

SONY®

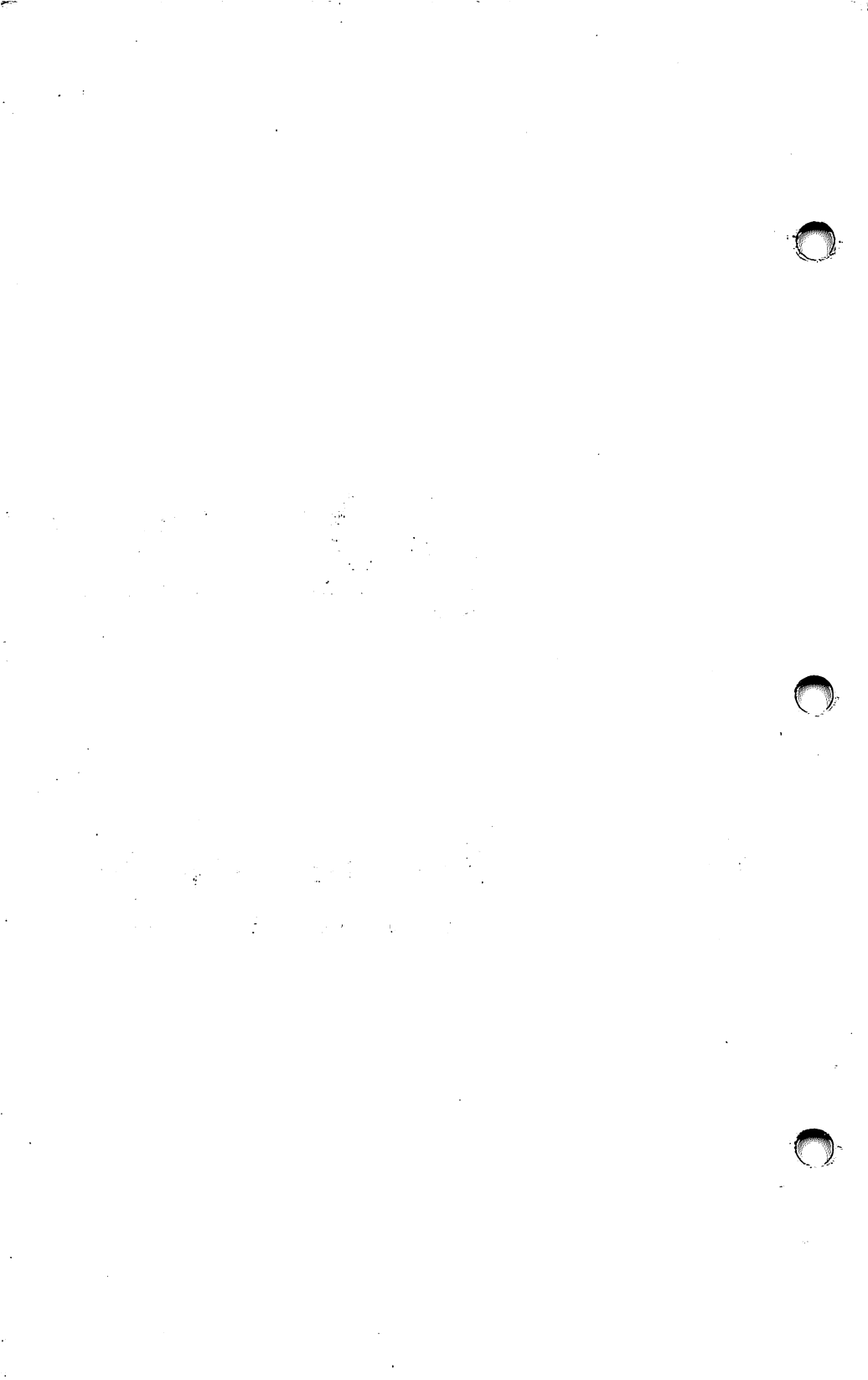
Sony Disk BASIC

Part 2

MICRO COMPUTER

SMC-70

© 1982 by Sony Corporation



Sony Disk BASIC

Part 2 Commands and Functions

PREFACE

The Beginner's All-purpose Symbolic Instruction Code (BASIC) is a programming language developed so that even a beginner can use it after very little study. Sony disk BASIC is a version of BASIC developed for use with the Sony SMC-70 and SMC-70G microcomputer (Hereafter, we call this series of microcomputer "SMC-70" in this manual.) It has many sophisticated commands for programming, such as high-level data processing, table generation, graphic processing and program/data file handling. The Sony disk BASIC programming reference manual is divided into two volumes, Part 1 and Part 2.

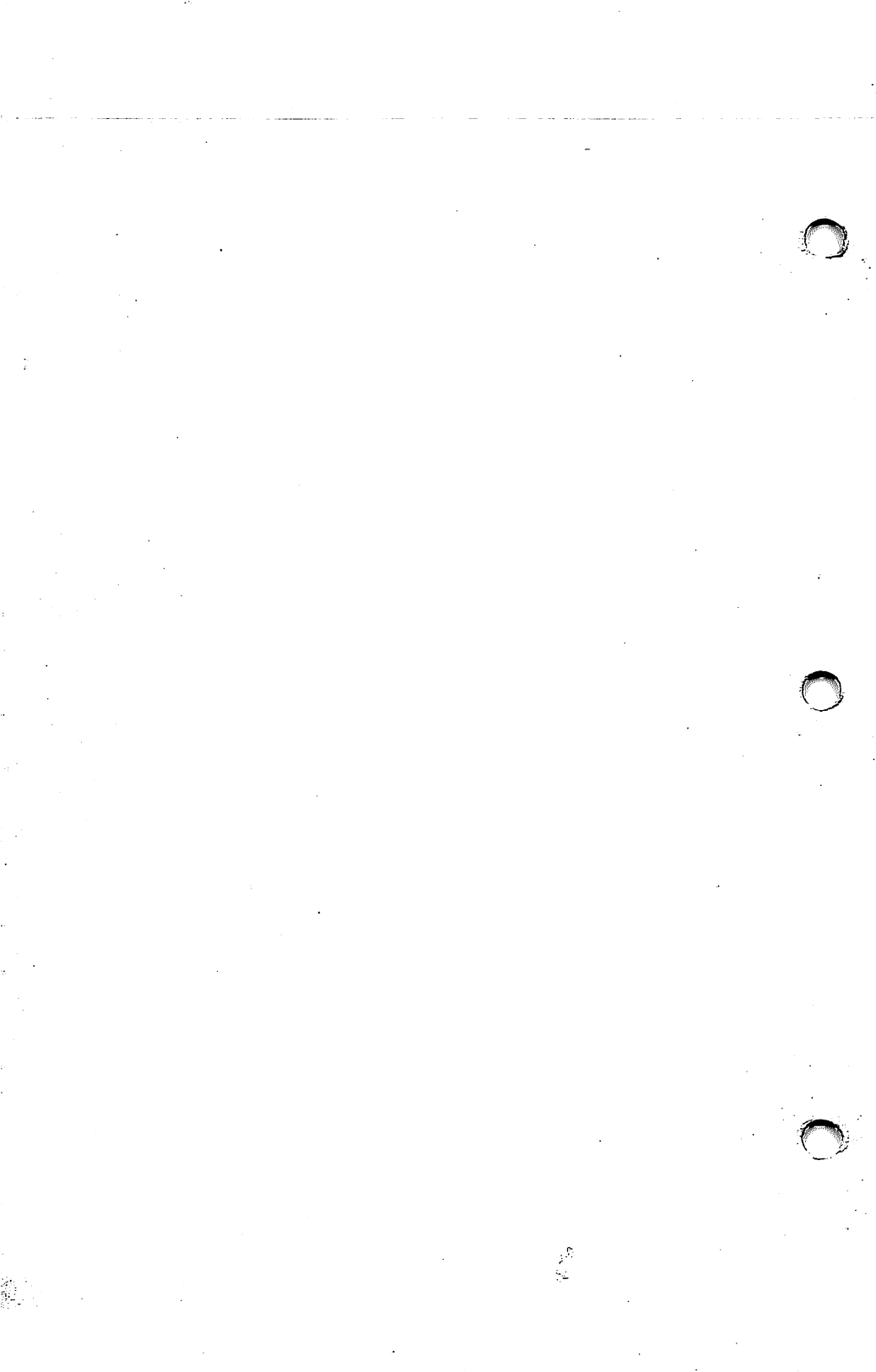
This volume, Part 2, explains all the commands, functions and error messages of Sony disk BASIC. For a general introduction to Sony disk BASIC, refer to the "Part 1. General information". Study these manuals carefully in order to be able to take full advantage of the SMC-70. If you are using BASIC for the first time, you might want to refer to "Sony BASIC: Introductory Manual".

TABLE OF CONTENTS

Chapter 1	Sony disk BASIC commands	5
Chapter 2	Sony disk BASIC functions	237
Chapter 3	Error messages	311

Part 1 (separated)

Chapter 1	Sony disk BASIC outline	5
1-1	Features	6
1-2	General information on Sony disk BASIC	7
Chapter 2	Operation	19
2-1	Start up	20
2-2	Display screen	32
2-3	Debugging the BASIC program	36
2-4	File operation	48
Chapter 3	Program example	59
Appendix		75
1.	Keys to programming	76
2.	Trigonometric and hyperbolic functions	79
3.	Memory configuration	80
4.	FOR to NEXT loop processing and relationship with GOSUB	84
5.	Machine language subroutine	88
6.	Format of the link module	98
7.	I/O port assignment	101
8.	Video RAM address	103
9.	Character/key code table.....	110
10.	Sony disk BASIC reserved word code assignment	111
11.	Color code table	112
12.	Screen map	115



CHAPTER 1

SONY DISK BASIC COMMANDS

This chapter explains the Sony disk BASIC commands in alphabetical order.

The following format is used for explanation :

Command name
Command function

Format

Indicates the correct syntax.

Specifications in brackets [] may be omitted and write only commas. Commas after the last specification may be omitted.

Application

The method for writing a command, functions, and applications are explained in detail.

- The execution examples of the graphic commands are explained in the standard resolution.

Program example

An example of a program using the command is given.

ACLEAR

(attribute clear)

Attributes within the specified range on the character display screen are cleared as specified.

Format

ACLEAR[clear start line], [number of lines to be cleared], [character color code], [background color code], [attribute code]

0 ≤ clear start line ≤ 24	} Numeric-constant, variable, array variable or expressions including these
1 ≤ number of lines to be cleared ≤ 25	
-32768 ≤ character color code ≤ 32767	
-32768 ≤ background color code ≤ 32767	
0 ≤ attribute code ≤ 255	

Application

Specification method

Specify the clear start line and the number of lines to be cleared.

See the Reference Chart for the character color and background color codes and the attribute code specification.

Execution example

Suppose the attribute of the character display screen is as follows :

Character color : White

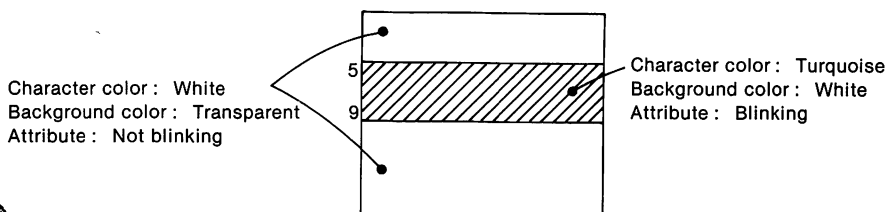
Background color : Transparent

Attribute : Not blinking

Execution of the

ACLEAR 5,5,3,1,2

will set the attribute of the characters in the hatched portion as shown below.



Even if the ACLEAR statement has been executed, the attribute of the characters to be subsequently displayed will remain the same as specified originally with the CCOLOR statement.

When the specification is omitted

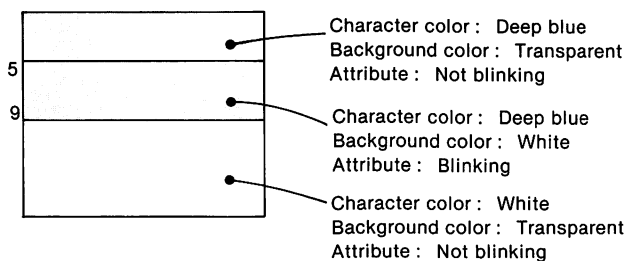
If the “clear start line” is omitted, the uppermost line of the screen is assumed to have been specified.

If the “number of lines to be cleared” is omitted, up to the bottom line of the screen is assumed to have been specified.

If the “character color code”, “background color code” or the “attribute code” is omitted, the codes which have been set in the specified range are assumed to have been specified. For example, if

```
ACLEAR ,10,1
```

is executed after the previous execution example, the attribute of the characters on the screen will be as shown below.



The execution of

```
ACLEAR
```

will not change the character display.

AGET (attribute get)

The attribute information (character color, background color, and attribute codes) in the specified character screen area are stored into the array variable.

Format

AGET (start point coordinates)–(end point coordinates), array variable

$0 \leq X\text{-coordinate} \leq 39 \text{ or } 79$	}	Numeric-constant, variable, array variable, or expressions including these
$0 \leq Y\text{-coordinate} \leq 24$		

array variable : Integer type array variable (The array must have been declared in the DIM statement in advance. Its size is equal to half of the area size.)

Application

The attribute information of all the characters inside the rectangle whose diagonal connects two points specified in the AGET statement is stored into the array variable.

This information can be output again with an APUT statement ; certain attribute information can be repeatedly output.

Specification method

Specify the coordinates on the character display screen.

If the coordinates of a point outside of the displayed screen is specified,

```
Error(8) : Out of screen
```

will be displayed.

Specify an array variable of the integer type . The subscript specified in the array indicates the first variable into which the character code is to be stored : usually specify 0. The array must be declared in the DIM statement in advance.

The attribute information of two characters can be stored into one integer variable (two bytes). The size of the array to be declared is as follows :

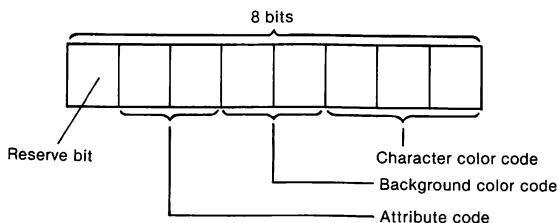
Number of characters in the specified area/2

If the specified value of the coordinates or the subscript is not an integer, the value is rounded to the nearest whole number.



Attribute

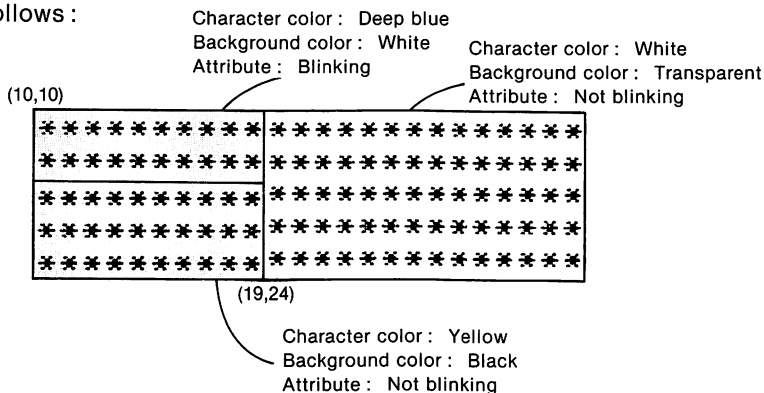
An attribute is a list of the binary character color, background color, and attribute codes. The format is as follows :



See the Reference Chart for the codes.

Execution example

When the characters are displayed on the screen with the attribute as follows :



The array size required to store the attribute information in the above colored rectangle is

$$(10 \times 5)/2 = 25$$

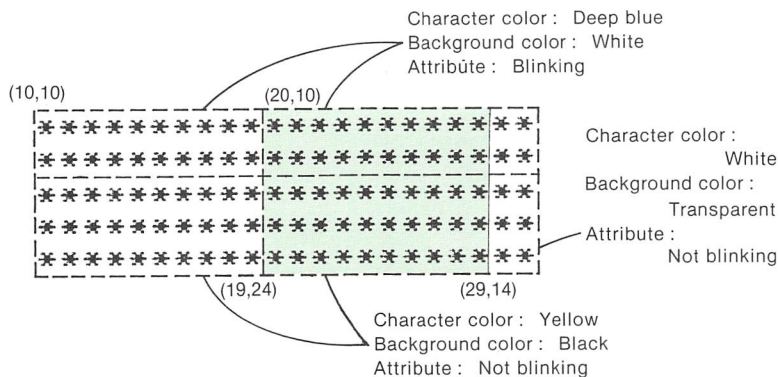
Execute the following to read the information :

```
DIM A%(24)
AGET(10,10)-(19,14),A%(0)
```

When

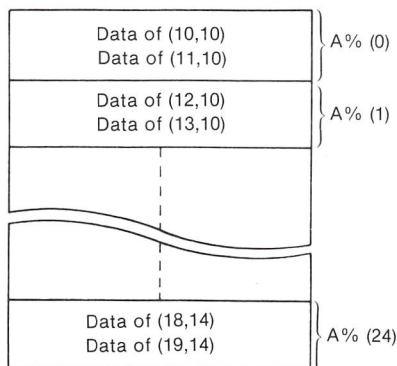
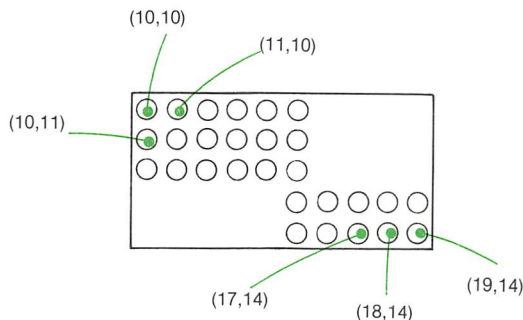
```
APUT(20,10)-(29,14),A%(0)
```


is executed, the characters displayed in the rectangle whose diagonal connects (20,10) and (29,14) will be displayed as follows :



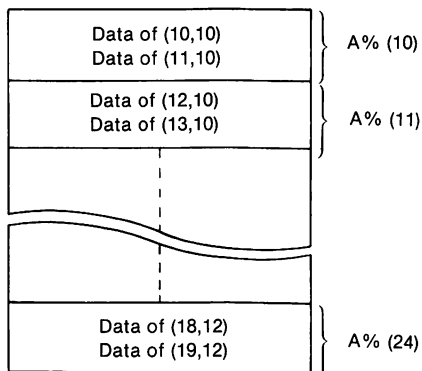
Reading the information

When the AGET (10,10)–(19,14), A%(0) is executed, the attributes of the characters displayed on the screen will be stored in the array specified with the DIM statement as follows :



Specification of array variable subscript

In the array allocated with the DIM statement, the information is sequentially stored into a variable of the specified subscript. When 10 is specified as the subscript, the information is stored as follows:



If DIM A%(24) has been declared, only data of 30 characters can be stored in A%(10) to A%(24).

APUT

(attribute put)

Changes the attribute of characters within a specified area. The attributes are determined by the data stored into the array with an AGET statement.

Format

APUT (start point coordinates)-(end point coordinates), array variable

$0 \leq \text{X-coordinate} \leq 39 \text{ or } 79$

$0 \leq \text{Y-coordinate} \leq 24$

array variable : Integer type variable

} Numeric-constant, variable, array variable,
or expressions including these

Application

An APUT statement changes the attributes of the characters displayed in the rectangle whose diagonal is obtained by connecting two specified points, depending on the attribute data stored in the array with an AGET statement. The attributes are set from the character at the upper-left corner of the rectangle.

Specification method

Specify the coordinates on the character display screen. If the coordinates of a point outside of the displayed screen is specified,

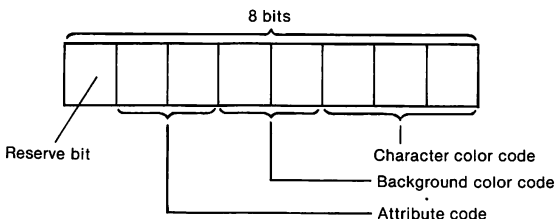
Error(8) : Out of screen

will be displayed.

The array variable is of the integer type used to store data with an AGET statement, and the subscript specifies the variable data to be read first.

Attribute

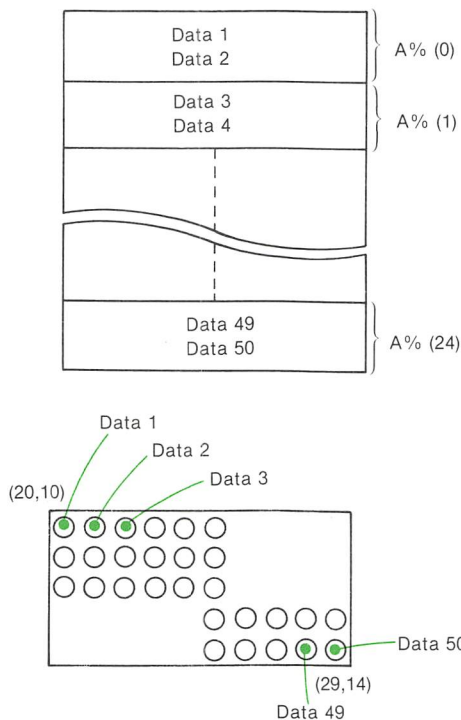
An attribute is a list of the binary character color, background color, and attribute codes. The format is as follows :



See the Reference Chart for the codes.

Information output

When the APUT (20,10)–(29,14), A%(0) statement is executed after AGET statement on page 10, the data in the memory are displayed sequentially from the top left of the area specified, using one byte for each attribute datum.

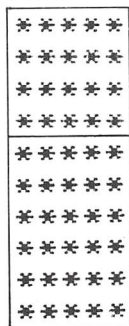


When

APUT(20,15)–(24,24),A%(0)

is then executed, the attribute in the rectangle whose diagonal is obtained by connecting (20,15) and (24,24) will be as follows:

(20,15)



Character color : Deep blue
Background color : White
Attribute : Blinking

Character color : Yellow
Background color : Black
Attribute : Not blinking

(24,24)

Specification of array variable subscript

The variable data to be read first in the array reserved with a DIM statement is indicated in the "array variable". For example, assume that the following are specified :

```
DIM A%(24)
AGET(10,10)-(19,14),A%(0)
APUT(20,10)-(29,14),A%(10)
```

Data beginning with A%(10) is output sequentially.

Since there is only data for 30 characters after subscript 10, the attributes are set only for the characters displayed in the upper 3/5 of the specified rectangle and the lower part remains unchanged.

When the array variable not specified in the AGET statement is used

If the variable is an integer type, each byte datum in the memory is assumed to be an attribute datum. In the case of a floating-point or string-type array variable,

```
Error(3) : Illegal function call
```

will be displayed.

Program example

The following program explains the AGET and APUT command. Pressing a key let the program continue.

```
100 WIPE:GCLEAR 1:DIM A%(24)
110 A$="*****"
120 CCOLOR 1,1,2
130 PRINT (10,10);A$;(10,11);A$;
140 CCOLOR 6,2,0
150 FOR I=12 TO 14:PRINT (10,I);A$;:NEXT
160 CCOLOR 7,0,0
170 FOR I=10 TO 14:PRINT (20,I);A$;"*";:NEXT
180 PRINT (6,9);"(10,10)";(17,15);"(19,14)";
190 PRINT (30,15);"(31,14)";
200 LOCATE(0,0):LIST 210
210 AGET(10,10)-(19,14),A%(0)
220 EVAL INKEY$(1):BEEP
230 LOCATE(0,0):LIST 240
240 APUT(20,10)-(29,14),A%(0)
250 EVAL INKEY$(1):BEEP
260 PRINT (0,1);SPACE$(38);
270 CCOLOR 7,0,0
280 FOR I=15 TO 24
290 PRINT (17,I);A$;
300 NEXT
310 EVAL INKEY$(1):BEEP
320 LOCATE(0,0):LIST 330
330 APUT(20,15)-(24,24),A%(0)
340 PRINT (13,15);"(20,15)";(25,24);"(24,24)";
350 LOCATE(0,0):EVAL INKEY$(1)
```

AUTO (auto)

Line numbers are automatically generated from the specified line number with the specified increment.

Format

AUTO [line number] [,increment]

$$\left. \begin{array}{l} 0 \leq \text{line number} \leq 65529 \\ 1 \leq \text{increment} \leq 65529 \end{array} \right\} \text{Integer constant}$$

Application

This command is used to save the line number key-in operation while entering a program.

Specification method

“line number” indicates the line number to be generated first. “increment” indicates the increment of the line number to be generated next.

Execution example

In the prompt mode, enter

AUTO 100,50 ←

The following will be displayed :

100 

Blinking cursor

Enter the desired statement, and then press **RETURN**.

The following line number will be generated:

```
100 INPUT "A=";A
150 
```

 Blinking cursor

The cursor indicates that the next entry is awaited. Repeat the entry of the desired statement and pressing **RETURN**.

To stop automatic line number generation

Press the **ESC** key ; the prompt mode will be restored.

If the specification is omitted

If "line number" is omitted, line numbers are generated from 10.

If "increment" is omitted, an increment of 10 is assumed to have been specified.

Specification	Line numbers to be generated
AUTO 100,20	100, 120, 140, ...
AUTO 100	100, 110, 120, ...
AUTO ,20	10, 30, 50, ...
AUTO	10, 20, 30, ...

