the left-hand end of the virtual screen and can move only up or down. In this mode, if the cursor position is outside the bounds of the physical screen, note that it will not come back into view unless the physical screen is moved, even if an output command is executed.
If you specify <mode> as 1, the physical screen will automatically move to follow the cursor.
When the system returns to command level after the execution of a programme, <mode> is set to 0.
<scroll step X> indicates the number of characters for which the screen should move at one time when CTRL + [→] or CTRL + [←] are typed. The value of <scroll step X> must be in the range of 1 to 20 for the LCD and 1 to 32 for the external display. <scroll step Y> indicates the number of lines the screen should move vertically at one time when CTRL + P, CTRL + Q or SCRN are typed. The value of <scroll step Y> must be in the range of 1 to 4 for the LCD and 1 to 16 for the external display. The default values at warm start are as follows:

For the LCD	SCROLL 9, 0, 10, 4
For the external display	SCROLL 9, 0, 16, 16

*SOUND

FORMAT SOUND <tone>,<duration>
PURPOSE To sound a specified tone.
EXAMPLE SOUND 10, 10

REMARKS SOUND causes the built-in piezoelectric speaker to sound at a specified <tone> for a specified <duration>.
<tone> may be a value in the range 0 to 56. 0 indicates a pause and the range of 1 to 28 corresponds to the scale from tone C (do) to tone B (ti) which is four octaves higher than the first C. 13 is equivalent to 880 Hz. The range of 29 to 56 corresponds to the scale with tones each of which is a half tone higher than those in the scale represented by 1 to 28.
<duration> must be a value in the range of 0 to 255. When <duration> is specified, the speaker sounds for an interval specified by <duration> multiplied by 0.1 second.

SAMPLE PROGRAMME

```
LIST

100 FOR I=1 TO 68
110 READ A,B
120 SOUND A,B
130 NEXT
140 END
200 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
210 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
220 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
230 DATA 12,8,11,4,10,4,9,8,10,4,9,4,8,3
2
240 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
250 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
260 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
270 DATA 12,8,11,4,10,4,9,8,10,4,9,4,8,2
4
>
```


*STAT

FORMAT STAT [| ALL |]
| <expression> |

PURPOSE To display the status of each programme area.
EXAMPLE STAT 3

REMARKS STAT displays the name and size of a programme stored in each programme area.
<expression> is the programme area number (1 to 5). If <expression> is used, the data in the area specified by the programme area number is displayed. If omitted, the data in the current programme area is displayed. If ALL is specified, the data from all the programme areas are displayed simultaneously along with the size of the RAM file area, MEMSET addresses, and the size of the unused text area (bytes free). However, as the size of the unused text area displayed here includes a work area for BASIC programme execution, it does not necessarily mean that all the unused text area is available for programme storage.

(See MEMSET and TITLE.)

STOP

FORMAT STOP
PURPOSE To terminate programme execution and return to command level.
EXAMPLE STOP

REMARKS STOP statements may be used anywhere in a programme to terminate execution and to return BASIC to command level. When a STOP statement is executed, the following message is output:

Break in nnnn (where nnnn is a line number at which the STOP statement has been executed.)

Unlike the END statement, the STOP statement does not close files. When BASIC has returned to command level, programme execution can be resumed by a CONT command.

(See CONT and END.)

SWAP

FORMAT SWAP <variable 1>,<variable 2>
PURPOSE To exchange the values of two variables.
EXAMPLE SWAP A$, B$

REMARKS With a SWAP statement, any type variable (integer, single precision, double precision, or string) may be swapped, but the two variables must be of the same type or a TM ("Type mismatch") error will occur.
When the type of <variable 2> is a simple variable and the value of the variable is not assigned, an FC ("Illegal function call") error will occur.

SAMPLE PROGRAMME

```
LIST

100 A=123.456
110 B=999999
120 PRINT "A=";A
130 PRINT "B=";B
140 SWAP A,B
150 PRINT
160 PRINT "SWAP!!"
170 PRINT
180 PRINT "A=";A
190 PRINT "B=";B
200 END
RUN
A= 123.456
B= 999999

SWAP!!

A= 999999
B= 123.456
>
```

*TITLE

FORMAT TITLE <programme name>
PURPOSE To name programmes.
EXAMPLE TITLE"TEST 1"

REMARKS TITLE names a programme in the programme area currently LOGged IN. <programme name> may be 8 characters max. Once assigned, this name will appear on the menu when a STAT command is executed or when power is applied to the HX-20.

The programme named by the TITLE statement can be executed directly by menu selection immediately after the power application to the HX-20, and is protected against accidental erasure by a NEW or LOAD command. If you attempt to execute either command in a programme area named by TITLE, a PP ("Protected programme") error will occur.

(See STAT.)

TRON/TROFF

FORMAT TRON
TROFF

PURPOSE To trace the execution of programme statements.