Program editing using AUTO

When a line having the generated line number has previously been stored in the memory, the statement will be displayed, and the cursor will blink at the beginning of the statement. To modify the statement, enter the desired (or modified) statement, and then press .

If the statement is OK, press .

For example, if the following program is stored in the memory,

```
10 FOR I=1 TO 10
20 A=A+I
30 NEXT I
```

and AUTO is executed, the following will be displayed :

```
10 █OR I=1 TO 10
```

If this is OK, press .

To modify the statement, move the cursor to the position(s) to be modified with the and keys, and make the necessary modifications. Then press . The modified statement will be entered, and the next line number will be generated.

In the statement

```
10 FOR I=1 TO 10
```

to change 10 at the end of the statement to 5, move the cursor to 1 and enter 5. Then delete 0 by pressing the space bar before pressing . If the is pressed before deleting the 0, 50 is input instead of 5.

If the lines exist between the generated lines, they will also be displayed.

```
AUTO 10,20
      10 FOR I=1 TO 10
      20 A=A+1
      30 NEXT I
```

Using the cursor move keys, the displayed lines can be modified.

Modification of the line number

The line number can be modified when AUTO is executed.

The line number generation will continue from the modified number.

BEEP

(beep)

A beep is sounded.

Format

BEEP

Application

Execution program

When

BEEP : BEEP : BEEP : BEEP : BEEP : BEEP : BEEP

is executed, a continuous “beep” is heard.

BOXE

(box empty)

A square frame whose diagonal connects two points on the graphic plane is drawn.

Format

BOXE [(starting point coordinates)] - (end point coordinates),
[display color code], [mode]

$-16384 \leq X\text{-coordinate} \leq 16383$
 $-16384 \leq Y\text{-coordinate} \leq 16383$
 $-32768 \leq \text{display color code} \leq 32767$
 $0 \leq \text{mode} \leq 4$

} Numeric-constant, variable, array variable,
or expressions including these

Application

Specification method

To specify the display color code and the mode, refer to the attached Reference Chart.

Execution example

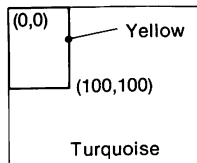
When the entire screen is displayed in turquoise (code 3), execute

```
BOXE(0,0)-(100,100),5,4
```

The current display code 3 and the specified color code 5 are logically operated (XOR of code 4).

Decimal	Binary
3	0011
5	0101
	0110

As a result, a square frame is displayed in yellow (code 6)



When the specification is omitted

“starting point coordinates” will be (X, Y) which were specified last in the preceding graphic command or the current pen position of DRAW statement. If there is no graphic command, (0,0) is assumed to have been specified.

The “display color code” and the “mode” will be the display color and mode previously specified in the last executed GCOLOR statement. If there is no GCOLOR statement, a display color of white is assumed to have been specified and no logical operation is performed.



BOXF

(box full)

A square whose diagonal connects two points on the graphic plane is drawn and its inside is colored.

Format

BOXF [(starting point coordinates)] - (end point coordinates),
[display color code], [mode]

-16384 ≤ X-coordinate ≤ 16383	} Numeric-constant, variable, array variable, or expressions including these
-16384 ≤ Y-coordinate ≤ 16383	
-32768 ≤ display color code ≤ 32767	
0 ≤ mode ≤ 4	

Application

Specification method

To specify the display color code and the mode, refer to the attached Reference Chart.

Execution example

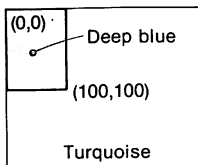
When the entire graphic plane is turquoise (code 3), execute

```
BOXF (0,0)-(100,100),5,2
```

Current display code 3 and the specified color code 5 are logically operated (AND of code 2).

Decimal	Binary
3	0011
5	0101
	0001

As a result, a square is displayed in deep blue (code 1).



BEEP (beep)

A beep is sounded.

Format

BEEP

Application

Execution program

When

BEEP : BEEP : BEEP : BEEP : BEEP : BEEP : BEEP

is executed, a continuous “beep” is heard.

BOXE

(box empty)

A square frame whose diagonal connects two points on the graphic plane is drawn.

Format

BOXE [(starting point coordinates)] - (end point coordinates),
[display color code], [mode]

-16384 ≤ X-coordinate ≤ 16383	} Numeric-constant, variable, array variable, or expressions including these
-16384 ≤ Y-coordinate ≤ 16383	
-32768 ≤ display color code ≤ 32767	
0 ≤ mode ≤ 4	

Application

Specification method

To specify the display color code and the mode, refer to the attached Reference Chart.

Execution example

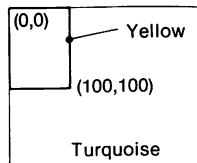
When the entire screen is displayed in turquoise (code 3), execute

```
BOXE(0,0)-(100,100),5,4
```

The current display code 3 and the specified color code 5 are logically operated (XOR of code 4).

Decimal	Binary
3	0011
5	0101
	0110

As a result, a square frame is displayed in yellow (code 6)



When the specification is omitted

“starting point coordinates” will be (X, Y) which were specified last in the preceding graphic command or the current pen position of DRAW statement. If there is no graphic command, (0,0) is assumed to have been specified.

The “display color code” and the “mode” will be the display color and mode previously specified in the last executed GCOLOR statement. If there is no GCOLOR statement, a display color of white is assumed to have been specified and no logical operation is performed.



BOXF

(box full)

A square whose diagonal connects two points on the graphic plane is drawn and its inside is colored.

Format

BOXF [(starting point coordinates)] - (end point coordinates),
[display color code], [mode]

$-16384 \leq \text{X-coordinate} \leq 16383$
 $-16384 \leq \text{Y-coordinate} \leq 16383$
 $-32768 \leq \text{display color code} \leq 32767$
 $0 \leq \text{mode} \leq 4$

} Numeric-constant, variable, array variable,
or expressions including these

Application

Specification method

To specify the display color code and the mode, refer to the attached Reference Chart.

Execution example

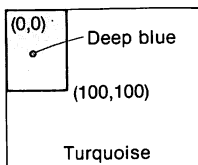
When the entire graphic plane is turquoise (code 3), execute

```
BOXF(0,0)-(100,100),5,2
```

Current display code 3 and the specified color code 5 are logically operated (AND of code 2).

Decimal	Binary
3	0011
5	0101
	0001

As a result, a square is displayed in deep blue (code 1).



When the specification is omitted

The “starting point coordinates” will be (X,Y) which were specified last in the preceding graphic command or the current pen position of the DRAW statement. If there is no graphic command, (0,0) is assumed to have been specified.

The “display color code” and the “mode” will be the display color and mode previously specified in the last executed GCOLOR statement. If there is no GCOLOR statement, a display color of white is assumed to have been specified and no logical operation is performed.



CALL (call)

A machine language subroutine is called.

Format

CALL subroutine name [(data, data,)]

data : Numeric-constant, variable, array variable, or expressions including these

Application

Specification method

When control is transferred to the machine language subroutine, assign the starting address of the subroutine to a subroutine name, and then execute the CALL statement.

If the subroutine is loaded with a LINK command, specify the subroutine name determined at file generation. For a subroutine loaded from a ROM pack, specify the subroutine name indicated on the ROM pack manual.

Execution example

To execute the subroutine written from hexadecimal address C000, execute as follows :

```
SUB=&HC000  
CALL SUB
```

When values are transferred to subroutine

Specify the desired variables and constants to be transferred after the "subroutine name" in the CALL statement. Up to 10 values can be transferred. (See "Appendix 5 Machine language subroutine" in Part 1 for details on value transfer.)

Use of underline (—)

An underline can be used instead of CALL.

```
CALL SUB  
_SUB
```

The above two statements have the same meaning.

CCLEAR (character clear)

Characters within the specified range are cleared from the character display screen.

Format

CCLEAR [clear start line] [,number of lines to be cleared]

$0 \leq \text{clear start line} \leq 24$ } Numeric-constant, variable, array
 $1 \leq \text{number of lines to be cleared} \leq 25$ } variable, or expressions including these

Application

Specification method

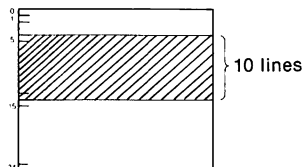
Characters displayed in the lines specified by “number of lines to be cleared” from “clear start line” will be cleared. The upper-most line of the screen is the line 0.

Execution example

Execute

CCLEAR 5,10

Characters in the hatched portion will be cleared.



When the specification is omitted

If the "clear start line" is omitted, the characters will be cleared from the uppermost line of the screen.

If the "number of lines to be cleared" is omitted, characters will be deleted up to the bottom line of the screen. If both are omitted, all characters on the screen will be cleared.

Note

"Clear" means that the character display screen in the specified range is colored with the background color specified in the CCOLOR statement.

When the following is specified

```
10 CCOLOR 7,0,0
20 CCLEAR
```

all characters will be deleted, and the character display screen will become transparent.

If the following is specified

```
10 CCOLOR 1,1,0
20 CCLEAR
```

the entire screen will become white.

In the direct command mode, CCOLOR 7,0,0 is always set.

CCOLOR

(character color)

Colors of characters and background on the character display screen and the display method are specified.

Format

CCOLOR[character color code], [background color code], [attribute code]

$-32768 \leq \text{character color code} \leq 32767$
 $-32768 \leq \text{background color code} \leq 32767$
 $0 \leq \text{attribute code} \leq 255$

} Numeric-constant, variable, array
variable, or expressions including these

Application

Specification method

The "character color code" specifies the character color, the "background color code" specifies the background color, and the "attribute code" specifies the display method.

Specify the codes according to the attached Reference Chart.

Execution example

Execute

```
10 CCOLOR 3,1,2
20 PRINT "ABC"
```

Turquoise characters blink on a white background.

This state is maintained until the next CCOLOR or WIPE statement is executed in a program or the prompt mode is reentered.

When the specification is omitted

The values specified in the preceding CCOLOR statement are assumed to have been specified.

For example, if

```
CCOLOR , , 0
```

is executed after execution of

```
CCOLOR 3,1,2
```

turquoise characters will be displayed on a white background without blinking.

Initial set upon entry of Sony disk BASIC

character color code : 7 (white)

background color code : 0 (transparent)

attribute code : Character and background colors are as specified. Not blinking.

In the direct command mode, CCOLOR 7,0,0 is always set and the character colors cannot be changed. When a key is pressed, the corresponding character is displayed in turquoise. When the **RETURN** key is pressed, the color is changed to white.

Note

- The CCOLOR statement can make different specifications for each character.

```
10 FOR I=0 TO 7
20 CCOLOR I,0,0
30 PRINT "A";
40 NEXT I
RUN
```

AAAAAAA

White
Yellow
Brilliant pink
Red
Turquoise
Bright green
Deep blue
Black

- When a monochrome monitor TV is used, the brightness of the displayed characters differs according to the specified color.

(character get)

The character codes in the specified character screen area are stored into the array variable.

Format

CGET (start point coordinates)–(end point coordinates), array variable

$$0 \leq \text{X-coordinate} \leq 39 \text{ or } 79$$

Numeric-constant, variable, array variable,
or expressions including these

$$0 \leq Y\text{-coordinate} \leq 24$$

array variable: Integer type array variable (The array must have been declared in the DIM statement in advance. Its size is equal to the half of the area size.)

Application

The character code information of all the characters inside the rectangle whose diagonal connects two points specified in the CGET statement is stored into the array variable.

This information can be displayed again with a CPUT statement ; certain characters can be repeatedly displayed.

Specification method

Specify the coordinates on the character display screen.

If the coordinates of a point outside of the displayed screen is specified,

Error(B) : Out of screen

will be displayed.

Specify an array variable of the integer type. The subscript specified in the array indicates the first variable into which the character code is to be stored: usually specify 0. The array must be declared in the DIM statement in advance.

The codes of two characters can be stored into one integer variable (two bytes). The size of the array to be declared is as follows :

Number of characters in the specified area/2

If the specified value of the coordinates or the subscript is not an integer, the value is rounded to the nearest whole number.

Execution example

When the following characters are displayed on the screen :

(10,10)  (19,14)

The array size required to store the character code information in this rectangle is

$$(10 \times 5)/2 = 25$$

Execute the following to read the information :

```
DIM A%(24)
CGET(10,10)-(19,14),A%(0)
```

When

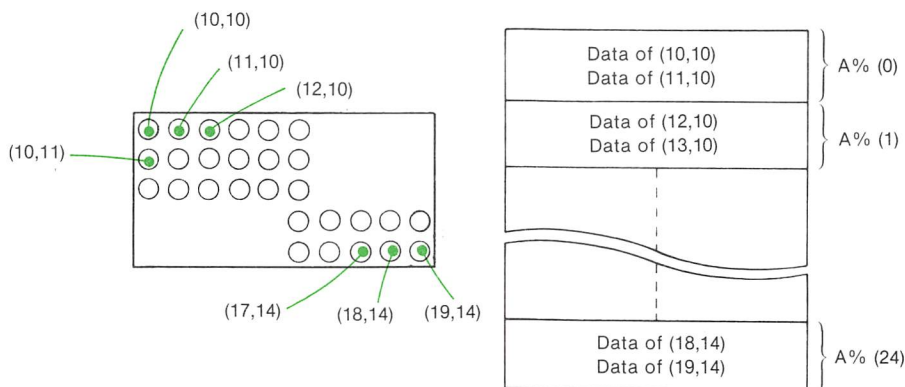
```
CPUT(20,10)-(29,14),A%(0)
```

is executed, the following will be displayed :

(10,10)	(20,10)
#####	#####
#####	#####
#####	#####
#####	#####
#####	#####
(19,14)	(29,14)

Reading the information

When CGET (10,10)-(19,14), A%(0) is executed, the character codes of the characters displayed on the screen will be stored in the array area specified with the DIM statement as follows :





CHAIN (chain)

Loads the specified Sony BASIC program, then executes it.

Format

CHAIN [/V] [/M] "[device name:] file name [.type]",
[line number], [DELETE [line number] [-] [line number]]

device name : DSK (disk), CTR (cassette tape), PAC (ROM pack) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of eight or fewer characters

0 ≤ line number ≤ 65529 : Numeric constant or a label specifying a line

Application

Specification method

To load a program from the currently logged in disk, and run the program, execute

CHAIN "[DSK:] file name [.type]"

If DSK is omitted, the result will be the same.

In the device name, specify DSK, CTR, PACK or CHN depending on the device from which the program is to be loaded. When a disk which is not in the current disk drive is to be specified, write the drive number, for example "B:", instead of "DSK:".

Specify the same file name and type specified when the program was saved.

When the CHAIN command is executed, the memory contents, and the program and the variable values are cleared, then the program is loaded.

Execution example

When

```
CHAIN "PROG1.BAS"
```

is executed, the current memory contents are cleared, the program (PROG1.BAS) is loaded into memory and it is executed from the first line.

To pass the variable value to the loaded program

Execute

CHAIN/V "[device name:] file name [.type]"

The program in memory is deleted, but the variable values are maintained. The specified program is loaded from the specified device and is executed with the variable value retained.

To merge a program and execute it

Execute

CHAIN/M "[device name:] file name [.type]"

The specified program on the specified device is merged into the program in memory. Then the merged program is executed.

The specified program should be in the ASCII format.

To specify the line number from which the loaded program is to be executed

Specify the line number in the CHAIN command as follows :

CHAIN [/V] [/M] "[device name:] file name [.type]", line number

For example, if

```
CHAIN " PROG .BAS " , 100
```

is executed, the PROG. BAS is loaded and it will be executed from line 100.

When the line number specification is omitted, execution starts from the first line.

If the CHAIN statement is in a program in memory, when this program is renumbered, the specified line number is not renumbered.

To delete some lines of the program in memory, merge the program, then execute it

Specify the DELETE command in a CHAIN command as follows :

CHAIN/M "[device name:] file name [.type]", line number,

DELETE line number-line number

The lines specified in DELETE are deleted from the program in memory, the specified program is merged, then the merged program is executed.

Specification method of the line numbers in the DELETE is the same as that in a DELETE command. See "DELETE" for details. When a CHAIN statement is in the program in memory and when this program is renumbered, the specified line numbers in DELETE are also renumbered.

When the specification of type is omitted

".BAS" is assumed to have been specified.

CIRCLE (circle)

Draws a circle or an oval on the graphic display screen.

Format

CIRCLE (coordinates), [radius in X-direction], [radius in Y-direction], [color code], [mode]

$-16384 \leq \text{X-coordinates} \leq 16383$
 $-16384 \leq \text{Y-coordinates} \leq 16383$
 $-32768 \leq \text{color code} \leq 32767$
 $0 \leq \text{mode} \leq 4$

Numeric-constant, variable, array variable, or expression including these

Application

Specification method

Specify the coordinates of the center position in a circle or an oval to be drawn on the graphic display screen.

The length in the direction of x or y axis of the circle or an oval will be twice as long as the specified radius in x or y-direction.

See the Reference Chart for the specification of color code and mode.

Execution example

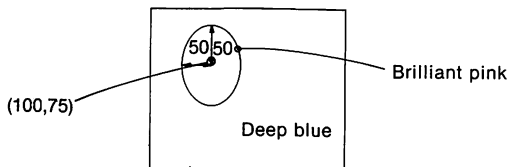
When the entire graphic screen is deep blue, execute

```
CIRCLE (100,75),50,50,4,3
```

The current display color code 1 and the specified color code 4 will be logically operated (OR of code 3)

Decimal	Binary
1	0001
4	0100
	0101

As a result, the brilliant pink will be selected for the graphic display. Because there is a difference in length between the height and width of one dot, an oval as shown below will be displayed.



To draw a circle

Omit either radius in X-direction or Y-direction.

When the specification is omitted

If the specification of the either radii is omitted, a circle will be drawn with the specified radius.

For example, execution of

```
CIRCLE (100,75),50,,3
```

draws a circle with a radius in x-direction of 50, and execution of

```
CIRCLE (100,75),,50,3
```

draws a circle with a radius in y-direction of 50. As one dot is longer in its height, the second statement draws a bigger circle.

If both of the specifications are omitted,

```
Error(130) : Syntax error
```

will be displayed.

The display color code and the mode, if omitted, will be the display color and mode previously specified in the last executed GCOLOR statement. If there is no GCOLOR statement, a display color of white is assumed to have been specified and no logical operation is performed.

When 0 is specified as a radius

When the radius in X-direction is 0, a line parallel to the Y-axis is drawn. When the radius in Y-direction is 0, a line parallel to the X-axis is drawn.

When both of the radii are 0's, a dot is plotted at the point specified by the coordinates.



CLEAR

(clear)

All numeric variable values are set to 0s. String variable values are set to null strings, and array variables are set in the undefined state. All opened files on the disk are closed.

Format

CLEAR

Application

This statement is used to initialize all variables.

If this statement is executed, the program currently stored in the memory will be held.



CLOAD

(cassette load)

A Sony BASIC program file is loaded into memory from the specified device.

Format

CLOAD [/V] "[device name:] file name [.type]"

device name : DSK (disk), CTR (cassette tape), PAC (ROM pack) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

Application

The CLOAD command is used in the same way as the LOAD command.

It is only different that the cassette tape is selected when the device name specification is omitted.

CLOSE

(close)

The specified data files, opened with OPEN statements, are closed.

Format

CLOSE [#file number] [,#file number]...

$0 \leq \text{file number} \leq 127$ } Numeric-constant, variable, array variable,
 } or expressions including these

Application

The files opened with OPEN statements must be closed with a CLOSE statement when data write/read operation is completed.

If the files are not closed, the file contents may be destroyed.

If the specification of the file numbers are omitted, all opened files are closed.

CONSOLE (console)

The number of characters per line displayed on the screen, display range, and scroll ON/OFF are specified.

Format

CONSOLE [number of characters per line], [character display start line], [number of character display lines], [scroll mode]

number of characters per line : 40 or 80

$0 \leq \text{character display start line} \leq 24$

$1 \leq \text{number of character display lines} \leq 25$

scroll mode : 0 or 1

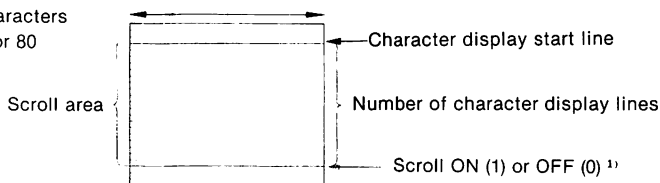
Numeric-constant, variable, array variable, or expressions including these

Application

Specification method

Specify as follows :

Number of characters
per line : 40 or 80



When the specified values are not integers, the values are rounded to the nearest whole number.

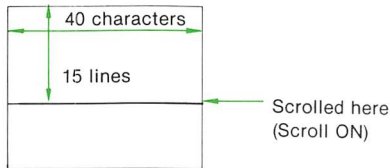
¹⁾ When the last line in the scroll area is reached, characters in the uppermost line will be cleared, and the entire screen will be scrolled one line space upward. If scroll OFF is specified, characters will be displayed up to the last line in the scroll area, and the cursor will move to the left end of the last line.

Execution example

Execute

CONSOLE 40,0,15,1

The display will be made as follows :



Screen display after execution

When the number of characters per line is changed, all the characters on the screen will be cleared, and the cursor will blink in the upper-left corner of the specified scroll area. When the number of characters per line is not altered, all the characters on the screen will be displayed as they are, and the cursor will blink in the upper-left corner of the specified scroll area.

When the specification is omitted

The following values are assumed to have been specified :

number of characters per line : Preceding value
character display starting line : 0
number of display lines : Up to the bottom line
scroll mode : ON

For example, if

CONSOLE ,5

is executed, the number of characters per line does not change and the scroll area will be below line 5. If

CONSOLE , ,5

is executed, the scroll area will be from the top line to line 4. To specify 80 characters per line, specify

CONSOLE 80

In this case, the entire screen will be the scroll area.

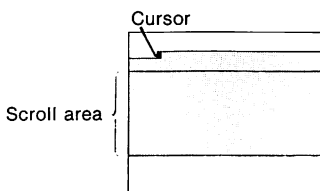
Initial set upon entry of Sony disk BASIC

number of characters per line :	80	
character display start line :	Line 0	} (Scroll area : Entire screen)
number of character display lines :	25	
scroll mode :	ON	

When the cursor is moved outside the scroll area

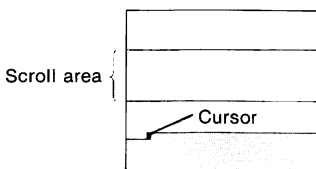
The cursor can be moved outside the scroll area with a program. At this time, characters will be displayed on the screen as follows :

When the cursor is moved above the scroll area



The cursor moves as usual every time characters are displayed on the screen. If characters are continuously output until the cursor reaches the lower-right corner of the scroll area, the characters will be displayed only in the scroll area in the specified scroll mode (ON or OFF).

When the cursor is moved beneath the scroll area



The cursor moves as usual every time characters are displayed on the screen. If characters are continuously output until the cursor reaches the lower-right corner of the screen, the characters will be displayed in the scroll OFF mode.

Note

- In the direct command mode, the entire screen is always set for the scroll area.



CONT

(continue)

The program interrupted with a STOP statement or **ESC** key is restarted.

Format

CONT

Application

Execute the CONT statement in the direct command mode when program execution is interrupted with a STOP statement or **ESC** key. Program execution restarts from the statement following the interrupted statement; however, if program execution is interrupted with an **ESC** key during execution of a PLAY, DRAW or INPUT statement, program execution will restart from the beginning of the statement.

- When a CONT statement is executed in a program or when program execution is interrupted with other than a STOP statement or **ESC** key, "Can't Continue" will be displayed.

Note

To restart program execution from the desired line, execute a RUN or GOTO statement in the direct command mode.

CPLLOT

(character plot)

Displays a character at the specified position on the character display screen.

Format

CPLLOT (coordinates), [character code], [character color code],
[background color code], [attribute code]

0 ≤ X-coordinate ≤ 39 or 79	}	Numeric-constant, variable, array variable, or expressions including these
0 ≤ Y-coordinate ≤ 24		
0 ≤ character code ≤ 255		
-32768 ≤ character color code ≤ 32767		
-32768 ≤ background color code ≤ 32767		
0 ≤ attribute code ≤ 255		

Application

Execution example

When

```
CPLLOT(10,10),&H41,1,1,2
```

is executed, the character A will be displayed blinking at the position of (10, 10) in deep blue on a white background.

When the specification is omitted

The character color code, the background color code and the attribute are set to those specified in the last executed CCOLOR statement. If there has been no CCOLOR statement, the following codes are assumed to have been specified:

- 7: for character color
- 0: for background color
- 0: for attribute

When the specification of the character code is omitted, the attribute of the character at the specified position is changed as specified.

CPUT

(character put)

Displays characters on the character display screen. The characters to be displayed are specified in the data stored in an array with a CGET statement.

Format

CPUT (start point coordinates)–(end point coordinates), array variable

$0 \leq \text{X-coordinate} \leq 39$ or 79

Numeric-constant, variable, array variable,
or expressions including these

$0 \leq \text{Y-coordinate} \leq 24$

array variable : Integer type array variable

Application

A CPUT statement displays the character code information, which has been stored in the array with a CGET statement, beginning at the upper-left corner of the rectangle whose diagonal is obtained by connecting two specified points.

Specification method

Specify the coordinates on the character display screen. If the coordinates of a point outside of the displayed screen is specified,

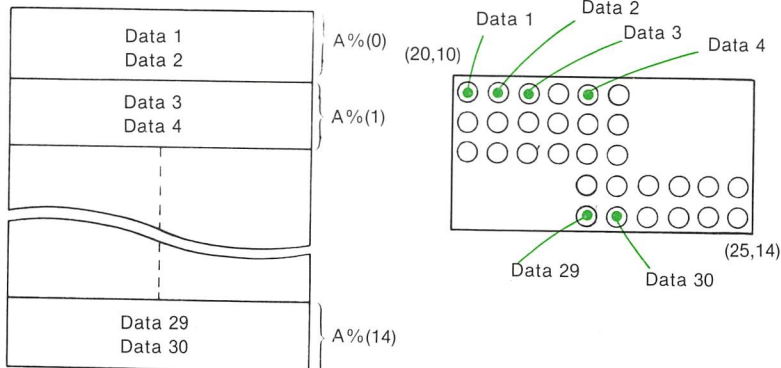
Error(8) : Out of screen

will be displayed.

The array variable is of the integer type used to store data with a CGET statement, and the subscript specifies variable data to start the display.

Information output

When the CPUT statement is executed, data in the memory are displayed sequentially from the top left of the area specified, using one byte for each character code.



When

CPUT(0,0)-(24,1),A%(0)

is executed after CGET statement in the execution example on page 30, characters A and B will be displayed as follows.

(0,0)

```
#####A#####A#####  
#####A#####A#####
```

(24,1)

Specification of array variable subscript

Specify the subscript of the array variable from which you are to start displaying data.

For example, assume that the following are specified :

```
DIM A%(14)  
CGET(10,10)-(15,14),A%(0)  
CPUT(20,10)-(25,14),A%(10)
```

Data beginning with A%(10) is output sequentially.

Since there are only data for 10 characters after subscript 10, the characters are displayed only in the upper 1/3 of the specified rectangle while the lower part remains unchanged.

When the array variable not specified in the CGET statement is used

If the variable is an integer type, each byte of datum in memory is assumed to be a character code. In the case of a floating-point or string-type array variable,

Error(3) : Illegal function call

will be displayed.

CSAVE

(cassette save)

The Sony BASIC program in memory is saved on to the specified device.

Format

CSAVE [/A or /P] [/C] "[device name:] file name [.type]"
[start address, end address]

device name : DSK (disk), CTR (cassette tape), CON (console), LST (list device) or
CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

$0 \leq \text{start address, end address} \leq 65535$: Numeric-constant, variable, or expressions including these

Application

The CSAVE command is used in the same way as the SAVE command. It is only different that the cassette tape is selected when the device name specification is omitted.

CURSOR

(cursor)

The cursor is displayed or erased.

Format

CURSOR ON or CURSOR OFF

Application

This command is useful to erase the cursor when a graphic display is made.

- Even if CURSOR OFF is executed, the cursor will be forcibly displayed when the program ends or is interrupted with the `ESC` key or `STOP` statement and the prompt mode is reentered. The cursor will also be forcibly displayed when an `INPUT` statement is executed, but after the execution, it will be erased again.



DATA

(data)

The data to be read is specified with constants.

Format

DATA constant [,constant] [,constant]...

constant : Numeric or string

Application

In the DATA statement, specify the data corresponding to the variables in the READ statement.

Specification method

To specify two or more data values in a DATA statement

Separate data items with a comma(s).

```
DATA 1,2,3
```

(Three numeric or string data items)

```
DATA A,B
```

(Two string data items)

To specify two or more DATA statements

Two or more DATA statements can be specified for the READ statements if numbers, sequences, and types of variables and corresponding data match.

For example, READ statements are written as follows :

```
10 READ A,B,C$,D$
```

```
·  
·  
·
```

```
50 READ E$,F$
```

DATA statements can be written as follows :

```
100 DATA 50,70,S,T,U,V
```

or

```
5 DATA 50,70
```

```
6 DATA S,T,U,V
```

When string data contains a comma (,) or space at the beginning or the end of the string

enclose the data in quotation marks ("").

DATA A	(String data)
DATA 5	(Numeric or string data)
DATA "A,B"	(String data containing a comma)
DATA A	(String data. The space is ignored.)
DATA " A"	(String data containing a space)
DATA 12 34	(String data containing a space)

When a DATA statement is written in the IF statement or in the direct command mode

The data statement is ignored. If there is no data item in a DATA statement, the DATA statement will be ignored.

Note

See the explanation of the READ command for the DATA statement data read sequence, type matching, and so forth.

DEBUG

(debug)

Debug a program in the debug mode.

Format

DEBUG [line number]

Application

This command is not included in the Sony disk BASIC interpreter, but supplied as a separate file with the name of "DEBUGGER.PAC" on the disk.

To use the DEBUG command, first load the "DEBUGGER.PAC" file with the LINK command. The program to be debugged may be loaded before or after the "DEBUGGER.PAC" is entered. Once this file is loaded, the DEBUG command is enabled.

Details on loading and using the DEBUG command is explained in "2-3 Debugging the BASIC program" in Part 1.

Note

Be sure to use a LINK command to load the "DEBUGGER.PAC" file. If CLOAD or LOAD command is executed for the file, an error will occur and the file is not loaded.

Command list

Following sub commands are operative under the DEBUG command so that errors in the program to be debugged are effectively found.

Command name	Entering format	Meaning
C	Clear	Clears all the variables.
D	Debug [line number]	Initializes the debugger.
E	Edit [line number]	Corrects the line.
G	Goto [start line No.] [/break line No.] [:statement number] [(number of repetitions)]	Executes a program between specified lines or up to the specified statement
L	List [start line number] [-] [end line number]	Displays a program list.
P	Print [@device number,] { variable } { constant }	Outputs the specified constant or variable to the desired device.
Q	Quit { Y } { N }	Terminates the debug mode by entering "Y".
R	Run [start line number]	Executes the program.
S	Step [number of statements]	Executes the program by specified number of statements
T	Trace [number of lines]	Executes the program by specified number of lines.
U	Undefined variable detection { Y } { N }	Sets or resets the undefined variable detection which causes an error when a variable which is not assigned to any value appears.
{ 1 } { 2 }	PRINT item to be printed	Executes the specified statement each time a statement in the program is executed.

- Either of the two in { } must be selected.
- Of these commands, the C, E, P, and R commands can be used exactly in the same way as the ordinary Sony disk BASIC commands.
- With the L command, the same result as the LIST command will be obtained ; however, a screen editing function is not operative on the program listed by L command.
- Editing must be effectuated with the E (Edit) command.
- If the space bar is pressed when the command menu is displayed, you can enter a statement in the same way as in the direct command mode.

DEF ANN

(define ANN)

The starting address to call the machine language subroutine is defined.

Format

DEF ANN [X] = address

$0 \leq X \leq 9$

$0 \leq \text{address} \leq 65535 \text{ (&HFFFF)}$

} Integer-constant, variable or expressions
including these

Application

Specification method

Specify a number from 0 to 9 as X. By specifying X, up to 10 subroutines may be defined.

Specify the start address of the subroutine to be defined as the “address”.

Execution example

Execute

```
DEF ANN1=&HC000
```

The machine language subroutine beginning with address &HC000 is defined as ANN1.

“ANN” is a function to give the subroutine execution result.

When the specification of X is omitted

X = 0 is assumed to be specified.

Note

See “Appendix 5 Machine language subroutine” in Part 1 for details on the uses of the DEF ANN statement and ANN function.

DEF FN

(define function)

Defines a user function.

Format

DEF FN function name [(parameters)] = expression

Application

Once an expression is defined as a function with a DEF FN statement, the result of the expression can be obtained by calling the defined function with the function name.

Specification method

The function name can be any numeric or string type variable name if the type of the function name and the value of the expression match. The parameters are a list of variables, separated by commas.

Up to ten parameters can be specified.

When the function is called, the values for the parameters listed will be given to the defined function, and the result will be returned.

The expression may be a numeric or string type constant, variable, array variable or expressions including these. The variables in the expression may or may not be the variables in the parameters. When the function is called, the value listed in the parameter list are assigned to the variables in the expression of the same name. To the other variables in the expression, the current values in the program are assigned.

Execution example

This is to get the temperature in Fahrenheit when the temperature in centigrade is given.

```
10 DEF FNF(C)=9/5*C+32
20 PRINT FNF(30);" F"
30 PRINT FNF(0);" F"
RUN
86 F
32 F
```

The value of the parameter C is given when the function is called. The value will be given to the function definition and the result will be returned as the function value.

To get the average of the five input data, execute the following ;

D

```
10 DIM A(5)
20 DEF FNA=(A(1)+A(2)
  +A(3)+A(4)+A(5))/5
30 FOR N=1 TO 5
40 INPUT A(N)
50 NEXT N
60 PRINT FNA
```

In this program, the defined function has no parameters. The variable values in the program are assigned to the variable in the defined function and the result will be obtained.

When this program is executed,

?

will be displayed. Enter the desired value and again

?

will be displayed.

When 5 values have been entered, the average value will be displayed.

```
?10
?20
?30
?40
?50
30
```

Definition of the user function

The DEF FN statement must have been executed before the user function is called. If a function is called before the definition,

Error(148) : Undefined user function

will be displayed.

Variable in the parameter list

They will not affect the variables of the same name in the program and vice versa.

```
10 A=5:B=3
20 DEF FNF(A)=A+B
30 PRINT FNF(10)
40 PRINT A+B
RUN
13
8
```

In line 30, 10 is assigned to variable A in the function and the result is obtained. In line 40, 5 is assigned to variable A and the result is obtained.

Execution in the direct command mode

If a DEF FN statement is executed in the direct command mode,

```
Error(3) : Illegal function call
```

will be displayed.

If a DEF FN statement has been executed in a program, the defined function can be called in the direct command mode.

Recursive call of the user function

The function to be defined can be written in the expression of the DEF FN statement, which we call "recursive call".

The following program is to get a factorial value of an input datum.

```
10 DEF FNFACT(N)=COND(N=0,1,N*FNFACT(N-1))
20 INPUT N
30 PRINT FNFACT(N)
40 GOTO 20
```

The function FNFACT(N) is defined to get a factorial value of N.

When $N=0$, $\text{FNFACT}(0)=1$

When $N \neq 0$, $\text{FNFACT}(N)=N * \text{FNFACT}(N-1)$

For example, when 5 is entered as a value of N

```
FNFACT(5)=5* FNFACT(4)
          =5* 4* FNFACT(3)
          =5* 4* 3* FNFACT(2)
          =5* 4* 3* 2* FNFACT(1)
          =5* 4* 3* 2* 1* FNFACT(0)
          =5* 4* 3* 2* 1* 1
```

In this way, the factorial value of 5 is obtained.

DEF FONT

(define font)

A font is defined.

Format

DEF FONT character code, "font data"

$0 \leq \text{character code} \leq 255$: Numeric-constant, variable, array variable, or expressions including these

font data: String constant, variable, array variable, or expressions indicating a font

Application

Specification method

The "character code" specifies the code whose font is to be redefined. See "Appendix 9 Character/key code table" in Part 1 for the character codes.

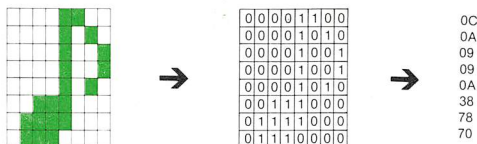
The "font data" specifies which dots are marked in a character area (8 vertical dots $\times$ 8 horizontal dots). A dot to be marked is represented by 1 and a dot not to be marked is represented by 0. Each horizontal 8-dot line is converted to a hexadecimal number, and eight hexadecimal numbers are arranged from the top one.

When the font data is a null string, the previously defined font is maintained.

Execution example

Let us define an eighth note (♪) for character code 126.

The font data is as follows:



Execute

```
DEF FONT 126, "0C0A09090A387870"
```

Then press the  key while pressing the  key, or execute

```
PRINT CHR$(126)
```

The eighth note will be displayed.

Maintaining the defined font

The defined font is maintained until it is redefined in a DEF FONT statement for the same character code, a FONT RESET statement is executed or the computer is reset.

DEF INT

(define integer)

DEF FLT

(define floating-point)

DEF STR

(define string)

A type of the variable beginning with a specified character is declared as an integer (INT), floating-point (FLT), or string (STR) type.

Format

```
DEF INT }  
DEF FLT } character[-character]  
DEF STR }
```

character: An alphabetic character

Application

Specification method

Specify an alphabetic character or a character range with starting and ending characters.

Execution example

Execute

```
DEF INT A-D
```

Variables beginning with A, B, C, or D (A, AA, D1, and so forth) in the succeeding statements in the program will be of the integer type.

If characters are defined in reverse alphabetical order

"Syntax error" will be displayed.

Example:

```
DEF INT D-A
```

Priority of the type declaration character to the statement

The type declaration character added to a variable name is given priority over the type declaration statement even if the type declaration statement is executed.

When there is no DEF statement and type declaration character

A floating-point type is assumed to have been specified.

Program example

10 DEF INT I-N

20 I=3.5

The DEF INT statement defines an integer type for I.

30 J#=3.5

The type declarator defines a floating-point type for J#.

40 JVAR=3.5

As the DEF statement is executed, JVAR is an integer type.

50 A=3.5

A is of the floating-point type because nothing is declared.

60 PRINT I,J#,JVAR,A

RUN

4

3.5

4

3.5

DEF KEY (define key)

A character string is defined for a function key on the keyboard.

Format

DEF KEY function key number, character string

$1 \leq \text{function key number} \leq 10$: Integer

character string: String constant, variable, array variable, or expressions of up to 16 characters

Application

If a character string is defined for a function key, pressing the key will give the same result as that obtained by pressing two or more defined character keys. If frequently-used character string are defined, key-in operation can be facilitated.

Specification method

Five function keys can be used as ten different keys if they are pressed together with the **SHIFT** key. Specify function key numbers as follows:

Key number	Key	Key number	Key
1	F1	6	SHIFT + F1
2	F2	7	SHIFT + F2
3	F3	8	SHIFT + F3
4	F4	9	SHIFT + F4
5	F5	10	SHIFT + F5

A character string having up to sixteen characters can be defined. In this case, control characters such as CHR\$(13) are counted as one character.

Execution example

To define the character string "RUN" for the **F4** key, execute the following :

```
DEF KEY 4, "RUN"
```

When executing a program, press the **F4** then **RETURN** .

To define **RETURN** at the same time, execute the following :

```
DEF KEY 4, "RUN" + CHR$(13)
```

Then press only the **F4** key to start the program.

Initial set upon entry of Sony disk BASIC

Key	Definition	Key	Definition
F1	List	SHIFT + F1	Wipe' ↵
F2	Edit	SHIFT + F2	Console
F3	Auto	SHIFT + F3	Gclear
F4	Debug	SHIFT + F4	Renum
F5	Run	SHIFT + F5	Keylist' ↵

- ↵ mark indicates **RETURN** .
- ' indicates REM, and the characters following to are assumed to be comments.

When **SHIFT** + **F5** are pressed, "Keylist" are displayed. Even if characters are displayed after they are assumed to be comments and KEYLIST statement is executed without deleting these characters.

Holding the defined contents

The defined contents are held until they are redefined in the DEF KEY statement or the computer is reset.

Check of the defined contents

Execute

```
KEY LIST
```

All the contents defined for all function keys are displayed on the screen.

DELETE (delete)

The specified line in the current program are deleted.

Format

DELETE [line number] [-] [line number]

line number : Integer from 0 to 65529 or a label specifying a line

Application

Specification method and execution examples

If the program is as follows :

```
10 ...  
20 ...  
30 ...  
40 ...  
50 ...  
60 ...
```

DELETE 40 deletes line 40.

DELETE 20-40 deletes lines 20, 30 and 40.

DELETE -50 deletes lines from the first line to line 50.

DELETE 30- deletes lines from line 30 to the last line.

After execution of the DELETE statement

the prompt mode will be generated.

If a non-existent line is specified

If

```
DELETE 35
```

is executed in the above example, "Undefined line number 35" will be displayed. If

```
DELETE 15-55
```

is executed, lines 20,30,40 and 50 will be deleted.

If DELETE . is executed

The line where the program execution is interrupted because of an error, STOP or other statement, or the line where edited or listed last is deleted.

Note

Make a list to check whether the specified lines have been deleted by the DELETE statement.

DIM (dimension)

An array variable name, data type, and array dimension and size are declared, and the necessary memory area is allocated. Numeric variables are set to 0s ; string variables are set to null strings.

D

Format

DIM variable name (maximum subscript value [,maximum subscript value] [,maximum subscript value]...)

variable name : Numeric or string type
maximum subscript value : Numeric-constant, array variable, or expressions including these of 0 or more

Application

Specification method

Specify "variable name" with a character string beginning with an alphabetical character and a type declarator (omittable).

Specify the size of the area to be allocated in the "maximum subscript value".

If the maximum subscript value is not an integer, the value will be rounded to the nearest whole number.

Execution example

Execute

```
DIM A$(5)
```

Six areas from A\$(0) to A\$(5) are allocated, and the variables are set to null strings.

A\$(0)
A\$(1)
A\$(2)
A\$(3)
A\$(4)
A\$(5)

Multi-dimensional array variable

If two or more maximum subscript values are specified in the DIM statement, multi-dimensional array elements will be declared. For example, if

```
DIM B(2,2)
```

is specified, 2-dimensional floating-point type array variables will be declared. Nine areas will be allocated, and each value will be set to 0.

B (0,0)	B (0,1)	B (0,2)
B (1,0)	B (1,1)	B (1,2)
B (2,0)	B (2,1)	B (2,2)

Up to 10-dimensional array elements can be declared.

Definition of multiple array variables

Two or more array variables can be specified in one DIM statement. These variables are separated with commas as follows:

```
DIM A(10),B(10),C$(10)
```

If declaration by the DIM statement is omitted

10 is assumed to have been specified as the maximum subscript value, and array variables of up to three dimensions can be used.

Allocated memory size

The memory size allocated by a variable in the DIM statement is as follows:

Integer type:	2 bytes
Floating-point type:	8 bytes
String type:	3 bytes for indicating the top address of data area

Error messages

If a subscript greater than the maximum subscript value specified in the DIM statement is used, "Subscript out of range" will be displayed. If an attempt is made to declare the array variable previously defined in the DIM statement, "Duplicate definition" will be displayed.

DIR (directory)

Lists file names on the disk.

Format

DIR ["[drive name:] file name [. type]"]

drive name : a character from A to P

file name : character string of up to 8 characters

type : character string of up to 3 characters

Application

Specification method

In "drive name", specify the name of the disk drive (A - P) in which the disk containing the files to be listed is inserted.

If the disk in question is the currently logged disk, the "drive name" specification can be omitted.

Execution example

Execute DIR command without specifying file name and type to list out all file names in the currently logged disk.

DIR

```
"A:GRAPHS .BAS "
```

```
"A:MENUE .BAS "
```

```
"A:MEMMAP .BAS "
```

```
"A:FARM .BAS "
```

```
"A:TARGET .BAS "
```

```
"A:MUSIC .BAS "
```

To list out all file names on the disk other than the currently logged disk, specify the drive name as follows :

DIR "B:"

```
"B:BIOS .COM RS "
```

```
"B:TERMS .DAT "
```

```
"B:DATA .NO1 "
```

```
"B:PROGA .BAS "
```

```
"B:PROGA .ASC "
```

```
"B:REPJAN .DAT "
```

```
"B:REPFEB .DAT "
```

```
"B:REPMARCH.DAT "
```

```
"B:REPAPRIL.DAT "
```

```
"B:ANNUAL .DAT "
```

```
"B:DATA .NO2 "
```

DIR "B:REPJAN.DAT"

" "B:REPJAN . DAT

When specifying a file name and type, a question mark "?" is used to represent any character; an asterisk "*" is used to represent any character string equivalent to a file name or a type.

DIR *BAS

```
"A: DEMO      .BAS R
"A: GRAPHS    .BAS
"A: MENU      .BAS
"
```

```
DIR "A:DATA.*"
```

```

"      "A:DATA      .NO2
"      "A:DATA      .NO1

```

DIR "B:REP????.*"Lists out all files with names consisting of 8 characters or less and having RFP as the first 3 characters. This command is effective with any file type on the drive B.

DIR "B:REP???.*"

```
"B:REPPJAN.DAT
"
"B:REPFEB.DAT
"
"B:REPMARCH.DAT
"
"B:REPAAPRIL.DAT
```

When the specification of the type is omitted
Three spaces are assumed to have been specified.

Note

If a file is write-protected by adding a "\$R/O" attribute with the STAT command of the CP/M, "R" will be displayed.
If a "\$SYS" attribute is attached to a file with the STAT command of the CP/M (so that the file name will not be displayed by the DIR command of the CP/M), "S" will be displayed.
For example :

DIR

```
"A: BASIC      .COM R  "
"A: DEBUGGER.PAC R  "
"A: IEEE488 .PAC R  "
"A: DEMO       .BAS R  "
"A: BIOS       .COM S  "
"A: SETUP     .COM  "
"A: BACKUP    .COM  "
"A: GRAPHS    .BAS  "
"A: MENU      .BAS  "
```

The "BASIC.COM", "DEBUGGER.PAC", "IEEE488.PAC", and "DEMO.BAS" are read only files.
The "BIOS.COM" is a file with "\$SYS" attribute, whose file name will not be displayed by the DIR command of the CP/M.

DISK RESET (disk reset)

Resets disk drive to read/write mode.

Format

DISK RESET ["drive name :"]

drive name : a character from A to P

Application

When a disk is replaced with another one while operating under disk BASIC, this disk is automatically set to "read only" mode; any write or erasure operation on the newly inserted disk is inhibited.

To make a write/erasure operation possible to that disk, execute a DISK RESET command.

If an attempt is made to write to that disk without executing DISK RESET, "Disk R/O" will be displayed.

Specification method

In "drive name", specify the name of the disk drive (A - P). The disk in the specified drive and the drive A are set to read/write mode.

When the "drive name" specification is omitted, the currently logged disk and the disk in drive A are reset.

Execution example

```
DISK RESET
```

will reset the currently logged disk and the disk in drive A to the read/write mode.

```
DISK RESET "B:"
```

will reset the disks in the drive A and B to the read/write mode.

Note

If a disk is not inserted into the specified disk drive, the computer waits until a disk is inserted. Insert a disk, or press the RESET button and re-activate Sony disk BASIC.

If a disk drive which is not handled by the CP/M is selected,

```
Error(162) : Disk select error
```

will be displayed.

DRAW (draw)

A graphic is drawn on the graphic display screen as specified in the graphic subcommands.

Format

DRAW "subcommands"

subcommands: String constant, variable or expressions including these

Application

The DRAW statement draws a graphic according to the instructions given. For example, if instructed to move three units to the right, five units up, three units to the left and five units down, it will draw a square with those dimensions.

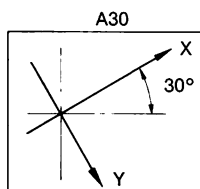
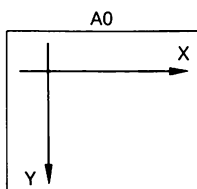
Graphic subcommands

Graphic subcommands are a set of the following.

An

(angle)

- A coordinate system is set by turning the initial coordinate axes n degrees counterclockwise.
 n is an integer ($-32768 \leq n \leq 32767$).



- The initial set is A0.
- If An is omitted, the preceding coordinate system will be used.
- If n is omitted,

`Error<129> : Macro language error`

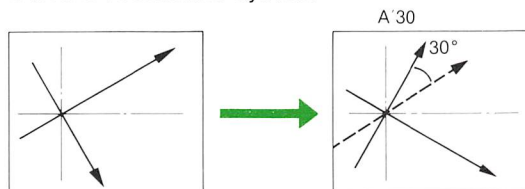
will be displayed.

- The result will be the same if a small a is used instead of A .

A'n

- A coordinate system is set by turning the current coordinate axes n degrees counterclockwise.
- n is an integer ($-32768 \leq n \leq 32767$)

Current coordinate system



- If n is omitted,

`Error(129) : Macro language error`

will be displayed.

- The result will be the same if a small a is used instead of A .

Sn

(scale)

- n dots are specified as a unit for drawing a line. n is an integer ($1 \leq n \leq 640$).
- The initial set is S4 (one unit = 4 dots).
- If S_n is omitted, the preceding set is assumed to have been specified.
- If n is omitted,

`Error(129) : Macro language error`

will be displayed.

- The values of h and v specified with $W, U, D, R, L, w, u, d, r,$ and l in subcommands are equivalent to h times n and v times n dots respectively.
- The result will be the same if a small s is specified instead of S .



S'n

- n is added to the value of the current scale and the result will be the new setting.
- n is an integer and the new setting of the scale should be positive.
- If n is specified so that the new setting of the scale is negative,

Error(129) : Macro language error

will be displayed.

- If n is omitted,
Error(129) : Macro language error
will be displayed.
- The result will be the same if a small s is specified.

S*

- The scale will have double the value of the current scale.
- The result will be the same if a small s is specified.

S/

- The scale will have half the value of the current scale.
- The result will be the same if a small s is specified.