EXAMPLE

REMARKS Execution of a TRON statement (in either the direct or indirect mode) as an aid in debugging enables a trace flag that displays each line number of the programme as it is executed. The numbers appear enclosed in square brackets. The trace flag is disabled when a TROFF statement or NEW command is executed.

SAMPLE PROGRAMME

```
LIST

10 FOR I=1 TO 5
20 PRINT I
30 NEXT
40 END
TRON
RUN
[10][20] 1
[30][20] 2
[30][20] 3
[30][20] 4
[30][20] 5
[30][40]
TROFF
RUN
 1
 2
 3
 4
 5
>
```


*WIDTH

FORMAT **WIDTH <characters per line>, <number of lines> [,<scroll margin>]**

PURPOSE To set the size of the virtual screen

EXAMPLE **WIDTH 20, 25, 5**

REMARKS WIDTH specifies the size of the virtual screen by <characters per line> and <number of lines>. <characters per line> and <number of lines> can be set arbitrarily within the ranges of 20 to 255 and 4 to 255, respectively.
The size of the virtual screen is limited by the capacity of the memory. If you set the screen width too large, an OM ("Out of memory") error may occur. The minimum values are the size of the LCD screen.

<scroll margin> is an allowance for margins that you can provide on the left and right of the display screen, so that you can read the display screen easily when it is moved laterally.
When the cursor comes to the edge of the screen, the next character is outside the display screen and will therefore not be visible. For this reason, when the cursor has moved over a certain set width, it is advisable to move the screen also while the next character is still in view. In this way, the character to the left of the cursor is always on display.
When the cursor has moved without leaving the scroll margin, it indicates that the cursor has reached the end of the virtual screen.

At warm start, the size of the virtual screen is set at WIDTH 40, 8, 3 for the LCD and WIDTH 40, 37, 5 for the external display.
When a WIDTH command is executed, all files opened at that time are closed and all variables, character areas and data defined by DEF statements are cleared.
You may set the size of the virtual screen freely.
When you perform programme corrections, the larger the virtual screen, the more easily you can use it. However, with a very large screen, you may not be able to find the statements at all. Even if statements are being output correctly on the virtual screen, there may be no visible changes in the display screen.

*WIDTH <device name>

FORMAT WIDTH ["LPT0:" / "COM0:"], <number of digits>

PURPOSE To set the print width of the printer.

EXAMPLE **WIDTH "LPT0:", 20**

REMARKS WIDTH sets the maximum value of characters per line that can be output on the printer. Normally, for PRINT statements (including LPRINT and PRINT#) LINE FEED (LF) signal is sent to the printer by a semicolon at the end of each such statement. By specifying the print width with a WIDTH command, LF signal is automatically sent to the printer when <number of digits> which specifies the number of characters to be printed per line is reached.
<number of digits> must have a value in the range of 1 to 255. However, 255 is interpreted as infinite and no automatic line feed according to the specified print width will take place.

NOTE:
At warm start, <number of digits> is set to 80 for the RS-232C port ("COM0:"). Therefore, if a long string exceeding 80 characters is output, line feed is effected automatically upon output of 80 characters. This will also be the result if a device other than the printer is connected to the RS-232C port of the HX-20. For this reason, when using the RS-232C port for communication with an external device, a string longer than 80 characters may not be transmitted properly with <number of digits> set at the default value. When programmes are to be transferred from one HX-20 to another HX-20, first execute WIDTH "COM0:", 255 to disable the automatic line feed operation and then start the transmission.

WIND

FORMAT **WIND [<counter value>]**

PURPOSE To control the microcassette drive for fastforward and rewind.

EXAMPLE **WIND 0**

REMARKS WIND causes the microcassette tape to be fast forwarded or rewound according to <counter value>. <counter value> must be within the range of −32768 to 32767.
The counter value of the microcassette recorder can be read by TAPCNT function.
Once this value is stored in memory, the target file can be searched irrespective of the current tape position by instructing the value with a WIND command.
If <counter value> is omitted, the microcassette recorder rewinds the tape to its beginning and resets the counter value to 0.
When a file is to be searched using a WIND command, you must execute in advance a WIND without <counter value>.

(See TAPCNT.)

SAMPLE PROGRAMME

```
LIST

100 COUNT=TAPCNT
110 OPEN "O",#1,"CAS0:TEST.DAT"
120 FOR A=1 TO 10
130 PRINT#1,A;SQR(A)
140 NEXT
150 CLOSE #1
160 WIND COUNT
170 OPEN "I",#1,"CAS0:TEST.DAT"
175 PRINT :PRINT" A ","SQR(A)"
180 IF EOF(1) GOTO 220
190 INPUT#1,A,B
200 PRINT A,B
210 GOTO 180
220 CLOSE #1
230 END
RUN

 A             SQR(A)
 1              1
 2              1.41421
 3              1.73205
 4              2
 5              2.23607
 6              2.44949
 7              2.64575
 8              2.82843
 9              3
 10             3.16228
>
```


CHAPTER 4
Functions