Cn

(color)

- n specifies the color of the line to be drawn.
- See the attached Reference Chart for color codes.
 n is an integer.
- The initial set is C7 (white).
- If Cn is omitted, the preceding color (specified in the DRAW or GCOLOR statement) is assumed to have been specified. If there is no previously specified color, white is assumed to have been specified.
- If n is omitted,

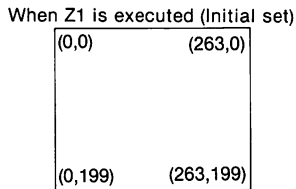
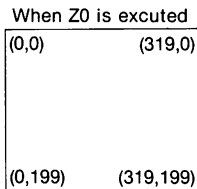
Error(129) : Macro language error

will be displayed.

- The result will be the same if a small c is specified instead of C.

Mx, y
(move)

- A line is drawn from the current position to point (x, y).
Point (x, y) on the screen is as follows :



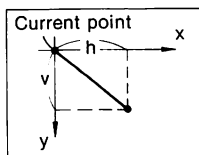
These points are not related to An and Sn. x and y are integers.

- If a small m is specified instead of M, only movement will be performed without drawing a line.
- If x or y is omitted,

Error<129> : Macro language error
will be displayed.

Wh, v
(walk)

- A line is drawn from the current position to point (h, v) in the coordinate system (whose origin is the current point) set with an A command. h and v are integers and their units are specified with an S command with a number of dots.



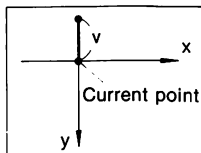
- If a small w is specified instead of W, only movement will be performed without drawing lines.
- If h or v is omitted,

Error<129> : Macro language error
will be displayed.

Uv

(up)

- A line is drawn from the current position to point $(0, -v)$ in the coordinate system (whose origin is the current position) set with an A command.
 v is an integer and its unit is specified with an S command with a number of dots.

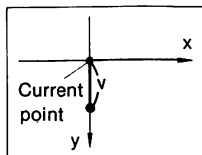


- If a small u is specified instead of U , only movement will be performed without drawing a line.
- If v is omitted, 1 is assumed to have been specified.

Dv

(down)

- A line is drawn from the current position to point $(0, v)$ in the coordinate system (whose origin is the current position) set with an A command.
 v is an integer and its unit is specified with an S command with a number of dots.

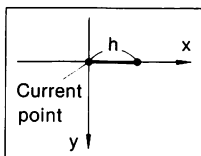


- If a small d is specified instead of D , only movement will be performed without drawing a line.
- If v is omitted, 1 is assumed to have been specified.

Rh

(right)

- A line is drawn from the current position to point $(h, 0)$ in the coordinate system (whose origin is the current point) set with an A command.
 h is an integer and its unit is specified with an S command with a number of dots.

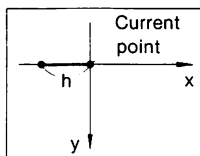


- If a small r is specified instead of R , only movement will be performed without drawing a line.
- If h is omitted, 1 is assumed to have been specified.

Lh

(left)

- A line is drawn from the current position to point $(-h, 0)$ in the coordinate system (whose origin is the current point) set with an A command.
 h is an integer and its unit is specified with an S command with a number of dots.



- If a small l is specified instead of L , only movement will be performed without drawing a line.
- If h is omitted, 1 is assumed to have been specified.

BEh, v

(box empty)

- A rectangle whose diagonal connects the current position and the point (h, v) in the coordinate system (whose origin is the current point) set with an A command is drawn.

h and *v* are integers and their units are specified with an S command with a number of dots.

Each side of the rectangle drawn is parallel to X or Y axis of the initial coordinate system.

- If *h* or *v* is omitted,

Error(129) : Macro language error

will be displayed.

- If a small be is specified instead of BE, the result will be the same.

BFh, v

(box full)

- A rectangle whose diagonal connects the current position and the point (h, v) in the coordinate system (whose origin is the current point) set with an A command is drawn and inside the rectangle is painted.

h and *v* are integers and their units are specified with an S command with a number of dots.

Each side of the rectangle drawn is parallel to the X or Y axis of the initial coordinate system.

- If *h* or *v* is omitted,

Error(129) : Macro language error

will be displayed.

- If a small bf is specified instead of BF, the result will be the same.

Or

(Circle)

- A circle whose radius is *r* in Y-direction is drawn.
- If *r* is omitted, 1 is assumed to have been specified.
- If a small o is specified instead of O, the result will be the same.

Fi

(flag i)

- A flag i is set at the current point. (There is no change on the screen.)
- i is an integer ($1 \leq i \leq 7$)
- By specifying the value i , up to seven flags can be set at the different positions at the same time.
- If i is omitted, 1 is assumed to have been specified.
- The flags are set at the point (0, 0) initially.
- If a small f is specified instead of F , the result will be the same.

Gi

(go to flag i)

- A line is drawn from the current point to the flag i position.
- i is an integer ($1 \leq i \leq 7$)
- If i is omitted, 1 is assumed to have been specified.
- If a small g is specified instead of G , only movement will be performed without drawing a line.
- If $G0$ is specified, a dot is only plotted at the current position. If $g0$ is specified, nothing happens.

Xi, j

(exchange flags)

- The positions of the flag i and j are reversed.
 i and j are integers ($1 \leq i \leq 7$)
- If i or j is omitted, 1 is assumed to have been specified.
- If a small x is specified instead of X , the result will be the same.
- If 0 is specified for i or j , the flag 0 means the current position. For example, if $x\ 0, 1$ is specified, the flag 1 is set at the current position and the current position will move to the position where the flag 1 used to be. A line is drawn between the current position and the flag position when a capital X is specified.

= i, j

- Set the flag i at the flag j position.
 i and j are integers ($1 \leq i, j \leq 7$)
- If i or j is omitted, 1 is assumed to have been specified.
- When 0 is specified as i , the current position will move to the flag j position drawing a line.
- When 0 is specified as j , the flag j is set at the current position.



Pn1, n2
(paint)

- The area containing the current point enclosed in the border line whose color code is $n2$ is colored with color $n1$.
- See the attached Reference Chart for color codes. $n1$ and $n2$ are integers.
- If a negative number is specified for $n2$, the area specified will be painted except for any points of a color or any area enclosed with a color different from the current position.
- If $n1$ or $n2$ is omitted,

Error(129) : Macro language error

will be displayed.

- If the border line is open, the entire screen will be colored, excluding the border line.
- If the current point is on the border line, or the display color of the current point is the color $n1$ or $n2$, no processing will be performed.
- If a small p is specified instead of P, the result will be the same.

Zn

- Compensates automatically for the difference of height and width of one dot.
 Z0: without compensation
 Z1: with compensation
- The initial set is Z1.
- If Zn is omitted, the preceding set is assumed to have been specified.
- If n is omitted,

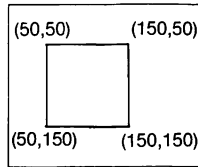
Error(129) : Macro language error

will be displayed.

These subcommands are sequentially executed from the first one in the quotation marks.

Execution example

To display the following rectangle in red with a DRAW statement :



the required commands are as follows :

- ① Move to (50, 50).
- ② Move 100 dots downward, 100 dots to the right, 100 dots upward, and then 100 dots to the left, while drawing a red line.

The statement for these commands can be written as follows :

DRAW "S100m50,50C4DRUL"

Diagram illustrating the command breakdown:

- S100**: Move to (50, 50). (1 unit = 100 dots)
- m50,50**: Move to (50, 50).
- C4**: Red
- DRUL**: Leftward, Upward, Rightward, Downward.

Use of variable

- A string variable can be used as the character data in the DRAW statement. In this case, the variable must be enclosed in brackets. For example, if the following is specified :

```
START$="m50,50"
DRAW "S100[START$]C4DRUL"
```

the above rectangle can be drawn.

If the coordinates in variable START\$ are altered, a rectangle of the same size can be drawn at the desired position on the screen. \$ in the variable enclosed in brackets may be omitted.

- Variables can be specified for x, y, h, v, and n in subcommands. In this case, variables must be enclosed in parentheses to discriminate the commands from the variables.

For example, if the following DRAW statement is written :

```
DRAW "A(n)S(s)C(c)m(x),(>)uLD2R2U2L"
```

the rectangle can be rotated and displaced, and its size and color can be changed by altering variables n, s, c, x, and y.

Repetition of the graphic subcommands

When a set of graphic subcommands are to be repeated in a DRAW statement, specify the number of times, write subcommands and enclose them with "<" and ">".

For example, to repeat drawing a rectangle and turning the axes 20 degrees 18 times, the DRAW statement will be as follows :

```
DRAW "S50C14m100,100<18URDLA'20>"
```

To write music subcommands in a DRAW statement

Write subcommand ^M and write music subcommands. To switch to the graphic subcommands again, write ^G. In this way, you can switch from drawing a graphic to playing music and vice versa easily. The following program is to draw a rectangle while playing sound.

```
10 WIPE
20 FOR I=1 TO 10
30 X=I*10:Y=I*5
40 DRAW "m(X),(Y)C(I)"
50 P$="L64T150
   <4N1N2N3N4N5N6N7N8N9N10N11N12>"
60 DRAW "S50D^M[P$]^GR^M[P$]
   ^GU^M[P$]^GL^M[P$]"
70 NEXT I
```

Program example

Program to randomly draw a star :

```
10 DRAW"S7"
20 ST$="A72LA0LA54DA72LA36RA54D
   A18UA36RA0LA18UA0d"
30 X=IRND(234)+10
40 Y=IRND(159)+10
50 C=IRND(14)+1
60 DRAW"m(X),(Y)C(C)[ST$]P(C),(C)"
70 GOTO 30
```

Press **ESC** key to stop the execution of this program.

To draw a graphic pattern

```
10 WIPE
20 FOR K=1 TO 30
30  DRAW "m120,100C1405L5"
40  DRAW "a'30s'1U2F1R2F2D2F3L2F4"
50  DRAW "w1,-1f5"
60  FOR I=1 TO 4
70    DRAW "G(I)g5"
80  NEXT I
90 NEXT K
```

D

EDIT

(edit)

The program is modified on the display screen in the edit mode.

Format

EDIT [line number]

$0 \leq \text{line number} \leq 65529$: Integer or a label specifying a line

Application

To enter edit mode

To modify the data in line 10, execute

```
EDIT 10
```

The character display screen will be cleared and the specified line will be displayed on the first line of the screen (edit mode). The cursor blinks at the beginning of the statement as follows :

```
10 █INPYT A
```

Use of the edit keys

In the edit mode, the edit keys can be used to correct the line in the same way as with the full screen editor.

The edit keys can also be used with **SHIFT** key or **CTRL** key.

Following table shows the function of edit keys, used independently or together with **SHIFT** or **CTRL** key.

Key(s) used	Function
INS DEL CLR HOME ← → ↑ ↓	A space is inserted at the cursor position. The character at the cursor position is deleted. The data between the cursor position and the end of the program line are deleted. The cursor moves back to the beginning of the program line. The cursor is moved one space in the direction of the arrow.
SHIFT + INS SHIFT + CLR SHIFT + HOME SHIFT + ← SHIFT + → SHIFT + ↑ SHIFT + ↓	The cursor becomes smaller in its height. Any number of characters can be inserted between the character at the cursor position and the preceding one. The data preceding the cursor position are deleted. The cursor moves to the end of the program line. The cursor moves to the beginning of the line on the display screen. The cursor moves to the end of the line on the display. The preceding line is displayed, the displays will scroll down, and the cursor blinks at the first character of the statement. The succeeding line will be displayed and the cursor blinks at the first character of the statement.
CTRL + INS CTRL + DEL CTRL + CLR CTRL + HOME	A blank line will be inserted between the line on which the cursor blinks and the preceding line. The line on which the cursor blinks disappears from the screen. Characters on the display screen are cleared and the edit mode is terminated. The cursor moves to the upper left corner of the screen.

To terminate the edit mode

Press the **RETURN** key; the prompt mode will be re-entered.

To invalidate the modification before pressing the **RETURN** key or **SHIFT** and **↑** / **↓** keys, press the **ESC** key; the statement before modification remains, and the prompt mode will be re-entered.

If EDIT. or EDIT is executed

The edit mode for the line in which execution was last interrupted due to an error or the line edited or listed last is selected.

Error message

If a nonexistant line number is specified, "Undefined line number" will be displayed.

END

(end)

Program execution is terminated and the prompt mode is reentered.

Format

END

Application

When an END statement is executed,

END

will be displayed and program execution will be terminated.

Program execution restart after execution of END statement

A RUN or GOTO statement is used. A CONT statement cannot restart program execution.

Omission of END statement

Execution of a program is automatically terminated if there is no line subject to execution (there is no line having a greater line number), and the END statement indicating the end of the program may be omitted.

The END statement is used to prevent restarting a subroutine after execution of the main program, when the subroutine is written after the main program.

ERA

(erase)

Erases file(s) on a disk.

Format

ERA "[drive name:] file name [.type]"

drive name : A character from A to P

file name : Character string of up to 8 characters

type : Character string of up to 3 characters

Application

Specification method

In "drive name", specify the name of the disk drive (A – P) in which the disk containing file(s) to be erased is inserted.

If the disk in question is the currently logged disk, the "drive name" specification can be omitted.

Execution example

```
ERA " DATA.NO1 "
```

will erase the file "DATA.NO1" on the currently logged disk.

```
ERA " B:ANNUAL.DAT "
```

will erase the file "ANNUAL.DAT" on the disk in drive B.

When the specification of the type is omitted

Three spaces are assumed to have been specified.

To erase files having common character(s) and/or a character string

When specifying a file name and type, a question mark "?" is used to represent any character; an asterisk "*" is used to represent any character string equivalent to a file name or a type.

ERA " * .BAS" Erases all files with the type "BAS" on the currently logged disk.

ERA "A : DATA. * " Erases all files with the file name "DATA" on the drive A disk.

ERA "B : REP?????. * " Erases all files with names consisting of 8 characters or less and having REP as the first 3 characters. This command is effective with any file type on the drive B.

ERA " * . * " Erases all files on the currently logged disk.
When this command is executed, a message "Erase all (Y/N) ?" will be issued before executing erasure of all files to confirm if the command is correct. Press key to erase all files. Press key to cancel the command; the prompt mode will be re-entered.

Note

- If a file is placed in a read only status by adding "\$R/O" attribute with the STAT command of the CP/M, this file cannot be erased by the ERA command. If an erasure of such file is attempted,

Error(164) : File R/O

will be displayed.

ERROR (error)

Simulates the error occurrence of an error of the number specified.

Format

ERROR error number

1 ≤ error number ≤ 255 : Numeric-constant, variable, array variable, or expressions including these

Application

When the ON ERROR GOTO statement has not been executed, the program execution is stopped by executing the ERROR command and the error message corresponding to the specified number is displayed. When the ON ERROR GOTO statement has been executed and the specified error number is between 1 and 127, the program execution is transferred to the line number specified in the ON ERROR GOTO statement. When the specified line number is 128 or over, the program execution is interrupted and the error message is displayed.

Execution example

When

```
ERROR 138
```

is executed,

```
Error(138) : RETURN without GOSUB
```

will be displayed.

If the specification is not an integer

It is rounded to a whole number, then the command is executed.

Values of the ERR and ERL

When the ERROR statement is executed, the specified error number is assigned to the ERR and the line number of the ERROR statement is assigned to the ERL. When the ERROR statement is executed in the direct command mode, 65535 will be assigned to the ERL.

User definition of the error messages

Using the ERROR statement, error messages can be defined. For example, the following program is to display an error message when a negative number is entered and the computer prompts you to enter a positive number.

```
10 ON ERROR GOTO 100
20 FOR I=1 TO 10
30   INPUT A
40   IF A<0 THEN ERROR 50
50   SUM=SUM+A
60 NEXT I
70 PRINT SUM
80 END
100 IF ERR=50 THEN PRINT
    "Negative number. Enter a
    positive number."
110 IF ERL=40 THEN RESUME 30
120 PRINT "Error!"
```

The END command in line 80 is to separate the main program and the error correction program. If line 80 is not inserted, the program execution continues after the execution of line 70, and

Error !

will be printed by executing line 120.

ERROR ON ERROR OFF

(error on)

(error off)

To validate or invalidate the execution of the ON ERROR GOTO statement.

Format

ERROR ON or ERROR OFF

Application

An ON ERROR GOTO statement prevents the program execution from being interrupted by an error occurrence. An ERROR OFF statement invalidates the ON ERROR GOTO statement and the program execution will be interrupted by an error occurrence.

To validate the ON ERROR GOTO statement again in a program, execute

ERROR ON

statement.

Program example

```
10 ON ERROR GOTO 60
20 INPUT "10/",A
30 PRINT "10/";A;"=";10/A
40 GOTO 20
50 END
60 BEEP:BEEP:BEEP
70 CCOLOR 3,,2
80 PRINT "DIVISION BY ZERO!!!!"
90 CCOLOR 7,0,0
100 PRINT "This is your LAST chance"
110 INPUT " to enter DATA ! 10/",A
120 ERROR OFF
130 IF A<>0 GOTO 150
140 BEEP:CCOLOR 4,,2:PRINT "AAAAhhh!"
150 PRINT "10/";A;"=";10/A
160 PRINT "OK."
170 ERROR ON
180 RESUME NEXT
```

In this program, 10 is divided by a value entered from the keyboard and the result is given. If you enter 0 in line 20, an error occurs in line 30 and the control is transferred to the line 60 which is the start line of the error correction routine. In the error correction routine, you are given a chance to enter data again. If you enter 0 again, because of the ERROR OFF command execution,



```
Error(2) : Division by zero
```

will be displayed. If you enter a datum other than 0, ERROR ON is executed and the control returns to the main routine.

EVAL (evaluate)

Evaluates expressions.

Format

EVAL expression

expression : Numeric or string-constant, variable, array variable, or expressions including these

Application

This is used to evaluate an expression when the result is not to be used. An expression is, for example, the right side of a LET variable = X statement.

Execution example

When

```
EVAL INKEY$
```

is executed just before a key data input statement (such as INPUT or INKEY\$), the previously input data which has not been read are read and the key data input statement gets the newly pressed key data.

When

```
EVAL INKEY$(1)
```

is executed, the computer waits until a key is pressed. When a key is pressed, the control is transferred to the next statement. The pressed key data is not assigned to any variable. Execution of A\$ = INKEY\$(1) functions in the same way. It is only different that the pressed key data is assigned to the variable A\$.

Execution of EVAL IRND(0) initializes the random integer generation sequence.

FONT RESET

(font reset)

Resets the font definition.

Format

FONT RESET

Application

A DEF FONT statement defines a font for a character code. Execution of a FONT RESET statement returns the font to the original value.

F

FOR ... NEXT (for ... next)

The program between the FOR and NEXT statements is repeatedly executed.

Format

FOR variable = initial value TO end value [STEP increment]

...

NEXT [variable]

variable : Numeric type (Write the same variables in both the FOR and NEXT statements.)

initial value, end value, and increment :

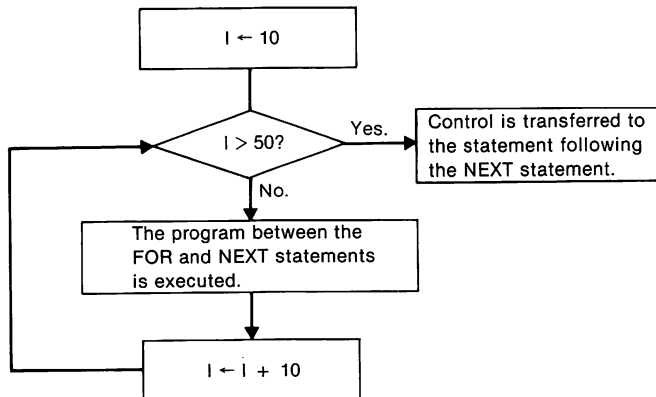
Numeric-constant, variable or expressions including these

Application

Execution example

```
FOR I=10 TO 50 STEP 10
.
.
.
NEXT I
```

This program will be executed as follows :



This repeatedly executed procedure is called a loop.
The FOR and NEXT statements must be used in pairs.

Omission of the specification

If "STEP increment" is omitted, 1 is assumed to have been specified as the increment.

"variable" in the NEXT statement may be omitted. If it is written, the correspondence between the FOR and NEXT statements in the program list can be understood at a glance.

Error messages

- If a FOR statement does not have a corresponding NEXT statement, "FOR without NEXT" will be displayed.
- If a NEXT statement does not have a corresponding FOR statement, "NEXT without FOR" will be displayed.
- When the increment is positive and the end value is less than the initial value, or when the increment is negative and the end value is greater than the initial value, the program in the loop is not executed even once, and the statement following the NEXT statement is executed.
- If control is transferred from the outside to the inside of the loop with a statement such as GOTO, the program will be executed from the line to which control was transferred; however, the NEXT statement does not have the corresponding FOR statement. "NEXT without FOR" will be displayed. Control can be transferred from the inside to the outside of the loop.
- If a FOR statement is executed in the direct command mode, "FOR without NEXT" will be displayed. When FOR and NEXT statements are used in the direct command mode, write them in one line and separate them with colons (:).

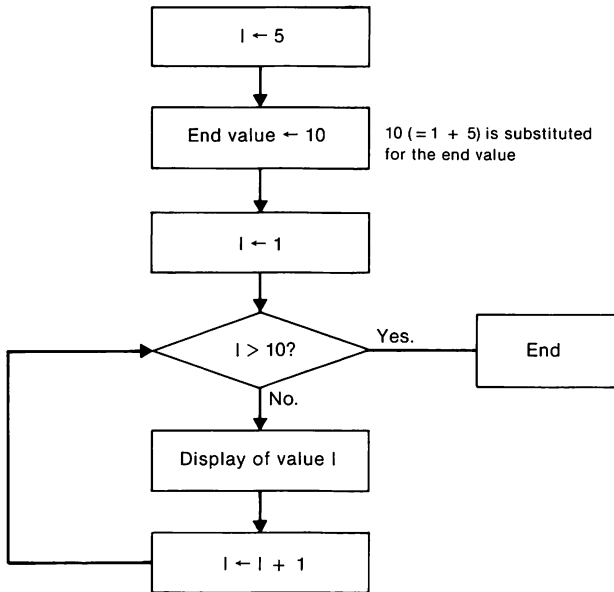
Setting of the end value

The end value is set before the initial value is assigned to a variable.

Example :

```
10 I=5
20 FOR I=1 TO I+5
30 PRINT I
40 NEXT I
```

This program will be executed as follows :



Multi-loop

The FOR-to-NEXT loop can contain another FOR-to-NEXT loop. A loop containing another loop is called a multi-loop (nesting). The internal loop must be completely within the external loop. The following program is valid :

```
10 FOR I=1 TO 5
20   FOR J=1 TO I
30     PRINT "*";
40   NEXT J
50   PRINT
60 NEXT I
RUN
```

} Internal loop
} External loop

*
**

However, the following program is invalid because the internal and external loops cross.

```
10 FOR I=1 TO 5
20   FOR J=1 TO I
30   PRINT "*";
40 NEXT I
50 PRINT
60 NEXT J
RUN
```

Error(136) : FOR without NEXT

For a multi-loop, different variables must be used for each loop. When two or more FOR-to-NEXT loops end at the same point, these loops can be closed simultaneously with one NEXT statement. In this case, specify the variables, separated from the most internal loop's variable with commas (,).

Example :

```
FOR I=1 TO 30
...
...
FOR J=1 TO 5
...
...
FOR K=1 TO 10
...
...
...
NEXT K,J,I
```



GCLEAR (gclear)

The graphic display is cleared and the screen is colored as specified.

Format

GCLEAR [color code] [,mode]

$-32768 \leq \text{color code} \leq 32767$
 $0 \leq \text{mode} \leq 4$

} Numeric-constant, variable or expressions
including these

Application

Specification method

Specify color code and mode according to the attached Reference Chart.

Execution example

If the following is executed when the displayed background color is turquoise (code 3), execute

```
GCLEAR 13,4
```

the current display code 3 and the specified color code 13 are logically operated (XOR of code 4).

Decimal	Binary
3	0011
13	1101
	1110

As a result, the graphic display is cleared and the screen is colored in pale pink.

If the specification is omitted

The color code and the mode will be the background color and mode previously specified in the last executed GCOLOR statement.

If there is no GCOLOR statement, black is assumed to have been specified and no logical operation is performed.

Note

When GMODE 0 has been executed, a GCLEAR statement clears the graphic display only on the write page selected by GPLANE. The display on other pages are not cleared.

GCOLOR (graphic color)

A graphic display color is specified.

Format

GCOLOR [display color code], [background color code],
[border area color code], [mode]

$-32767 \leq \text{color code} \leq 32768$ } Numeric-constant, variable, array variable, or expressions including these
 $0 \leq \text{mode} \leq 4$ }

Application

Specification method

Specify color code and mode according to the attached Reference Chart.

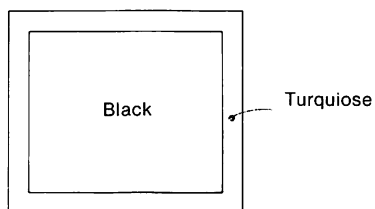
Execution example

Specify

```
GCOLOR 4,7,3,3
```

to specify red for the display color, white for the background, turquoise for the border area, and OR for logical operation.

When the GCOLOR statement is executed, only the border area color changes.



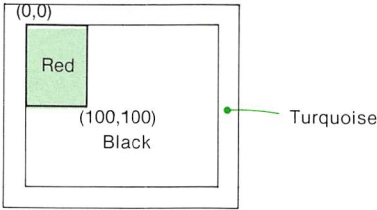
Execute

```
BOXF(0,0)-(100,100)
```

Current display color code 0 and the specified color code 4 are logically operated (OR).

		Code	Binary representation
Display color :	Red	4	0100
Current display color :	Black	0	0000
Logical operation :	OR		
Result			0100 ↓ RED

As a result, the following is displayed :



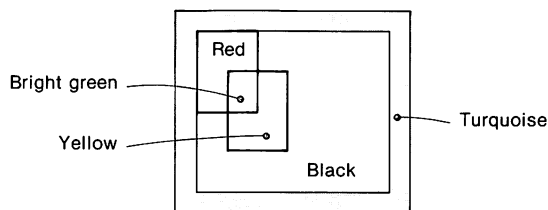
Execute

```
BOXF(50,50)-(150,150),6,4
```

The logical operation is performed as follows :

		Code	Binary representation
Display color :	Yellow	6	0110
Current display color :	Red, Black	4, 0	0100, 0000
Logical operation :	XOR		
			0010, 0110 ↓ ↓ Bright green Yellow

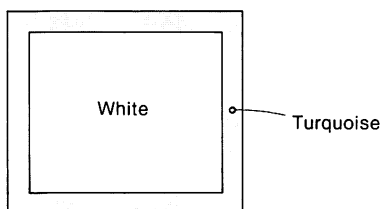
The following is displayed :



Execute

GCLEAR

The specified background color in the GCOLOR statement appears for the first time.



If the specification is omitted

“color code” and “mode” specified in the preceding GCOLOR statement are assumed to have been specified. If there is no GCOLOR statement, the initial value is assumed to have been specified.

Initial set upon entry of Sony disk BASIC

display color code : 7 (white)
background color code : 0 (black)
border area color code : 0 (black)
mode : 0 (No logical operation)

Maintaining the specification in GCOLOR statement

The specification is maintained until the next GCOLOR or WIPE statement is executed.

GET (get)

Reads data of one record from a random file and assigns them to the variables according to the RECORD statement specification.

Format

GET # file number [,record number]

$0 \leq \text{file number} \leq 127$

$0 \leq \text{record number} \leq 65535$

Application

Specification method

Specify the same file number as in the OPEN statement.

When the file number is omitted in the OPEN statement, specify # 0.

In record number, specify which record is to be read.

If the specification of the record number is omitted, record number following that of the record accessed by the last PUT or GET statement will be selected. When the GET statement is to be executed right after an OPEN statement, 0 is specified as the record number.

Data read operation

Before reading data with a GET statement, an OPEN and a RECORD statement must be executed.

Execution example

Execution of

```
GET #2,3
```

reads the data of 4th record of the file device and assigns them to the variables specified by RECORD.

When, for example,

```
RECORD #1,RJUST 4 AS P$,B%
```

has been executed, the following record is read by GET,

&H34	&H46	&H41	&H42	&H43	&H20	&H20
4	F	A	B	&H2043		

"4FAB" is assigned to the variable P\$ and a value &H2043 is assigned to the variable B%.

Program example

```
10 OPEN #1,"TEST.DAT"
20 RECORD #1,A%,LJUST 5 AS S$
30 PRINT "RECORD #1,A%,LJUST 5 AS S$"
40 PRINT "PUT #1,0"
50 A%=&H4634
60 S$="ABC"
70 PRINT "A%=&H";HEX$(A%,4)
80 PRINT "S$="CHR$(34);S$;CHR$(34)
90 PUT #1,0
100 CLOSE
110 PRINT "-----"
120 OPEN #2,"TEST.DAT"
130 RECORD #2,RJUST 4 AS P$,B%
140 PRINT "RECORD #2,RJUST 4 AS P$,B%"
150 PRINT "GET #2,0"
160 GET #2,0
170 PRINT "P$="CHR$(34);P$;CHR$(34)
180 PRINT "B%=&H";HEX$(B%,4)
190 CLOSE
200 END
```

```
RUN
RECORD #1,A%,LJUST 5 AS S$
PUT #1,0
A%=&H4634
S$="ABC"

-----
RECORD #2,RJUST 4 AS P$,B%
GET #2,0
P$="4FAB"
B%=&H2043
```

G

GGET (graphic get)

The color codes of the dots in the specified graphic screen area are stored into the array variable.

Format

GGET [(start point coordinates)]-(end point coordinates), array variable

$-16384 \leq x\text{-coordinates} \leq 16383$	}	Numeric-constant, variable, array variable, or expressions including these (standard resolution)
$-16384 \leq y\text{-coordinates} \leq 16383$		

array variable : Integer type array variable (The array must be declared in the DIM statement in advance.)

Application

The information about the color(s) of all the dots inside the rectangle whose diagonal connect two points specified in the GGET statement is stored into the array variable.

This information can be displayed again with a GPUT statement ; a certain graphic pattern can be repeatedly displayed.

Specification method

Specify an integer array variable and the subscript indicates the first variable into which the color code is to be stored : usually specify 0. The array must be declared in the DIM statement in advance.

The information about one dot is represented with four bits in low or standard resolution mode.

In other words, information about four dots can be stored into one integer variable (two bytes). The size of the array to be declared is as follows :

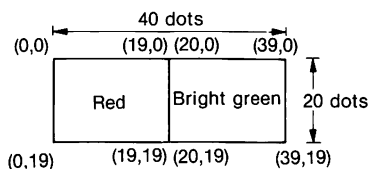
Number of dots in the specified area $\div 4$

In the high resolution mode, the information about one dot is represented with two bits. Therefore 8 dots information is stored in an integer variable. In super-high resolution mode, one bit is for one dot and 16 dots information is stored in an integer variable.

If the specified value is not an integer, the value is rounded to the nearest whole number.

Execution example

When the following square is drawn on the screen in the standard resolution mode :



the array size required to read the color information about all dots in this square is :

$$(40 \times 20) \div 4 = 200$$

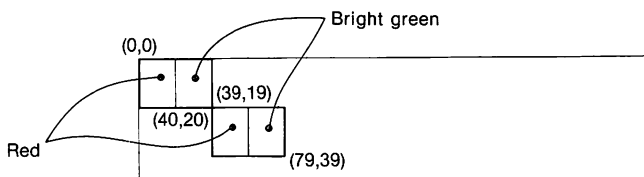
Execute the following to store the information :

```
DIM A%(199)
GGET(0,0)-(39,19),A%(0)
```

If

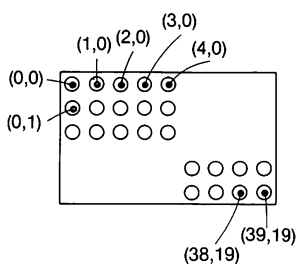
```
GPLOT(40,20)-(79,39),A%(0)
```

is executed, the following will be displayed :

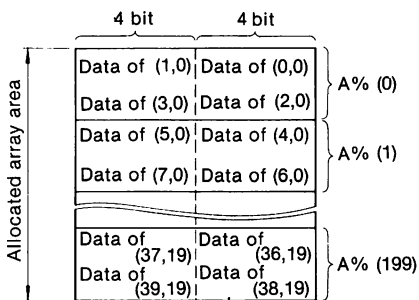


Reading the information

If the above-mentioned GGET statement is executed, the color codes of each dot on the screen will be converted to binary numbers, and then stored into the array area specified with the DIM statement as follows :



Dots on the screen



Memory

If the specification is omitted

The start point coordinates are the last coordinates (x,y) specified in the preceding graphic command or the current position of a DRAW statement. When there is no graphic or DRAW command, (0,0) are assumed to have been specified.

If a point outside the display screen is specified

Error(8) : Out of screen

will be displayed.

Specification of array variable subscript

In the array area allocated with the DIM statement, the color information is sequentially stored into the variable of the specified subscript. When 10 is specified as the subscript, the information is stored as follows :

(1,0)	(0,0)	}	A% (10)
(3,0)	(2,0)		
(5,0)	(4,0)	}	A% (11)
(7,0)	(6,0)		
...		}	A% (199)
(37,18)	(36,18)		
(39,18)	(38,18)		

If DIM A% (199) has been declared, only 760-dot data (190 × 4) can be stored in A% (10) to A% (199).

GLOCATE

(graphic coordinate system locate)

Moves the coordinate system of the graphic display screen to the desired position.

Format

GLOCATE (coordinates), X direction , Y direction

$-16384 \leq X \text{ coordinate} \leq 16383$ } Numeric constant, variable, array variable,
 $-16384 \leq Y \text{ coordinate} \leq 16383$ } or expressions including these
 X direction, Y direction : Integer type constant, variable, array variable, or expressions including these

Application

Specification method

The parameters in the () specify the origin of the new coordinate system with respect to the initial coordinate system.

“X direction” and “Y direction” specify the direction of the coordinates.

When a positive value (including 0) is specified

negative positive
 X axis  X
 The right side of the screen is for a positive coordinate.

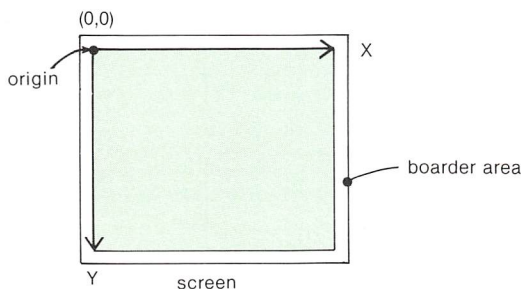
negative
 Y axis  Y
 positive
 The lower side of the screen is for a positive coordinate.

When a negative value is specified

positive negative
 X  X
 The left side of the screen is for a positive coordinate.

positive Y
 Y  Y
 negative
 The upper side of the screen is for a positive coordinate.

Initial coordinate system upon entry of Sony disk BASIC



Execution example

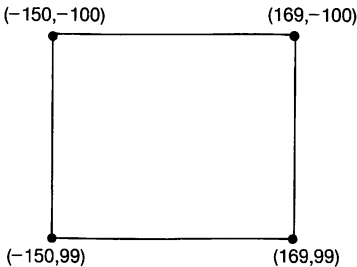
Let us now explain the execution of the following program.

```
10 CIRCLE (100,100),80,,4
20 GLOCATE (150,100)
30 CIRCLE (100,100),80,,4
```

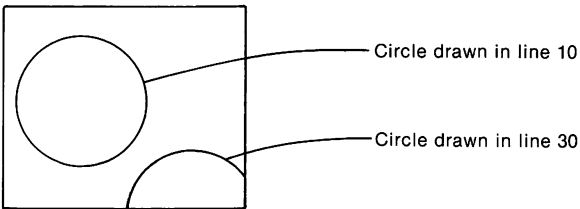
The initial coordinates on the screen are as follows ;



and a circle is drawn based on the coordinates.
Then the coordinate system is moved as shown below.



and another circle is drawn based on these new coordinates.



When the specification is omitted

When the parameter defining the X or Y direction is omitted, the current value is retained.



GMODE (graphic mode)

Sets the resolution of the graphic display screen.

Format

GMODE resolution mode [,color set code]

$0 \leq \text{resolution mode} \leq 3$
color set code = 0 or 1

}

Numeric-constant, variable, array variable,
or expression including these

Application

Four graphic display screen modes are classified according to their resolutions. This command selects one of the modes. Note that GMODE 3 is invalid when the SMC-70G is used in Gen Lock ON mode. For details about the graphic display screen, see "Chapter 2-2 Display screen" in Part 1.

Specification method

Specification of the resolution mode sets the resolution as follows.

Mode	Resolution
0	Low resolution (160 × 100 dots × 4 pages)
1	Standard resolution (320 × 200 dots)
2	High resolution (640 × 200 dots)
3	Super high resolution (640 × 400 dots)

The color set code must be specified only when the high resolution mode is selected. It sets the display colors.

Code	Display colors
0	black, red, green, white
1	black, red, green, blue

When it is specified in another resolution,

Error(3) : Illegal function call

will be displayed.

If the specification of the color set code is omitted,
0 is assumed to have been specified.

Initial set upon entry of Sony disk BASIC
The standard resolution mode is set.

Maintaining the specification in GMODE statement

The specification is maintained until the next GMODE statement is executed.

Note

For the coordinates on the graphic display screen in each resolution, see Chapter 2-2 in Part 1.

To select one of the pages in the low resolution mode, use the GPLANE command.

SONY®**SUPPLEMENT**

The enclosed page(s) supersede those existing in this manual.
Please replace the appropriate page(s) with the one(s) contained
in this packet.

Sony Disk BASIC

The files on the Sony Disk BASIC disk are as follows :

File name	Description
BASIC. COM	The BASIC interpreter
DEBUGGER. PAC	The link module for debugging
IEEE. PAC	The link module for the IEEE-488 interface
PIC. PAC	The link module for saving/loading graphic data
COPY. PAC	The link module for making a hard copy of the graphics and characters being displayed on the screen
DEMO. BAS	A BASIC demonstration program

HOW TO USE THE "PIC. PAC" FILE

The "PIC. PAC" file is a package of the machine language subroutines used to save the graphics being displayed on the graphic display on a disk as a graphic data file. It is also used to load the graphic data from the file on a disk and display it on the screen. To use these machine language subroutines, first load the PIC. PAC file into the memory with a LINK command.¹⁾

To save or load graphic data, execute a CALL statement to transfer control to the subroutine in the linked file. You can select the following four subroutines in the CALL statement : GSAVE, GSAVEALL, GLOAD, GOVLY.

Though the PIC. PAC file can be used in any graphic resolution mode, the coordinates of a point must be specified with those of GMODE 1 (standard resolution mode).

¹⁾ When the PIC. PAC file is linked,

GRAPHICS EXTENDED COMMAND FOR PICTURE FILES VER 1.2

will be displayed on the screen, and graphic data files used in optional software, such as the Graphics Editor, Video Titler, etc., can be saved and loaded with BASIC.

- 1) **GSAVE**: Saves the graphic data being displayed in the specified area on a disk as a data file.

Format

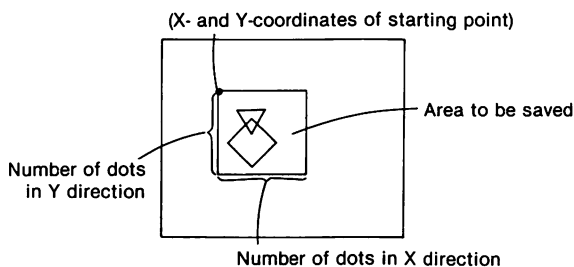
{ — or CALL } GSAVE ("[drive name:] file name" [,X-coordinate of starting point, Y-coordinate of starting point, number of dots in X direction, number of dots in Y direction])

Application

When the graphic data are to be saved on the disk in the current drive, the drive name can be omitted. It is not necessary to specify the type for the file name; the graphic data are saved automatically with type ".PIC".

The area of the graphic display to be saved is specified by the X- and Y-coordinates of the upper lefthand corner and by the number of dots in the X and Y directions. If these specifications are omitted, the entire display area is saved.

Note that the X-coordinate must be an even number. If an odd number is specified as the X-coordinate, the even number before the specified odd number is assumed to have been specified.



Execution examples

To save a particular area

```
_GSAVE ("FLOWER",100,30,150,150)
```

The graphic data in the square area whose diagonal connects point (100,30) with point (250,180) are saved on the disk in the current drive with the file name "FLOWER. PIC".

To save the entire display

```
_GSAVE ("B:STAR")
```

The graphic data in the entire graphic display area are saved on the disk in drive B with the file name "STAR. PIC".

- 2) **GSAVEALL**: Saves the graphic data in the entire graphic display area as well as the data specifying the border area color code on a disk.

Format

$\left\{ \begin{array}{c} \text{—} \\ \text{or} \\ \text{CALL} \end{array} \right\}$ GSAVEALL ("[drive name:] file name", border area color code)

Application

When the graphic data are to be saved on the disk in the current drive, the drive name can be omitted. It is not necessary to specify the type for the file name; the graphic data are saved automatically with type ".PIC".

Specify the border area color code with one of the color codes from 0 to 15.

Execution example

```
_GSAVEALL ("DOG",1)
```

The entire graphic display area is saved on the disk in the current drive with the file name "DOG. PIC". The border area color is registered as blue (color code 1). (The border area color being displayed has no effect on the execution of the above statement.)

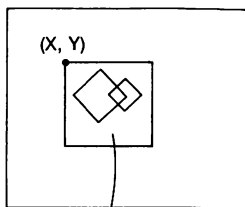
- 3) **GLOAD**: Loads and displays the graphic data from the disk on the graphic display. Any graphic data displayed in the specified graphic display area will be cleared.

Format

$\left\{ \begin{array}{l} \text{—} \\ \text{or} \\ \text{CALL} \end{array} \right\} \text{GLOAD ("[drive name :] file name" [,X-coordinate, Y-coordinate])}$

Application

Specify the X- and Y-coordinates of the upper lefthand corner of the square area. Note that the X-coordinate must be an even number, just as when saving. The number of dots in the X and Y directions is as specified when saving. If the coordinates are not specified, the upper lefthand corner of the screen (0,0) is assumed to have been specified.



Area to be displayed
(the number of dots in the X
and Y directions is the same
value as specified when saving.)

Execution examples

To load the graphic data in a particular area

```
_GLOAD ("FLOWER",160,100)
```

The graphic data saved with the name "FLOWER.PIC" on the disk in the current drive are loaded on the graphic display area with the point (160,100) as the upper lefthand corner of the area.

To load the graphic data on the entire display

```
_GLOAD ("B:STAR")
```

The graphic data saved with the name "STAR.PIC" on the disk in drive B are loaded, starting from the upper lefthand corner of the screen.

Note

If no coordinates are specified in GLOAD, the border area color will be the color specified in GSAVEALL.

- 4) **GOVLY**: Loads and displays the graphic data saved on the disk over the picture being displayed on the screen.

Format

$\left\{ \begin{array}{c} \text{—} \\ \text{or} \\ \text{CALL} \end{array} \right\}$ GOVLY ("[drive name:] file name" [,X-coordinate, Y-coordinate]
 [,transparent color code])

Application

Specify the X- and Y-coordinates of the upper lefthand corner of the graphic display area over which the graphic data are to be overlaid, just as in GLOAD. If the coordinates are not specified, the upper lefthand corner of the screen (0,0) is assumed to have been specified.

Specify the code of the color which is to be transparent. When the graphic data are loaded, the specified color becomes transparent, and the picture displayed on the screen is seen through the transparent areas. If the transparent color code is not specified, the color of the upper lefthand corner of the graphic data saved on the disk is assumed to have been specified as transparent.

Execution examples

To overlay the graphic data in a particular area

```
_GOVLY ("FLOWER",100,100,6)
```

The graphic data saved on the disk in the current drive with the file name "FLOWER. PIC" are loaded over the graphic display area, starting from the point (100,100). The areas occupied by the color of code 6 (yellow) will become transparent and the picture displayed on the screen can be seen through these areas.

To overlay the graphic data on the entire display

```
_GOVLY ("B:STAR")
```

The graphic data saved on the disk in drive B with the file name "STAR. PIC" are loaded and displayed over the entire screen, starting from the upper lefthand corner of the screen. The color of the upper lefthand corner of the graphic data on the disk is assumed to be transparent.

HOW TO USE THE "COPY. PAC" FILE

The "COPY. PAC" file is a machine language program used to make a hard copy of the graphics and characters being displayed on the screen with the SMI-7020 dot matrix printer. To use this program, first load the file into the memory with a LINK command. Once the file has been linked to the BASIC interpreter, you can make a hard copy simply by executing a CALL statement in which the subroutine name "COPY" is specified.

COPY: Makes a hard copy of the graphics and characters being displayed on the screen.

Format

$\left\{ \begin{array}{c} \text{—} \\ \text{or} \\ \text{CALL} \end{array} \right\} \text{COPY [("density code")}]$

Application

The density code defines the density of the dots on the hard copy for each color being displayed. Specify the density with string data of up to 16 values in hexadecimal notation (using from 0 to F). The first value in the string data indicates the density of the dots of the first color (code 0, black) in the standard-resolution mode color code; the second value indicates the density of the second color (code 1, deep blue), and so on.

The larger the value of the density code is, the more densely the dots are printed out on the hard copy. If density code 0 is specified, no dots are printed (white is shown on the hard copy).

If no density code is specified, the values will be assigned as in this initial setting:

```
_COPY ("02468ACE13579BDF")
```

If less than 16 values are assigned in the density code, the values of the above initial setting will be applied to colors to which no value has been assigned.

Note that the standard-resolution mode density code format must be used to specify density code values for all graphic resolution modes. These values will be automatically applied to the same colors in any resolution mode.

Execution example

```
_COPY ("02468ACE"+STRING$(8,"0"))
```

A hard copy will be printed out of only the part of the display using the density code values for the colors specified here (color code 1 to 7).

Note

When printing out the characters displayed in the 80-characters-per-line mode, the density of dots on the hard copy might not be as specified.

GOSUB ... RETURN

(Go to subroutine ... return)

A GOSUB statement transfers control unconditionally to the subroutine program beginning with the specified line number. A RETURN statement indicating the end of the subroutine program returns control to the statement succeeding the GOSUB statement.

Format

GOSUB line number

:

RETURN

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

If the same processing is repeated in a program, specify this processing in a subroutine program.

To transfer control to a subroutine in a program, specify the subroutine heading line number in a GOSUB statement. Write RETURN in the last line of the subroutine.

GOSUB and RETURN statements must be used in combinations.

If there is no RETURN statement related to a GOSUB statement, control does not return from the subroutine.

If there is no GOSUB statement related to a RETURN statement, "RETURN without GOSUB" will be displayed.

GOSUB statement multiplicity

A subroutine program can be called from more than one GOSUB statement (See Figure 1). Moreover, more than one subroutine program can be specified in a program (see Figure 2) and a subroutine can be called from another subroutine (see Figure 3).

In any case, GOSUB and RETURN statements must be used in combination.

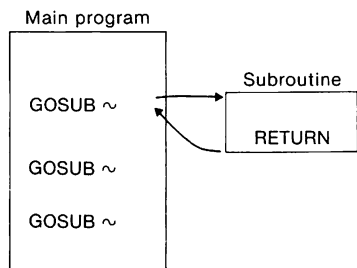


Fig. 1

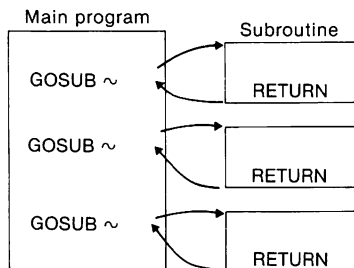


Fig. 2

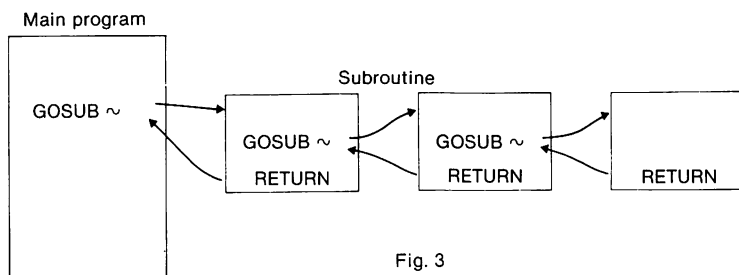


Fig. 3

Program example

Program to change the colors of eight squares at random :

```
10 DEF INT A-Z
20 RANDOMIZE
30 WIPE:CURSOR OFF
40 FOR X=0 TO 280 STEP 40
50   BOXF (X,0)-(X+39,100),X/40+1
60 NEXT X
70 /
80 FOR I=1 TO 3000:NEXT I
90 POS=RND*7
100 X1=POS*40:X2=X1+39
110 C=GPOINT(X1,0)
120 FOR I=1 TO 2
130   FOR D=0 TO 19:GOSUB 190:NEXT D
140   C=(C+1) MOD 7+1
150   FOR D=20 TO 39: GOSUB 190:NEXT D
160 NEXT I
170 GOTO 80
180 /
190 LINE (X1+D,0)-(X1+D,100),C,4
200 LINE (X2-D,0)-(X2-D,100);C,4
210 RETURN
```

1)

2) 

3)

4)

1) Draws eight squares with different colors.

2) Selects one of the eight squares at random.

3) Gradually deletes the color of the selected square and then displays a new color. Performs this operation twice.

4) Gradually changes color of squares.

GOTO (go to)

Unconditionally transfers control to the specified line.

Format

GOTO line number

line number : Integer constant in the range of 0 to 65529 or a label specifying a line

Application

Specification method

A statement can be specified before or after a GOTO statement, but a line number existing in the program must be specified with the number or a label. Variables cannot be used.

When a nonexistent line is specified

“Undefined line number” will be displayed.

When a GOTO statement is executed in the direct command mode

Execution starts from the line specified in the program.

Unlike a RUN statement, the variables are not initialized.

Use of GO TO

GO TO can be used instead of GOTO.

GPLANE (graphic plane)

Selects the page to be displayed and the page on which the graphic data is to be written in the low resolution mode.

Format

GPLANE [display page number] [,write page number]

-32768 ≤ number ≤ 32767 : Numeric-constant, variable, array variable, or expressions including these

Application

In the low resolution mode, the display screen has four pages, each page displayed separately. This command selects one page for display and one page on which data are to be written. By switching the four pages using this command, an interesting display such as animation can be produced.

For details about the graphic display screen, see "Chapter 2-2 Display screen" in Part 1.

Specification method

The relation between number and page is as follows :

Number	Page
0	Page 0
1	Page 1
2	Page 2
3	Page 3

If the specification is not an integer, it will be rounded to the nearest whole number.

If a negative value or a value of 4 or over is specified as a page number, MOD 4 of the specified value will be obtained and the page is selected according to the result.

Execution example

In the low resolution mode, when

```
GPLANE 2,1  
BOXF(0,0)-(50,50),3
```

are executed, page 2 is displayed and a box in turquoise is drawn on page 1. When

```
GPLANE 1
```



is executed, the box which has been drawn on page 1 can be seen for the first time.

Initial set upon entry of Sony disk BASIC

Page 0 is set for the display and write page.

Maintaining the specification in GPLANE statement

The specification is maintained until the next GPLANE statement is executed.

If the specification of the page is omitted,

The display page or the write page specified in the preceding GPLANE statement is assumed to have been specified. If there is no GPLANE statement, the initial value is assumed to have been specified.

If the GPLANE statement is executed in another resolution mode,

The specification is maintained and when the low resolution mode is selected, it will be effective.

Program example

```
10 GMODE 0
20 WIPE
30 A$="Z1S1A(N)C1m80,50u20R20D40L40U40R20"
40 FOR I=0 TO 3
50   GPLANE 0,I
60   N=I*22:DRAW A$
70 NEXT
80 FOR I=1 TO 100
90   GPLANE (I MOD 4)
100  J=(I MOD 4)+9
110  PLAY "L1603N(J)"
120 NEXT
```

G P L O T (graphic plot)

Displays a dot at the specified point on a graphic display screen.

Format

G P L O T (coordinates), [display color code], [mode]

-16384 ≤ X-coordinate ≤ 16383	} Numeric-constant, variable, array variable, or expressions including these
-16384 ≤ Y-coordinate ≤ 16383	
-32768 ≤ display color code ≤ 32767	
0 ≤ mode ≤ 4	

Application

Specification method

See the attached Reference Chart for the coordinates, display color code, and mode.

Execution example

When the entire graphic screen is bright green (code 2), execute

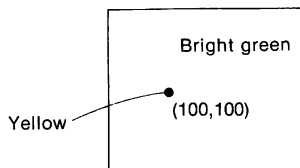
G P L O T (1 0 0 , 1 0 0) , 4 , 3

Current display color code 2 and specified color code 4 are logically operated (OR of code 3).

Decimal	Binary
2	0010
4	0100

0110

As a result, code 6 is set; a yellow dot is displayed at coordinates (100,100).



When the specification is omitted

The display color code and mode are set to those specified in the last executed GCOLOR statement. If there are no GCOLOR statements, the display color will be set to white, and the mode will be set to no logical operation.

GPUP

(graphic put)

Displays dots on the graphic display screen. The color codes of these dots are specified by data stored in the array with a GGET statement.

Format

GPUP [(start point coordinates)]-(end point coordinates), array variable, [mode], [color code not displayed]

$-16384 \leq \text{X-coordinate} \leq 16383$

$-16384 \leq \text{Y-coordinate} \leq 16383$

$0 \leq \text{mode} \leq 4$

$0 \leq \text{color code not displayed} \leq 255$

} Numeric-constant, variable, array variable,
or expressions including these

array variable : Integer type array variable (The array must be declared in the DIM statement in advance.)

Application

A GPUP statement displays color information, which has been stored in the array with a GGET statement, beginning at the upper-left corner of a rectangle whose diagonal is obtained by connecting the two specified points.

Specification method

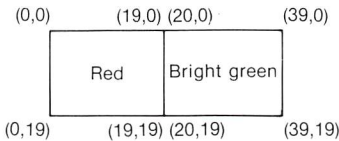
Specify the coordinates on the graphic display screen and the mode according to the attached Reference Chart.

The array variable is of the integer type used to store data with a GGET statement, and the subscript specifies variable data to start the display.

"Color code not displayed" specifies the color which you do not want to display. When it is specified, the data in the array which correspond to the specified code are not displayed and the current display color is maintained.

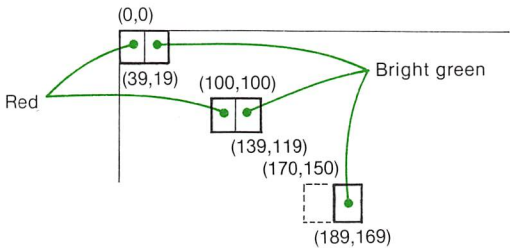
Execution example

Assume that the following rectangles are drawn on the screen :



The screen will be changed as shown below by executing the following :

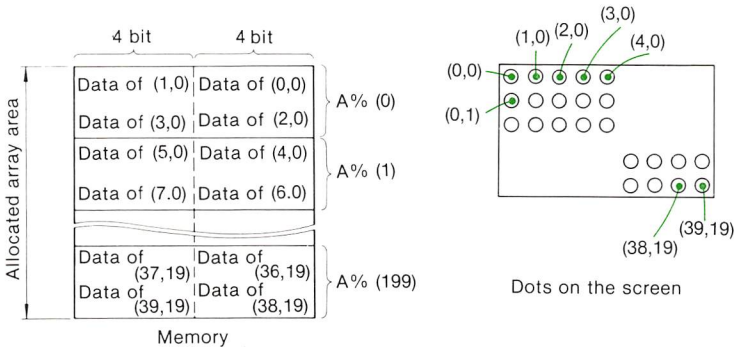
```
DIM A%(199)
GGET(0,0)-(39,19),A%(0)
GPUT(100,100)-(139,119),A%(0),0
GPUT(150,150)-(189,169),A%(0),0,4
```



Because a color code 4 is specified, the data of red are not displayed.

Information output

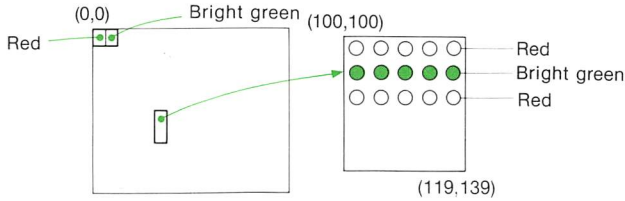
When the GPUT statement is executed in the above example, data in the memory stored by the GGET statement is displayed sequentially from the top left of the area specified, using each four bits as the color code. At this time, the logical operation specified in the mode for each dot is executed, and the result is displayed.



When

```
GPUT(100,100)-(119,139),A%(0),0
```

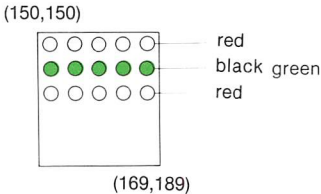
is executed after the GGET statement in the above execution example, the red and bright green dot strings will be displayed alternately beginning at the top of a rectangle whose diagonal is obtained by connecting two points (100, 100) and (119, 139) as follows :



When

```
GPUT(150,150)-(169,189),A%(0),0,2
```

is then executed, because of the specification of the color code of 2, red and the current display color black dot strings will be displayed alternately as follows :



Specification of array variable subscript

The display starting variable data in the array reserved with a DIM statement should be specified.

For example, assume that the following are specified :

```
DIM A%(199)
GGET(0,0)-(39,19),A%(0)
GPUT(100,100)-(139,119),A%(100),0
```

Data beginning with A% (100) is output sequentially.

Since there is only data for 400 dots ($100 \times 4 = 400$) after subscript 100, dots are displayed only in the upper half of the specified rectangle and the lower half remains unchanged.

When specification is omitted

The start coordinates are the last coordinates (x, y) specified in the preceding graphic command. If there are no graphic commands or DRAW statement, the start coordinates will be (0, 0).

When the mode is omitted the mode specified in the preceding GCOLOR statement is set. If there is no GCOLOR statement, no-logical-operation mode will be set.

When the array variable not specified in the GGET statement is used

In the case that an integer type array variable is specified by a GPUT statement, each 4-bit data in the memory is assumed to be a color code and it is displayed on the screen as explained in "Information output" on page 117.

In the case of a floating-point or string-type array variable,

Error(3) : Illegal function call

will be displayed.



IF ... THEN ... ELSE (if ... then ... else)

Obtains the value in the IF statement. If the result is true (not 0), executes the statement succeeding THEN ; if false (0), executes the statement succeeding ELSE.

Format

IF expression [THEN statement] [ELSE statement]

Application

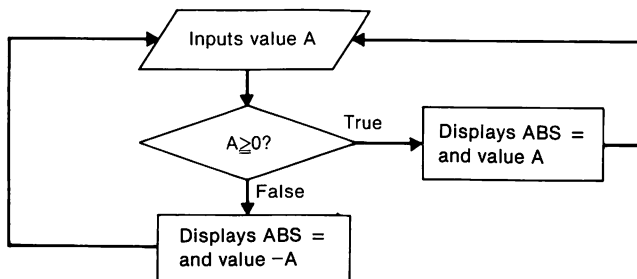
The expression may be a relational, logical, or arithmetic expression if the result is of a numeric type.

When an IF statement is executed, the value of the expression is obtained. If the result is true (not 0), the statement succeeding THEN will be executed ; if the result is false (0), the statement succeeding ELSE will be executed. Then the next statement is executed.

Execution example

```
10 INPUT A
20 IF A>=0 THEN PRINT "ABS=" ;A
   ELSE PRINT "ABS=" ; -A
30 GOTO 10
```

The above program will be executed as follows :



When THEN or ELSE statement is omitted

When THEN statement is omitted and if the expression value is true, the succeeding statement will be executed immediately ; if it is false, the statement succeeding ELSE will be executed.

When ELSE statement is omitted and if the expression value is true, the statement succeeding THEN will be executed. If it is false, the THEN and subsequent statement will be ignored, and the next line will be executed.

When THEN statement is succeeded by GOTO, GOSUB or LET statement

Either THEN or succeeding command (GOTO, GOSUB, LET) may be omitted.

For example,

```
IF A=0 THEN GOTO 30
```

may be written as follows :

```
IF A=0 THEN 30
```

or

```
IF A=0 GOTO 30
```

When ELSE statement is succeeded by GOTO statement

GOTO may be omitted.

Multi statements in THEN or ELSE statement

When THEN or ELSE is succeeded by more than one statement, these statements must be separated with colons (:).

These statements are executed sequentially from left to right.

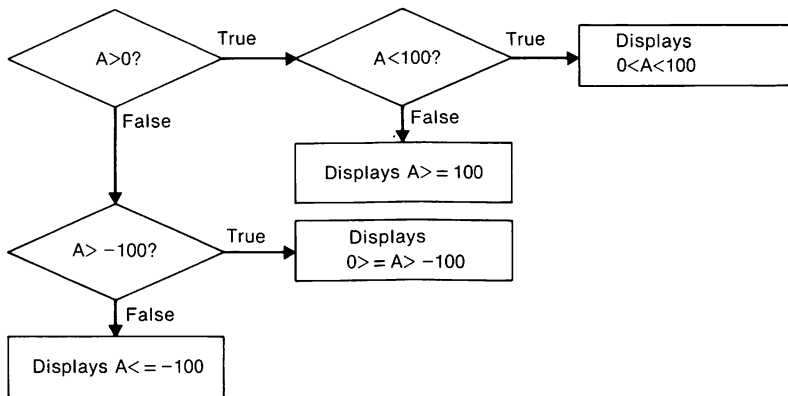
Multiple IF statements

An IF statement can be written after THEN or ELSE (nesting).

Example

```
IF A>0 THEN IF A<100 THEN PRINT "0<A<100"  
                ELSE PRINT "A>=100"  
ELSE IF A>-100 THEN PRINT "0>=A>-100"  
                ELSE PRINT "A<=-100"
```

These statements will be executed as follows :



INPUT (input)

Inputs variable value(s) from the keyboard.

Format

- ① INPUT [,] variable [,variable] [,variable]...
- ② INPUT "prompt statement" { ; or } variable [,variable] [,variable]...
variable : Numeric or string-variable or array variable

Application

Execution example

Execute

```
INPUT A
```

Then,

```
? █
```

is displayed. Key in data to be assigned to the variable (in this case, A), and then press **RETURN**. The keyed in value will be assigned to the variable.

Execute

```
INPUT "DATA=" ; D
```

```
DATA=? █
```

will be displayed and the value to be input can be learned with this display. "DATA=" is called a prompt statement.

When a question mark (?) is to be suppressed

Write a comma (,) preceding the variable(s).

For example, execution of INPUT, A will display

■

and execution of INPUT "DATA = ", D will display

DATA= ■

Multi-data input

More than one variable can be written in an INPUT statement.

Separate these variables with commas (,).

When

INPUT A,B,C

is executed,

? ■

is displayed. Key in three numeric data separated with commas (,).

Example

```
10 INPUT "X,Y,Z=" ,X,Y,Z
20 PRINT X+Y+Z
RUN
X,Y,Z=100,200,300
600
```

Type of variables and data

The input data type must match the related variable type.

If character-type data is entered when the variable is of the numeric type, the following message will be displayed :

```
Non Numeric. Redo from start.
```

The cursor blinks again and the data input wait state is set. In this case, enter the numeric data again.

If numeric data is entered when the variable is of the character type, the numeric data is assumed to have been character data and is assigned to the variable.

Data input for numeric variable

- If a + or - sign is assigned at the beginning, the + will be omitted and will be entered.
- The succeeding and preceding spaces are ignored.
- For a floating-point expression, E is entered directly.
- If there is a character other than numbers and E, for example, letter, +, -, space, etc. between the numbers, the following message will be displayed :

```
Non Numeric. Redo from start.
```

The cursor blinks again, and the data input wait state is set.

Data input for string variable

- Normally it is not necessary to enclose character data in quotation marks.
- When enclosed in quotation marks ("), only the contents are entered as character data.
- When data contains a comma (,), enclose this data in quotation marks as follows :

```
? "A, B"
```

Otherwise, the comma is assumed to be a data delimiter.



- If a quotation mark (") is not written at the beginning of the data (for example, X"ABC"), it will be entered as character data. If a quotation mark is written at the beginning of data, it is assumed to be one of a pair of quotation marks enclosing character data, not data. For example, assume that the following is specified to input character data X enclosed in quotation marks :

? " "X" "

The first quotation mark is assumed to be one of a pair of quotation marks enclosing character data. If a second quotation mark is found, data (null string in this case) enclosed in these two quotation marks is assumed to be the first data. However, since there are no commas delimiting data, the following message is output :

Syntax error. Redo from start.

Data entry is awaited again.

A quotation mark can not be written at the beginning of the character data.

The numbers of data sets to be entered

Data sets entered must be equal to the number of the variables in the INPUT statement. When excessive data is entered, data is assigned from the beginning, and excessive data is ignored. In this case, the message

Extra lost

is output, and processing continues. When data entered for a variable is insufficient, the following message is displayed

Too few. Redo from start.

Data entry is awaited again. In this case, enter again from the first data.

INPUT/L (line input)

Inputs variable value(s) from the keyboard.

Format

- 1 INPUT/L [,] string variable
- 2 INPUT/L "prompt statement" or string variable

Application

The INPUT/L command is used in the same way as the INPUT command.

It is only different that a comma is not assumed to be a separator of data. A return code is assumed to be a separator and a comma is also assigned to the variable.

Execution example

```
INPUT/L A$  
4,321,230
```

```
PRINT A$  
4,321,230
```

4,321,230 is entered. All characters including commas are assigned to the variable A\$.

A prompt statement can be displayed as with INPUT.

```
INPUT/L "NAME,AGE ";B$  
NAME,AGE ?JANE,22
```

```
PRINT B$  
JANE,22
```

When a comma is specified before the variable, ? will not be displayed.

```
INPUT/L "NAME,AGE ",B$  
NAME,AGE JEFFREY,25
```

```
PRINT B$  
JEFFREY,25
```

A quotation mark " is also regarded as a character of a datum. Spaces succeeded to the datum or included in a datum are entered as characters of a datum, but the preceeding spaces are ignored.

On/off of the cursor

Even if the CURSOR OFF statement has been executed in a program, the cursor will be displayed during the execution of an INPUT/L statement. When the execution is completed, the cursor will be erased again.



INPUT # (input number)

Reads data sequentially from a file opened with an OPEN/I statement, then assigns the data to the variables.

Format

INPUT # file number, variable [,variable]...

$0 \leq \text{file number} \leq 127$

variable : Numeric or string type.

Application

Before data is read from a sequential file with this command, the file must be opened with an OPEN/I statement.

```
OPEN/I #1, "NAME.BAS"  
INPUT #1, A, B, C
```

Specify the same file number in INPUT # as in OPEN/I.

When the file number is omitted in OPEN/I statement, specify #0 as the file number.

The read values are sequentially assigned to the variables listed in the INPUT # statement. After the read operation is completed, execute a CLOSE statement.

Specification method

Specification of the variables can be made in the same way as in an INPUT statement. The difference is that the INPUT # statement reads data from a file on a specified device while the INPUT statement reads data from the keyboard.

Data read operation

The INPUT # statement reads data sequentially from the first datum. For numeric data, the spaces before the data are ignored and the data up to the first comma or carriage return code are assigned to a variable. For string data, the first character is assumed to be the start of a unit of string data. All characters up to the first comma or carriage return code are assumed to be a single string data.

If a comma or carriage return code is not encountered for 255 characters,

```
Error(135) : Line buffer overflow
```

will be displayed.

Execution example

The following program writes "A, B, C, D, E, F, G, H, I, J," into the file "DATA.BAS" on the disk. Then it reads 10 units of string data and assigns the data to the 10 array variables and displays them.

```
10 DIM A$(10)
20 OPEN /O #1,"DATA.BAS"
30 FOR I=1 TO 10
40 PRINT #1,CHR$(&H40+I);", ";
50 NEXT I
60 PRINT #1
70 CLOSE #1
80 OPEN /I #1,"DATA.BAS"
90 FOR I=1 TO 10
100 INPUT #1,A$(I)
110 PRINT A$(I)
120 NEXT I
130 CLOSE #1
```

Matching of variable and data type

The type of the variable and data to be received must match.

If the read data type does not match the variable type in the INPUT # statement, for example, if characters are to be assigned to a numeric variable,

```
Error(6) : Non numeric
```

will be displayed.

Error messages

- If the last datum of the file has been read and you want to read another datum,

```
Error(17) : Read past EOF
```

will be displayed.

- If an INPUT # statement is executed before an OPEN/I statement has been executed or different file numbers have been specified in the OPEN/I and INPUT # statements,

```
Error(155) : File not open
```

will be displayed.

- If an error occurs during reading of data

Error(14) : File read error

will be displayed.

Note

The INPUT # statement can be used in the same way as the RECEIVE statement.

However, when a file has been generated by OPEN/S and SEND, the file must be opened by OPEN/R and the data must be read by RECEIVE. When a file has been generated by OPEN/O and PRINT #, the file must be opened by OPEN/I and the data must be read by INPUT # or INPUT/L #.



INPUT/L

(line input number)

Reads string data sequentially from a file opened with an OPEN/I statement, then assigns the data to the string variables.

Format

LINE INPUT # file number, variable [,variable]...

$0 \leq \text{file number} \leq 127$

variable : String type

Application

Before data is read from a sequential file with this command, the file must be opened with an OPEN/I statement.

```
OPEN/I #1, "NAME.BAS"  
INPUT/L #1, A$
```

Specify the same file number in INPUT/L # as in OPEN/I.

When the file number is omitted in OPEN/I, specify #0 as the file number.

The read value is assigned to the variable in the INPUT/L # statement. After the read operation is completed, execute a CLOSE statement.

Specification method

Specifications of the variable can be made in the same way as in an INPUT/L statement. The difference is that the INPUT/L # statement reads data from a file on a specified device while the INPUT/L statement reads data from the keyboard.

Data read operation

The INPUT/L # statement reads data sequentially from the first datum. The first character is assumed to be the start of a unit of string data. All characters up to the first carriage return code are assumed to be a single string data.

If a carriage return is not encountered for 255 characters,

```
Error(135) : Line buffer overflow
```

will be displayed.

Execution example

The following program is to save 10 units of string data-A to J-into a file "DATA.BAS" and read the data to one variable.

```
10 OPEN /O #1,"DATA.BAS"
20 FOR I=1 TO 10
30 PRINT #1,CHR$(&H40+I);", ";
40 NEXT I
50 PRINT #1
60 CLOSE #1
70 OPEN /I #1,"DATA.BAS"
80 INPUT /L #1,A$
90 PRINT A$
100 CLOSE
```

When this program is executed, string data A to J are written separated by commas into a file named DATA.BAS and the file is closed. In line 70, the file is opened again and the characters up to a carriage return code are read as one data and are assigned to the variable A\$. That is,

A,B,C,D,E,F,G,H,I,J,

is displayed as the value A\$.

Type of variable and data

The type of the variable and data to be received must be string type. If the data saved is numeric type, the data is assigned assuming that it is of the string type.

Error messages

- If the last datum of the file has been read and you want to read another data,

Error(17) : Read past EOF

will be displayed.



- If an INPUT/L # statement is executed before an OPEN/I statement or if different file numbers are specified in the OPEN and INPUT/L # statements,

Error(155) : File not open

will be displayed.

- If an error occurs during reading of data,

Error(14) : File read error

will be displayed.



KEY LIST (key list)

Displays the contents defined for the function keys.

Format

KEY LIST

Application

When this command is executed, the function key contents are displayed on the screen together with the key numbers.

F1 to **F5** are related to function key numbers 1 to 5, and **SHIFT** + **F1** to **SHIFT** + **F5** are related to function key number 6 to 10.

The color of control characters is interchanged with the background color.

An **M** with the colors interchanged indicates **RETURN**.

Initial set upon entry of Sony disk BASIC

Key	Definition	Key	Definition
F1	List	SHIFT + F1	Wipe' M
F2	Edit	SHIFT + F2	Console
F3	Auto	SHIFT + F3	Gclear
F4	Debug	SHIFT + F4	Renum
F5	Run	SHIFT + F5	Keylist' M

K

LET

(let)

Assigns the value on the right side to the variable on the left side.

Format

[LET] variable = X

variable : Numeric or string- variable or array variable

X : Numeric or string-constant, variable, array variable, or expressions including these

Application

Specification method

- In the LET statement, the type of the variable on the left side must be the same as that of data on the right side.
- When data on the right side is a string type constant, enclose this data in quotation marks. For example, when

```
LET A$="A+B"
```

is specified, string data A + B is assigned to string variable A\$.
When

```
LET A$=A+B
```

is specified, this indicates the assigning of the sum of numeric variables A and B to string type variable A\$. In this case "Type mismatch" will be displayed.

- LET in the LET statement can be omitted.
When omitted, the result is the same.
Example :

```
LET A=A+3
```

```
A=A+3
```

The above two statements specify the assigning of the sum of variable A and 3 to variable A.

When a new value is assigned to a variable, the preceding value is deleted.

Example :

```
A=3
```

Assigns 3 to variable A.

```
A=A+5
```

Adds 5 to the variable A value. In other words, 8 is assigned as the variable A value. At this time, the preceding variable A value, 3, is deleted.

```
PRINT A
```

```
8
```

Type conversion

When both sides of a LET statement are of different numeric types (integer or floating-point type), the type on the right side is converted to the variable type on the left side.

Example :

```
A=45.6
```

```
A%=45.6
```

```
PRINT A,A%
```

```
45.6      46
```

(The digit are rounded to the nearest whole number.)

L

LINE (line)

Connects the two specified points on the graphic display screen with a line.

Format

LINE [(start point coordinates)]-(end point coordinates), [display color code], [mode]

-16384 ≤ X-coordinate ≤ 16383	} Numeric-constant, variable, array variable, or expressions including these
-16384 ≤ Y-coordinate ≤ 16383	
-32768 ≤ display color code ≤ 32767	
0 ≤ mode ≤ 4	

Application

Specification method

See the Reference Chart for the specification of the display color code and mode.

Execution example

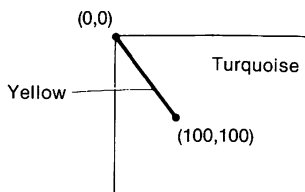
When the entire graphic screen is turquoise (code 3), execute

```
LINE(0,0)-(100,100),5,4
```

The current color code 3 and specified color code 5 are logically operated (XOR of code 4)

Decimal	Binary
3	0011
5	0101
	0110

Therefore, the line will be displayed in yellow (code 6).



When the specification is omitted

The "start point coordinates" are the last coordinates (x,y) specified in the preceding graphic command or the current point of the DRAW statement. If there are not graphic commands or a DRAW statement, the start point coordinates will be (0,0).

Display color code and mode are the same as those specified in the last executed GCOLOR statement. When there are no GCOLOR statements, the display color is white, and the no-logical-operation mode is set.

Program example

```
10 DEF INT A-Z
20 RANDOMIZE
30 WIPE:CURSOR OFF
40 PRINT "Hit ESC to break.";
50 GCOLOR RND*7+.5,,,RND*5-.5
60 FOR I=1 TO 43
70   X1=(X1+1) MOD 320:LINE -(X1,199)
80   Y1=(Y1+1) MOD 200:LINE -(319,Y1)
90   X2=(X2+3) MOD 320:LINE -(X2,0)
100  Y2=(Y2+7) MOD 200:LINE -(0,Y2)
110 NEXT I
120 GOTO 50
```

LINK (link)

Loads the machine language program from a device to memory holding the memory contents.

Format

LINK "[device name :] file name [.type]"

device name : DSK (disk), CTR (cassette tape), PAC (ROM pack) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

Application

A LINK command is used to load a machine language program file having a predetermined format without destroying the memory contents. (See Appendix 6 in Part 1 for the format.)

Specification method

Specify as device name the device from which the program is to be loaded.

When the specification of the device name is omitted, the current disk is selected. To select another disk drive, specify the drive name instead of "DSK :".

In file name and type, specify the same names as when the file was generated.

Execution example

This Sony BASIC disk includes a program for debugging named "DEBUGGER.PAC".

This program is loaded with a LINK command.

When

```
LINK "DSK:DEBUGGER.PAC"
```

is executed, the program is searched and loaded. When loading is completed, the prompt mode is reentered.

Memory hold

The program stored in memory before executing a LINK statement is not affected by the LINK statement execution.

Deleting a file loaded with the LINK statement

A file loaded with a LINK statement is deleted from memory when an MCLEAR statement is executed or when a machine language subroutine and BASIC program are loaded simultaneously with a LOAD or CLOAD statement.

Error messages

- If the specified program is not found

Error(10) : File not found

will be displayed. When the file being searched for is on a cassette tape, the computer continues to search for the specified file until you press the **ESC** key or an error occurs.

- If the specified file does not have the predetermined format

Error(140) : Invalid format

will be displayed.

- If an error occurs during loading

Error(11) : Check-sum error

or

Error(14) : File read error

will be displayed.

Note

The machine language subroutine which is loaded with a LINK command is called from the Sony BASIC program with a CALL statement using the subroutine name specified when it was generated.

LIST

(list out)

Lists the program stored in the memory on the screen.

Format

LIST [line number] [-] [line number]

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

Specification method and execution example

Programs are listed by specifying the line numbers.

Assume that the following program lines exist :

```
10 .....  
20 .....  
30 .....  
40 .....  
50 .....  
60 .....
```

L

LIST lists all lines.

LIST 40 lists line 40.

LIST 20-40 lists lines 20, 30 and 40.

LIST -50 lists lines 10 to 50.

LIST 30- lists lines 30, 40, 50 and 60.

When a nonexistent line is specified

When

LIST 35

is executed in the above example, no line will be displayed. If

LIST 15-55

is specified, lines 20 to 50 will be listed.

When LIST . is executed

The line at which execution has been interrupted because of an error or the line listed or edited last is displayed.

To temporarily stop screen scrolling

Press the **CTRL** and **S** keys simultaneously. A small cursor will blink indicating that the listing has been interrupted. Pressing the space bar lists the program line by line.

To restart the display, keep the space bar pressed or press any other key other than the **ESC** key.



LJUST (left justify)

Used in a RECORD statement and specifies the left justification of a string variable value setting.

Format

LJUST number of bytes AS string variable

number of bytes: Numeric-constant, variable, array variable, or expressions including these

Application

Execution example

When

```
RECORD #1,LJUST 100 AS A$
```

is executed, 100 bytes are allocated for the variable A\$ and the value is set into the allocated area with left justification. Execution of

```
RECORD #1,LJUST 10 AS B$(10)
```

specified 11 array variables B\$(0) to B\$(10) with 10 bytes each. The values of these array variables are set with left justification.

When the length of the variable values are longer than the length specified in RECORD, the extra characters are ignored.

Program example

The following program is to write data in a file "TEST.DAT"

```
10 OPEN #1,"TEST.DAT"  
20 RECORD #1,LJUST 10 AS A$(2),LJUST 20 AS B$  
30 A$(0)="ABC"  
40 A$(1)="0123456789abc"  
50 A$(2)="defgh"  
60 B$="SONY BASIC"  
70 PUT #1  
80 CLOSE
```

After the execution of this program, the file contents is displayed by a TYPE command.

```
TYPE "TEST.DAT"
```

```
ABC          0123456789defgh          SONY BASIC
```

As shown above, each variable value is written into the specified allocation with left justification. The last three characters of the value of the variable A\$(1) are ignored.

LLIST

(line printer list out)

Outputs the list of the programs stored in the memory to a printer.

Format

LLIST [line number] [-] [line number]

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

Specifications and program list format are the same as those for a LIST statement.

The list is not displayed on the screen during execution of an LLIST statement.

LOAD

(load)

A Sony BASIC program file is loaded into memory from the specified device.

Format

LOAD [V] "[device name:] file name [.type]"

device name : DSK (disk), CTR (cassette tape), PAC (ROM pack) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

Application

Specification method

Specify the DSK, CTR or PACK depending on the device from which the program is to be loaded.

When the specification of the device name is omitted, the current disk is selected. To select another disk drive, specify the drive name instead of "DSK:".

Specify as file name and type those specified when the program was saved. When loading from the ROM pack, specify as indicated in the ROM pack manual. When type is omitted, "BAS" is specified.

Execution example

Execute

```
LOAD "TEST.BAS"
```

The specified program (TEST.BAS) is found and the program in memory is cleared. All numeric variables are set to 0's; string variables are set to null strings. Array variables are set in the undefined state (as when a NEW statement is executed), then the program is loaded. When the loading ends, the prompt mode will be re-entered.

To verify the program on the device with memory contents

LOAD/V statement is used to check whether the memory contents have been saved correctly.

Execute

LOAD/V "[device name:] file name [.type]"

"Compared O.K." will be displayed when the verification is completed without an error. If an error occurs, the "Verify error" will be displayed. In this case, save the memory contents again.

The memory contents are maintained after execution of the LOAD/V statement.

Loading of programs saved in different ways

When the program was saved with a SAVE/P,
it is loaded into memory, but the program list cannot be displayed.

When the program was saved with a SAVE/C,
it is loaded into memory, then the commands entered when the program was saved will be sequentially executed.

When the program was saved together with the machine language subroutine,

the low and high memory addresses set when the program was saved are automatically set, and the program is loaded together with the machine language subroutine. Addresses need not be specified in LOAD; the program will be loaded between the start and end addresses specified when the program was saved.

- If the program to be loaded with LOAD includes a machine language subroutine and if another program is to be loaded with LINK at the same time, execute the LOAD statement first, then executed the LINK statement. Otherwise, the program loaded with LINK will be cleared.

If only a Sony BASIC program is loaded with LOAD and a program is loaded with LINK, either statement may be executed first.

If the specified program is not found

```
Error(10) : File not found
```

will be displayed. When cassette tape is being used, the computer continues to search for the specified file until you press the **ESC** key or until an error occurs. The memory contents before the LOAD statement was executed will be held.

If an error occurs during loading

“Check-sum error” or “File read error” will be displayed. If the program to be loaded was saved with SAVE or CSAVE, the memory contents were completely cleared when the error occurred. If the program to be loaded was saved in the ASCII format with SAVE/A or CSAVE/A, the lines loaded before the error occurred will not be cleared.

LOCATE

(locate)

Moves the cursor to location (x, y) on the character display screen.

Format

LOCATE (x-coordinate, y-coordinate)

$0 \leq$	x-coordinate	≤ 79	(for 80-character display)	} Numeric-constant, variable, array variable, or expressions including these
$0 \leq$	x-coordinate	≤ 39	(for 40-character display)	
$0 \leq$	y-coordinate	≤ 24		

Application

Execution example

A LOCATE statement is normally used together with a PRINT or INPUT statement.

```
LOCATE(10,10): PRINT "A"
```

specifies displaying character A at location (10,10).



LOG IN (log-in disk)

Changes the currently logged disk.

Format

LOG IN "drive name :"

drive name : a character from A to P

Application

The currently logged disk is the disk to which the disk operating commands (DIR, ERA, RENAME, TYPE, DISK RESET, USER, LOAD, SAVE and others) will be effective when they are executed without specifying the drive name.

Use the LOG IN command to change the currently logged disk from one to another when two or more disks are used.

Specification method

In "drive name", specify the name of the disk drive (A to P) to be set as the currently logged disk.

Execution example

```
LOG IN "B:"
```

will set the drive B disk as the currently logged disk.

```
LOG IN "A:"
```

will set the drive A disk as the currently logged disk.

Note

- If a drive has been set to write-protected state (read only state), the LOG IN command does not change this state.
- If a disk is not inserted into the specified disk drive, the computer waits until a disk is inserted. Insert a disk, or press the RESET button and re-activate Sony disk BASIC.

If a disk drive which is not handled by the CP/M is selected,

```
Error(162) : Disk select error
```

will be displayed.

MCLEAR (memory clear)

Sets the memory area used by Sony disk BASIC.

Format

MCLEAR [lower limit address] [,upper limit address]

address : Numeric-constant, variable, array variable, or expressions including these

Application

This command limits the memory area used by the Sony disk BASIC program to protect the Sony disk BASIC program area from a machine language subroutine or vice versa.

A machine language subroutine generated outside the limited area will not be destroyed.

Specification method

In the lower limit address, specify the lowest address of the Sony disk BASIC program area to be reserved in the memory and specify the highest address in the upper limit address.

If an address within the BASIC interpreter or the system program is specified,

Error(3) : Illegal function call

will be displayed and the MCLEAR command is not executed.

MCLEAR statement execution and memory contents

When an MCLEAR statement is executed, the Sony disk BASIC program and machine language programs loaded by LINK command in the memory are deleted.

When specification of the address is omitted

The value is reset to the initial setting.

Holding low and high memory addresses

The low and high memory addresses set when MCLEAR statement is executed are held until an MCLEAR statement is executed again, the control is transferred to the CP/M mode and Sony disk BASIC is re-activated, or the computer is reset.

Note

The low and high memory addresses set when Sony disk BASIC is activated differ among the version of BASIC. To obtain approximate addresses, see "Appendix 3 Memory configuration" in Part 1.

MERGE (merge)

Merges a program which has been saved on the file device with the ASCII format into a program in memory.

Format

MERGE "[device name:] file name [.type]"

device name : DSK (disk), CTR (cassette tape), PAC (ROM pack) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

Application

This command is used to, merge a program in memory with subroutines on a file device.

Specification method

Specify DSK, CTR, PACK or CHN depending on the device from which the program is to be loaded.

If the device name is omitted, the disk on the current drive will be specified.

When a disk drive which is not the current drive is to be specified, specify the drive name in the device name. To merge a program on the disk in the drive B, execute.

```
MERGE "B:PROG1.ASC"
```

Specify as file name and type those specified when the program was saved.

Execution example

Execute

```
MERGE "DSK:PROG.ASC"
```

The specified program (PROG.ASC) is loaded. The memory contents, the program and the variable values, are maintained. After the program has been loaded, the prompt mode is re-entered.

Program format to be merged

The program to be merged must have been saved with the ASCII format. If you attempt to merge a program which has not been saved with the ASCII format,

**Error(147) : Can't MERGE non-ASCII
file**

error will occur.

If the line numbers of the program to be merged have the same line numbers as the program in memory

The program lines in memory are deleted and the lines on the device will replace them.

MOTOR (motor)

Starts or stops the cassette tape recorder motor.

Format

MOTOR ON or MOTOR OFF

Application

This command is effective only while the SMC-70 is connected to the cassette tape recorder remote control jack.

MOTOR statement execution and tape movement

When

MOTOR ON

is executed while the recorder PLAY or RECORD button is pressed, the tape starts ; when

MOTOR OFF

is executed, the tape stops.

MOTOR ON/OFF automatic switching

If the remote control jack is connected, the tape can not be started by pressing the recorder PLAY or RECORD button ; the motor is automatically started or stopped when a CSAVE, CLOAD, SAVE, LOAD, or OPEN statement is executed. To manually operate the tape, execute a MOTOR ON statement, and then manually operate recorder buttons.

Note

When a MOTOR ON statement is executed, the read/write operation starts after approximately 1 second which is required for motor rotation stabilization.

M

NEW

(new)

Deletes programs being stored, sets all numeric variable values to 0s and string variable values to null strings, and places array variables in the undefined state.

Format

NEW

Application

Execute this command before entering a new program. When this command is executed, all previous programs are deleted, and the prompt mode is generated.

Note

- Whether the NEW statement is executed can be checked by listing a program with a LIST statement. If a NEW statement is executed, nothing will be listed with a LIST statement.
- Machine language programs loaded with LINK are not deleted by executing a NEW statement.

ON ERROR GOTO

(on error go to)

Transfers the control to the specified line when an error occurs during program execution.

Format

ON ERROR GOTO line number

$0 \leq \text{line number} \leq 65529$: Numeric-constant or a label specifying a line.

Application

This command is used to prevent the program execution being interrupted because of an error occurrence. When the ON ERROR GOTO statement has been executed, and when an error occurs, the execution will be transferred to the specified line which is the first line of the error correction routine.

Execution example

```
10 ON ERROR GOTO 100
20 INPUT A
30 B=SQR(A)
40 PRINT "SQR(A)=";B
50 END
100 IF ERR=3 AND ERL=30
    THEN A=ABS(A):RESUME
```

In this program, if a negative number is entered in line 20, illegal function call error (error code 5) will occur in line 30 and the control is transferred to line 100. In line 100, the absolute value of A is obtained and the execution returns to line 30 with RESUME.

When an error whose code is 128 or over occurs

The ON ERROR GOTO statement is invalid. The program execution is interrupted and an error message is displayed.

If an error occurs in the error correction routine,

The program execution is interrupted in that line and

Error(145) : No RESUME

will be displayed.

Note

- To invalidate the execution of the ON ERROR GOTO statement, execute the ERROR OFF command.
- For the numbering of errors, see Chapter 3.

ON ... GOSUB (on ... go to subroutine)

According to the indicated numeric data value, branches to the subroutine program beginning with the specified line number. When a RETURN statement indicating the end of the subroutine program is executed, control returns to the statement succeeding the ON ... GOSUB statement.

Format

ON X GOSUB line number, line number, line number ...

X : Numeric-variable, array variable, or expressions including these

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

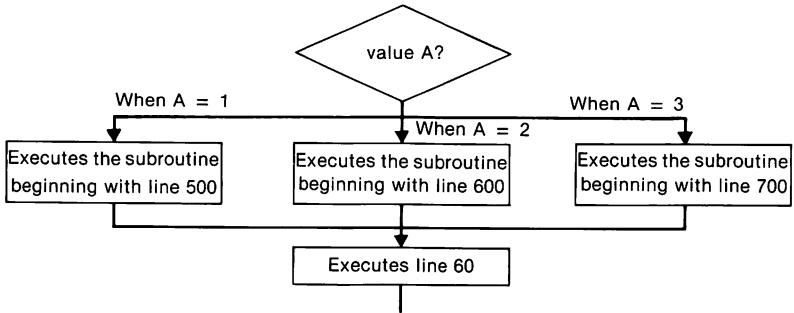
If the numeric data succeeding ON is 1, control will be transferred to the subroutine program beginning with the first line number listed after GOSUB ; if it is 2, control will be transferred to the subroutine program beginning with the second line number listed after GOSUB ; and so on.

After execution of the subroutine program, control returns to the statement succeeding the ON ... GOSUB statement with a RETURN statement.

Execution example

```
      :  
      :  
40    :  
50    ON A GOSUB 500,600,700  
60    :  
      :  
      :
```

The program will be executed as follows :



List of line numbers

After GOSUB, the necessary line numbers are listed, separated with commas (,).

The specified line numbers must exist in the program.

When X is not an integer

The digits are rounded to the nearest whole number.

When the value X is 0 or it is greater than the number of line numbers listed after GOSUB

Control is transferred to the statement succeeding the ON ... GOSUB statement.

When the value X is minus or it is greater than or equal to 256

"Overflow" will be displayed.



Program example

```
10 DEF INT A-Z
20 RANDOMIZE
30 GCLEAR : CCLEAR : CURSOR OFF
40 PRINT "Hit ESC to break.";
50 D=10
60 X=160 : Y=100
70 /
80 C=RND*7+.5
90 BOXF(X,Y)-(X+D-1,Y+D-1),C
100 ON RND*4+.5 GOSUB 130,160,190,220
110 GOTO 80
120 /
130 X=X+D+1 : IF X>319-D THEN X=160
140 RETURN
150 /
160 X=X-D-1 : IF X<0 THEN X=160
170 RETURN
180 /
190 Y=Y+D+1 : IF Y>199-D THEN Y=100
200 RETURN
210 /
220 Y=Y-D-1 : IF Y<0 THEN Y=100
230 RETURN
```


ON ... GOTO (on ... go to)

According to the indicated numeric data value, branches to the specified line.

Format

ON X GOTO line number, line number, line number ...

X : Numeric-variable, array variable, or expressions including these

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

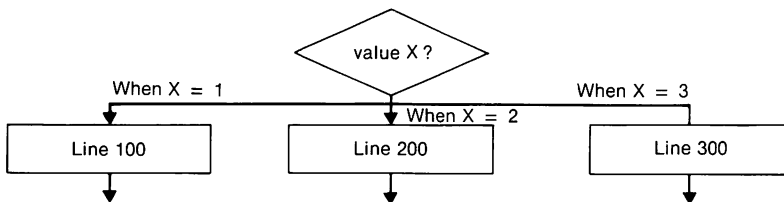
When an ON ... GOTO statement is executed, control is transferred according to the numeric data value succeeding ON:

If the numeric data value is 1, control will be transferred to the first line number listed after GOTO ; if it is 2, control will be transferred to the second line number ; and so on.

GO TO can be written instead of GOTO.

```
40 ...  
50 ON X GOTO 100,200,300  
60 ...
```

The above program will be executed as follows :



List of line numbers

After GOTO, the necessary line numbers are listed, separated with commas (,).

The specified line numbers must exist in the program.

When X is not an integer

The digits are rounded to the nearest whole number.

When the value X is 0 or it is greater than the number of line numbers listed after GOTO

Control is transferred to the statement succeeding the ON ... GOTO statement.

When the value X is minus or it is greater than or equal to 256

"Overflow" will be displayed.

Program example

Branching is controlled with signs.

```
10 INPUT A
20 ON SGN(A)+2 GOTO 30,40,50
30 PRINT "A<0" : GOTO 10
40 PRINT "A=0" : GOTO 10
50 PRINT "A>0" : GOTO 10
RUN
?-10
A<0
?0
A=0
?10
A>0
?
```

OPEN

(open random access file)

Opens a file to write or read data of a random access file.

Format

OPEN [# file number,] "[device name:] file name [.type]" [,record length]

$0 \leq \text{file number} \leq 127$

$1 \leq \text{record length} \leq 16384$

device name : DSK (disk) or PAC (ROM pack)

file name : Character string or string variable of up to eight characters

type : Character string or string variable of up to eight characters

} Numeric-constant, variable, array variable
or expressions including these

Application

A file must be opened with an OPEN statement before a data write/read operation can be performed. The OPEN statement assigns a file number to a file and the data are written or read using the file number.

Specification method

Specify any number between 0 and 127 as the file number. This number will be used to access the file until the file is closed.

Specify the device name according to the device on which the file is to be accessed. To specify a disk other than the current logged-in disk, write the drive name and a colon instead of "DSK:".

The file name is a character string up to eight characters and the type is a character string up to three characters.

The record length is specified in bytes.

When the specification is omitted

When the file number is omitted, number 0 is assumed to have been specified.

When the device name is omitted, the current logged-in disk is selected.

When the type is omitted, three spaces are specified.

When the record length is omitted, the length of 128 bytes is assumed to have been specified.

Execution example

Execution of

```
OPEN #1, "TEST.DAT"
```



accesses the file "TEST.DAT" and opens it. If the specified file does not exist, a file of the specified name is generated and opened. Data is written with a PUT statement and is read with a GET statement. When a GET statement is executed for a newly generated file, 0's are assigned to numeric variables and null strings are assigned to string variables.

If the OPEN statement is executed specifying a file number which has been opened,

```
Error(153) : File already open
```

will be displayed.

Program example

The following program writes string data into a file, reads the third and second data and display them on the screen.

```
10 OPEN #1,"DSK:TEST.DAT"
20 RECORD #1,LJUST 30 AS A$
30 PLAY "CDE"
40 PRINT "Write ASCII data to disk"
50 FOR I=0 TO 3
60 A$=STRING$(10+I*5,CHR$(I+65))
70 PRINT A$
80 PUT #1,I
90 NEXT
100 PLAY "EDC"
110 PRINT "Read ASCII data from disk"
120 FOR I=2 TO 1 STEP -1
130 GET #1,I
140 PRINT A$
150 NEXT
160 CLOSE

RUN
Write ASCII data to disk
AAAAAAAAAAAA
BBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDD
Read ASCII data from disk
CCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBB
```

```
* End *
Ready
```

OPEN/I (open to input)

Opens a sequential file on the specified device to read data.

Format

OPEN/I[# file number,] "[device name:] file name [.type]" [,buffer length]

$0 \leq \text{file number} \leq 127$

$128 \leq \text{buffer length} \leq 16384$

device name : DSK(disk), CTR(cassette), PAC(ROM pack), CON(console) or CHN(RS-232C)

file name : Character string or string variable of up to eight characters

type : Character string or string variable of up to three characters

Application

This command opens a file to read data which have been written by PRINT # statements. The data will be read by INPUT # or INPUT/L # statements.

Specification method

Specify any number between 0 and 127 as a file number. This number will be used to access the file until the file is closed.

Specify the device name according to the device from which the data are to be read. To specify a disk other than the current logged disk, write the drive name instead of "DSK:".

The file name is a character string up to eight characters and the type is a character string up to three characters.

Specify the buffer length in a multiple of 128.

The buffer length indicates the number of bytes to be read from the file at one time.

When the specification is omitted

When the file number specification is omitted, 0 is assumed to have been specified.

When the device name specification is omitted, the current logged disk is selected.

When the type is omitted, three spaces are specified.

When the buffer length specification is omitted, 128 bytes are specified.



Error messages

- When the specified file is not found on the disk or the ROM pack, or when the **ESC** key is pressed when a file is being searched for or when the file name received from the RS-232C and the specified file name do not match,

Error(10) : File not found

will be displayed.

- When the same file number is specified in different OPEN statements at the same time,

Error(153) : File already open

will be displayed.

- The data file whose data have been written by the SEND statement can be opened by an OPEN/I statement. As the data, however, have been written in the internal representation format, an error may occur during reading of data by an INPUT # statement.



OPEN/O

(open to output)

Opens a sequential file on the specified device to write data.

Format

OPEN/O [# file number,] "[device name:] file name [.type]" [,buffer length]

$0 \leq \text{file number} \leq 127$

$128 \leq \text{buffer length} \leq 16384$

device name : DSK(disk), CTR(cassette), CON(console), LST(list device) or CHN(RS-232C)

file name : Character string of string variable of up to eight characters

type : Character string or string variable of up to three character.

Application

This command opens a file to write data by PRINT # or PRINT # USING statements.

Specification method

Specify any number between 0 and 127 as a file number. This number will be used to access the file until the file is closed.

Specify the device name according to the device onto which the data are to be written. To specify a disk other than the current logged disk, write the drive name instead of "DSK:".

The file name is a character string up to eight characters and the type is a character string up to three characters.

Specify the buffer length in a multiple of 128. The buffer length indicates the number of bytes to be written into the file at one time.

When the specification is omitted

When the file number specification is omitted, 0 is assumed to have been specified.

When the device name specification is omitted, the current logged disk is selected.

When the type is omitted, three spaces are specified.

When the buffer length specification is omitted, 128 bytes are specified.

Program example

This program displays the data entered from the keyboard.

```
10 OPEN /O "CON:CONS.ASC"
20 INPUT A$
30 IF A$="END" THEN CLOSE :END
40 PRINT #0,A$
50 GOTO 20
```

OPEN/R (open to receive)

Opens a sequential file on the specified device to read data.

Format

OPEN/R[# file number,] "[device name:] file name [.type]" [,buffer length]

$0 \leq \text{file number} \leq 127$

$128 \leq \text{buffer length} \leq 16384$

device name : DSK(disk), CTR(cassette), PAC(ROM pack), CON(console) or CHN(RS-232C)

file name : Character string or string variable of up to eight characters

type : Character string or string variable of up to three characters

Application

This command opens a file to read data which have been written by SEND statements. The data will be read by RECEIVE statements.

Specification method

Specify any number between 0 and 127 as a file number. This number will be used to access the file until the file is closed.

Specify the device name according to the device from which the data are to be read. To specify a disk other than the current logged disk, write the drive name instead of "DSK:".

The file name is a character string up to eight characters and the type is a character string up to three characters.

Specify the buffer length in a multiple of 128.

The buffer length indicates the number of bytes to be read from the file at one time.

When the specification is omitted

When the file number specification is omitted, 0 is assumed to have been specified.

When the device name specification is omitted, the current logged disk is selected.

When the type is omitted, three spaces are specified.

When the buffer length specification is omitted, 128 bytes are specified.



Error messages

- When the specified file is not found on the disk or the ROM pack, when the **ESC** key is pressed when a file is being searched for or when the file name received from the RS-232C and the specified file name do not match,

Error(10) : File not found

will be displayed.

- When the same file number is specified in different OPEN statements at the same time,

Error(153) : File already open

will be displayed.

- The data file whose data have been written by the PRINT # statement can be opened by an OPEN/R statement. As the data, however, have been written in the ASCII format, an error may occur during reading of data by an RECEIVE statement.

Program example

This program reads the data of the file generated by executing the program in "OPEN/S".

```
10 OPTION BASE (1)
20 OPEN /R "ADDRESS"
30 RECEIVE N
40 DIM N$(N),A$(N),T$(N)
50 FOR I=1 TO N
60 RECEIVE N$(I),A$(N),T$(I)
70 NEXT I
80 FOR I=1 TO N
90 PRINT I
100 PRINT N$(I)
110 PRINT A$(I)
120 PRINT T$(I);PRINT
130 NEXT I
140 END
```

OPEN/S (open to send)

Opens a sequential file on the specified device to write data.

Format

OPEN/S [# file number,] "[device name:] file name [.type]" [,buffer length]

$0 \leq \text{file number} \leq 127$

$128 \leq \text{buffer length} \leq 16384$

device name : DSK(disk), CTR(cassette), CON(console), LST(list device) or CHN(RS-232C)

file name : Character string or string variable of up to eight characters

type : Character string or string variable of up to three characters

Application

This command opens a file to write data by SEND statements.

Specification method

Specify any number between 0 and 127 as a file number. This number will be used to access the file until the file is closed.

Specify the device name according to the device onto which the data are to be written. To specify a disk other than the current logged disk, write the drive name instead of "DSK:".

The file name is a character string up to eight characters and the type is a character string up to three characters.

Specify the buffer length in a multiple of 128.

The buffer length indicates the number of bytes to be written into the file at one time.

When the specification is omitted

When the file number specification is omitted, 0 is assumed to have been specified.

When the device name specification is omitted, the current logged disk is selected.

When the type is omitted, three spaces are specified.

When the buffer length specification is omitted, 128 bytes are specified.

Program example

This program generates a data file of the addresses for 50 persons.

```
10 OPTION BASE (1):Q=50
15 DIM N$(Q),A$(Q),T$(Q)
20 FOR I=1 TO 50
30 PRINT I,:PRINT :INPUT "NAME:      ",N$(I)
40 INPUT "ADDRESS:  ",A$(I)
50 INPUT "TEL No.   ",T$(I):PRINT
60 NEXT I
70 OPEN /S "ADDRESS"
80 SEND Q
90 FOR J=1 TO Q
100 SEND N$(J),A$(J),T$(J)
110 NEXT J
120 CLOSE
130 END
```

OPTION BASE (option base)

Sets the array variable subscript minimum value to 0 or 1.

Format

OPTION BASE X

X: 0 or 1

Application

When OPTION BASE 0 is executed, the variable subscripts of the arrays declared in a DIM statement begin with 0.

When OPTION BASE 1 is specified, the variable subscripts begin with 1.

Execution example

Execution of

```
OPTION BASE 0  
DIM A(10)
```

reserve eleven areas having array variables A (0) to A (10)

Execution of

```
OPTION BASE 1  
DIM A(10)
```

reserve ten areas A (1) to A (10)

Holding a set value

The value set with an OPTION BASE statement is held until a NEW, CLEAR, or RUN statement is executed.

If an attempt is made to define a different minimum value for each array variable or to execute OPTION BASE statement more than once in a program, "Duplicate definition" will be displayed.

Initial set upon entry of Sony disk BASIC

The minimum subscripts are initialized to 0. They are initialized when the Sony disk BASIC starts or when a NEW, CLEAR, or RUN statement is executed.

OUT (out)

Outputs data to the specified I/O port.

Format

OUT port number, data

$0 \leq \text{port number} \leq 255$	}	Numeric-constant variable, array variable, or expressions including these
$0 \leq \text{data} \leq 255$		

Application

This is a command to output data to an SMC-70 I/O port.

See “Appendix 7. I/O port assignment” in Part 1 and Hardware Reference Manual for I/O port assignments.

Execution example

I/O port 35 is assigned to the display screen border area.

Therefore, when

```
OUT 35,4
```

is executed, the border area becomes red.

Note

There is an INP function related to an OUT command to read arbitrary I/O port data.



PAINT

(paint)

Paints the specified color in the area which is enclosed with the specified color lines, and which contains the specified coordinates.

Format

PAINT (coordinates), [display color code], [boundary line color code]

$-16384 \leq \text{X-coordinate} \leq 16383$
 $-16384 \leq \text{Y-coordinate} \leq 16383$
 $-32768 \leq \text{color code} \leq 32767$

Numeric-constant, variable, array variable, or expressions including these

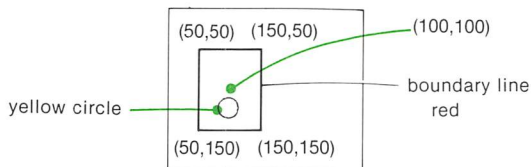
Application

Specification method

See the Reference Chart for the color code specifications. By specifying a negative number for the boundary line color code, the area specified will be completely painted except for any points of a color or any area enclosed with color different from the specified coordinates.

Execution example

Assume that a rectangle is drawn on the display screen with red lines and yellow circle is also displayed as follows :



PAINT (100,100),3,4

specifies painting the area enclosed in red boundary lines and containing point (100, 100) with turquoise. Therefore, when this statement is executed, the whole rectangle is painted turquoise and the yellow circle will disappear.

When

PAINT (200,100),3,4

is executed instead of the above statement, the area outside the rectangle is painted turquoise.

Assume the same display screen as illustrated above and when

PAINT (100,100),3,-1

is executed, the area inside the red rectangle is colored turquoise excluding the yellow circle and the area inside of the circle.

When the specification is omitted

When the display color code or both the display and boundary line color codes are omitted, the foreground color specified in the last executed GCOLOR statements is assumed to have been specified. If there has been no GCOLOR statement, they are set to white.

When only the boundary line color code is omitted, the specified display color code in the PAINT statement is also used for the boundary color.

When boundary lines are not closed

When the specified boundary line color code is positive, the entire screen except the boundary lines is painted. When the specified boundary color code is negative, all area of the same color as the specified coordinates are colored excluding the area enclosed with the color different from the specified coordinates.

When the specified coordinate is on a boundary line or the color of the specified point is the same as the display or boundary line color specified in a PAINT statement

The screen will not change even if a PAINT statement is executed.

PAUSE (pause)

Program execution pauses for the specified duration.

Format

PAUSE N

$0 \leq N \leq 255$: Numeric-constant, variable, array variable or expressions including these

Application

When N is specified, the program execution is stopped for N/10 seconds ($\pm 10\%$). For example, if 10 is set as the N, the execution will be stopped for 1 second and then be continued to the next statement. If N is not an integer, it is rounded n to the nearest whole number.

PEN READ

(pen read)

Reads position information when a light pen is touched to the screen.

Format

PEN READ [X] [,0 or 1]

X: Integer constant, variable, array variable or expressions including these

Application

A PEN READ statement reads the last vertical and horizontal positions on both the character display and the graphic display screens picked up by a light pen.

When PEN READ , 0 is executed and when a light pen has been touched to the screen, the last position will be read, even if a light pen is not used when the PEN READ , 0 statement is executed.

If a light pen has not been touched to the screen after the last PEN READ , 0 statement, the position will be read as (-1, -1).

When PEN READ , 1 is executed, the computer waits until a light pen is touched and reads the position information.

If 0 or 1 is omitted, 0 is assumed to have been specified.

The position information on the character display screen will be given with PENCLM and PENLIN functions and that on the graphic display screen will be given with PENX and PENY functions.

Execution example

To enter data alternately from the keyboard and with a light pen, execute the following program :

```
10 GCLEAR 15
20 INPUT A$
30 PEN READ,1
40 PRINT PENX,PENY
50 GOTO 20
```

X specification

X in a PEN READ statement specifies the horizontal correction value for a light pen delay.

Specify X if the PENCLM or PENX function value is set to 0, 1 or 2 by touching the right side of the screen, or to 39, 79, 319 or 639 by touching the left side of the screen.

When X is plus, the PENCLM or PENX function value decreases; when X is minus, the value increases.

The vertical position cannot be corrected.

When a border area which is not black is touched with a light pen

Meaningless data is read. If the border area is black, the function values will be set to -1. Refer to the Light Pen manual for light pen operations.

Program example

The following program makes it possible to draw any figure on the screen with a light pen.

```
10 WIPE:CURSOR OFF
20 LOCATE(0,0) :PRINT "CL"
30 GCOLOR,15 :GCLEAR
40 FOR X=1 TO 15
50 BOXF(X*20,0)-(X*20+19,10),X
60 NEXT
70 PEN READ
80 PEN READ : IF PENX<0 THEN 80
90 IF PENY<12 THEN IF PENX<20 THEN 30
    ELSE C=PENX\20 :GOTO 70
100 BOXF(PENX,PENY)-(PENX+1,PENY+1),C
110 GOTO 70
```

When this program is executed, 15 colors are displayed on the screen. When any point on the screen is touched after selecting any color, a point is displayed at the same position with the selected color. To delete the drawn figure, touch CL at the upper-left position on the screen.

PLAY (play)

Generates tones specified in the music subcommands.

Format

PLAY "subcommands"

subcommands : String type constant or variable or expressions including these

Application

A PLAY statement is a command to play music at the speed, length, and tone specified in music subcommands.

Music subcommands

Music subcommands are a list of the followings :

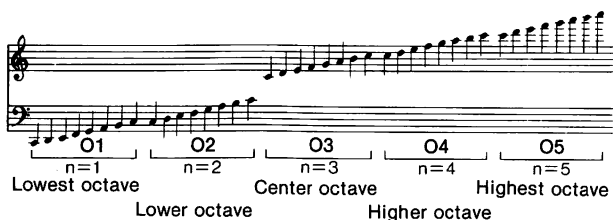
- Tn** (tempo):
- Specifies the speed (tempo) of the tone.
 - n is the number of quarter notes (L4) counted in a minute. When n is high, the tempo is high.
 - n is an integer in the range of 32 to 255.
 - The initial set is T120.
 - When Tn is omitted, the tempo previously set is used.
 - When n is omitted, "Macro language error" will be displayed.
- Ln** (length):
- Specifies the tone length.
 - Ln indicates an nth note. Ln succeeded by a period indicates a dotted nth note.
 - n is an integer in the range of 1 to 64.
 - The initial set is L4.
 - When Ln is omitted, the tone length previously set is used.
 - When n is omitted, "Macro language error" will be displayed.

	Whole note	L1			
	Half note	L2		Dotted half note	L2.
	Quarter note	L4		Dotted quarter note	L4.
	Eighth note	L8		Dotted eighth note	L8.
	Sixteenth note	L16		Dotted sixteenth note	L16.
	Thirty-second note	L32		Dotted thirty-second note	L32.
	Sixty-fourth note	L64		Dotted sixty-fourth note	L64.

On

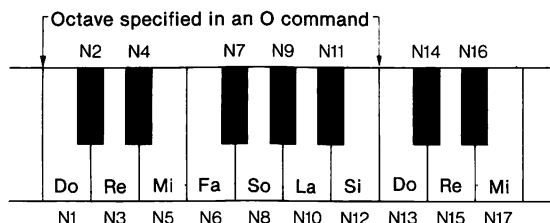
(octave):

- Specifies the octave.
- There are five octaves. n specifies the nth octave from the lowest octave.
- n is an integer in the range of 1 to 5.
- The initial set is O3.
- When On is omitted, the octave previously specified is used.
- When n is omitted, "Macro language error" will be displayed.

**Nn**

(note):

- Specifies the note in the specified octave.



- The available n range is decided by the O command specification.
n must be an integer in the following range :
 - When O1 is specified, $1 \leq n \leq 60$
 - When O2 is specified, $1 \leq n \leq 48$
 - When O3 is specified, $1 \leq n \leq 36$
 - When O4 is specified, $1 \leq n \leq 24$
 - When O5 is specified, $1 \leq n \leq 12$
- When n is omitted, "Macro language error" will be displayed.

- When N is written with a small letter n, it indicates that the next octave above the one specified with the O command.

N13 = n1

N14 = n2

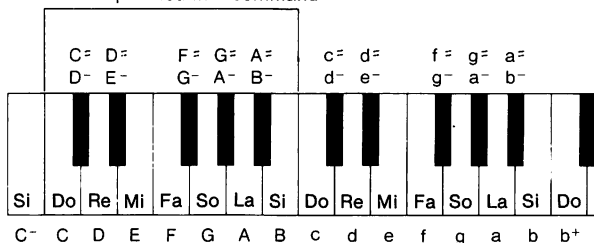
N24 = n12

- In addition to an N command, a tone can be specified with letters A to G and # (or +) or - indicating a halftone.

When A to G are written with small letters a to g, they indicate the next octave above the one specified in the O command.

When A to G or small a to g is used to specify a tone, the length of each tone can be set by adding the value n after each tone. The specification method is the same as that of L command.

Octave specified in Ocommand



Pn

(pause):

- Specifies a pause.
- Pn indicates an nth rest and Pn. indicates a dotted nth rest.
- n is an integer in the range of 1 to 64.
- When n is omitted, 4 is assumed to be specified.

	Whole rest	P1		
	Half rest	P2		Dotted half rest P2.
	Quarter rest	P4		Dotted quarter rest P4
	Eighth rest	P8		Dotted eighth rest P8.
	Sixteenth rest	P16		Dotted sixteenth rest P16.
	Thirty-second rest	P32		Dotted thirty-second rest P32.
	Sixty-fourth rest	P64		Dotted sixty-fourth rest P64.

Subcommands T, O and L may be specified in an arbitrary order before N.

Tn	}	Nn
On		
Ln		

Tn, On, and Ln may be omitted.

When tones are specified as character data in a PLAY statement, the specified tones are played in the specified order.

Execution example

```
PLAY "T120L4N1N1N8N8N10N10L4.N8P8
      L4N6N6N5N5N3N3L4.N1P8"
```

Variables

- String variables can be used as string data in a PLAY statement. In this case, enclose the string variables in brackets [].
For example, the program for playing "Twinkle twinkle little star" is as follows:

```
10 M1$="L4CCGGAAL4.GP8L4FFEEDDL4.CP8
20 M2$="L4GGFFEEL4.DP8"
30 PLAY "T12004[M1$][M2$][M2$][M1$]"
```

The \$ in the variable enclosed in brackets may be omitted.

- A variable may also be specified in the n of a subcommand. In this case, enclose the variable in parentheses to separate the variable from the command.

Add the following to the above program:

```
5 FOR O=1 TO 5
30 PLAY "T1200(O)[M1$][M2$][M2$][M1$]"
40 NEXT O
```

"Twinkle twinkle little star" can be played at different octaves by changing the variable in subcommand O.

Repetition of the music subcommands

When a set of music subcommands are to be repeated in a PLAY command, specify the number of times, write subcommands and enclose them with "<" and ">".

For example, to repeat playing do to do 5 times, execute the following PLAY statement.

```
PLAY "<5CDEFGABc>"
```

To write graphic subcommands in a PLAY command

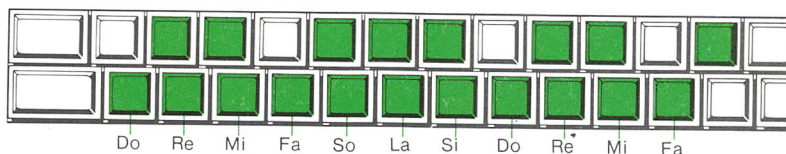
Write subcommand ^G and write graphic subcommands. To switch to the music subcommands again, write ^M. In this way, you can switch from playing music to drawing a graphic and vice versa easily.

Program example

Tones are generated from the keyboard with the following program :

```
10 A$=INKEY$(1)
20 N=INSTR("AWSEDFGTGYHUJKOLP;`' ",A$)
30 IF N=0 THEN 10
40 PLAY "T160N(N)"
50 GOTO 10
```

When this program is executed, the console keyboard operates as a musical keyboard.



- To stop playing the music, press the ESC key.

POKE (poke)

Writes data at the specified memory address.

Format

POKE address, data

$0 \leq \text{address} \leq 65535$	}	Numeric-constant, variable, array variable, or expressions including these
$0 \leq \text{data} \leq 255$		

Application

Execution example

When

```
POKE 30000,16
```

is executed, value 16 is written at memory address 30000.

When

```
POKE &H7000,&HFF
```

is executed, value &HFF is written at memory address &H7000.

Note

- If a POKE statement is executed thoughtlessly, the Sony disk BASIC may not operate correctly. Specify addresses carefully. (See Appendix 3 in Part 1 for the memory configuration.)
- Related to the POKE command there is a PEEK function to read the specified memory contents.

PRINT

(print)

Displays numeric or character data on the character display screen.

Format

PRINT [(coordinates)] [separator] [X] [separator] [Y] [separator]...

$0 \leq x\text{-coordinate} \leq 39$ or 79

$0 \leq y\text{-coordinate} \leq 24$

X, Y : Numeric or string-constant variable, array variable, or expressions including these

separator : Comma (,) or semicolon (;)

} Numeric constant, variable, array variable, or expressions including these

Application

Execution example

In a PRINT statement, enclose only a string constant in quotation marks (") and write other constants and variables directly.

```
PRINT 3
3
```

```
PRINT 3+5
8
```

```
PRINT "ABC"
ABC
```

```
PRINT A
0
```

The numeric variable value A is displayed.

```
PRINT N$
FUKUDA
```

The string variable value N\$ is displayed. Nothing is displayed when N\$ is a null string.

P

Display of floating-point data

If floating-point data has 15 or more significant digits, the 15th and subsequent digits are rounded off, then stored in the memory.

A PRINT statement displays floating-point data in the following format :

- ① When there are 17 or more digits after the decimal point
(-)X.XXXXXX...E-XX
- ② When there are 16 or fewer digits after the decimal point
(-)X.XXXXXXXXXX...
- ③ When there are 13 or fewer digits in the integer part
(-)XXXX.XXXX
- ④ When there are 14 or more digits in the integer part
(-)X.XXXXXXXXXXXE + XX

Example

```
10 A=1.23456789012E-8
20 FOR I=1 TO 24
30 PRINT A
40 A=A*10
50 NEXT I
```

RUN

```
1.23456789012E-8      } In case of ①
1.23456789012E-7
.00000123456789012
.0000123456789012
.000123456789012
.00123456789012
.0123456789012
.123456789012
1.23456789012
12.3456789012
123.456789012
1234.56789012
12345.6789012
123456.789012
1234567.89012
12345678.9012
123456789.012
1234567890.12
12345678901.2
123456789012
1234567890120
12345678901200
1.23456789012E+14    } In case of ②
1.23456789012E+15    }
```

The plus (+) sign is not displayed, but the minus (-) sign is displayed.

Multi data in a PRINT statement

When more than one data value are listed in a PRINT statement, separate data with commas (,) or semicolons (;). These symbols are called separators. The display format varies with the separator as follows :

When the separator is a comma (,)

The next data is displayed in the next 8-column tab position.

```
PRINT "A" , "B"
A          B
PRINT 123456789 , 123
123456789      123
```

When

```
PRINT 12345678. , 123
```

is executed, the cursor is first placed at the left side of the screen, and then displays 1, 2, 3, ... sequentially. When 8 is displayed, the cursor is in column 8. Since tab is set with a comma, the next data is displayed from column 16. The execution results are as follows :

```
12345678      123
```

When the separator is a semicolon (;)

The next data is displayed succeeding the current data.

```
PRINT "A" ; "B"
AB
```

```
PRINT 2 ; "+" ; 3 ; "=" ; 2+3
2+3=5
```

When a separator is at the end of a PRINT statement

If a separator is not written at the end of a PRINT statement, the data in the next PRINT statement will be displayed in the next line ; if it is written, data in the next PRINT statement will be displayed in the same line.

When no data is written in a PRINT statement

Only line feed occurs.

Example

```
10 A=3:B=5:C=7
20 PRINT A,B,
30 PRINT C
40 PRINT A,B
50 PRINT C
60 PRINT
70 PRINT C
RUN
3          5          7
3          5
7
7
```

Since there is a comma at the end, the next PRINT statement displays the data C value in the same line.

Since there is no separator at the end, the data C value is displayed in the next line.

Only line feed and carriage return occur. Therefore, a blank line is generated.

When coordinates on the character display screen are specified in a PRINT statement

The cursor is moved to the specified coordinates and the data is displayed.

Use of the CHR\$ function

A CHR\$ function may be written in a PRINT statement.

PRINT CHR\$(7) : A "beep" sound is generated.

PRINT CHR\$(9) : The cursor moves to the tab position.

Use of ?

A question mark (?) may be specified instead of PRINT.

PRINT A,B,C

?A,B,C

When listing, the question mark (?) is converted to PRINT.

PRINT @ PRINT @ USING

(print @)
(print @ using)

Sends numeric or character data to the specified device.

Format

PRINT @ N, [X] [separator] [Y] [separator]...

PRINT @ N, USING format characters, [X] [separator] [Y] [separator]...

N : 0- To the RGB or B/W MULTI OUT connector.	} Constant, variable, array variable, or expressions including these
1- To the PRINTER, PARALLEL connector.	
2- To the RS-232C connector.	

X, Y : Numeric or string-constant, variable, array variable, or expressions including these

separator : Comma (,) or semicolon (;)

format character : See "PRINT USING".

Application

This is a command to send data to a device connected to the RGB MULTI OUT and B/W MULTI OUT connectors, the PRINTER, PARALLEL connector or the RS-232C connector of the SMC-70.

Specification method

Specify the output destination in N.

When the specification of N is omitted, the data are sent to the display screen.

Data, separator and format character specifications and the output format are the same as those for a PRINT or PRINT USING statement.

```
PRINT A,B,C
```

```
PRINT @0,A,B,C
```

The above two statements are equivalent and they display the value of A, B and C on the display screen.

Use of ?

A question mark (?) may be specified instead of PRINT.

```
PRINT @1,A,B,C
```

```
?@1,A,B,C
```

The above two statements sends data to the printer connected to the PRINTER, PARALLEL connector.

When listing, the question mark (?) is converted to PRINT.

PRINT # PRINT # USING

(print number)

(print number using)

Writes data sequentially in a file opened with an OPEN/O statement.

Format

PRINT # file number [, X] [separator] [Y] [separator]...

PRINT # file number, USING format characters, [,X] [separator] [Y] [separator]...

$0 \leq \text{file number} \leq 127$

Numeric constant, variable, array variable or expressions including these

X, Y, Z... : Numeric- or string constant, variable, array variable or expressions including these

format characters : Refer to "PRINT USING".

Application

Before data is written in a sequential file with this command, the file must be opened with OPEN/O statement.

```
OPEN/O #1, "DATA.BAS"  
PRINT #1, X, Y, Z
```

Specify the file number in PRINT # statement which is used to open the file in OPEN/O statement.

When a file number is omitted in OPEN/O statement, specify #0 as the file number.

The data listed in the PRINT # statement are written sequentially. After the write operation is completed, execute a CLOSE statement.

Specification method and data write operation

Specifications, such as expressions or format characters, can be made in the same way as in a PRINT or a PRINT USING statement. The data are written sequentially in the specified file. The format in which the data is written in a file is the same as that displayed on the screen with the PRINT or PRINT USING statement.

Execution example

The following program generates a file having the name DATA.BAS on the current disk, then writes 10 units of string data - A to J separated by commas.

```
10 OPEN /O #1, "DATA.BAS"  
20 FOR I=1 TO 10  
30 PRINT #1, CHR$(&H40+I);", ";  
40 NEXT I  
50 PRINT #1  
60 CLOSE
```

In line 30, a “,” is written as the delimiter to separate the string data so that the written data are handled as ten separated data when they are read by an INPUT # statement. If the “,” is omitted, the written data “ABCDEFGHJIJ” is handled as one datum.

Error messages

- If a PRINT # statement is executed before an OPEN/O statement has been executed, or if different file numbers are specified in the OPEN and PRINT # statements

```
Error(155) : File not open
```

will be displayed.

- If a PRINT # statement is executed for a file which has been opened with an OPEN statement other than OPEN/O

```
Error(151) : Bad file access
```

will be displayed.

- If an error occurs during writing data,

```
Error(12) : File write error
```

will be displayed.

Note

The PRINT # statement can be used in the same way as the SEND statement.

However, when the file has been opened with an OPEN/S statement, a SEND statement must be used and when the file has been opened with an OPEN/O statement, a PRINT # statement must be used.

A SEND statement writes data in the internal representation format while a PRINT # statement writes data in the ASCII format.



PRINT USING

(print using)

Displays numeric and string data in the specified format on the screen.

Format

PRINT USING format characters, expressions

format characters : String constant, variable, array variable or expressions including these, indicating the display format

expressions : A list of string type or numeric type separated by a comma or a semicolon.

(A comma and a semicolon are interchangeable.)

Application

The values of the expressions will be displayed in the format specified by the format characters in the quotation marks.

The characters other than format characters in the quotation marks will be displayed as they are written.

Format characters

String data display format

- ! • The first character of the string data is given.

```
10 PRINT USING "!", "ABC", "DEF", "GHI"
20 PRINT USING "<!>", "ABC", "DEF", "GHI"
RUN
ADG
<A><D><G>
```

- | n spaces | • Enclose n spaces with “| |”.
- The n + 2 characters from the left is displayed and the rest are ignored. If the string data is shorter than n + 2 characters, the string data is displayed from the left and the remaining spaces are kept blank.

```
10 PRINT USING "<!!!>", "ABC"
20 PRINT USING "<|    |>", "ABC"
RUN
<AB>
<ABC  >
```


&

- The string data is displayed as it is typed.

```
PRINT USING "<&>", "ABC"  
<ABC>
```

Numeric data display format

#

- Specify "#" as many times as there are digits to be displayed. "." indicates the decimal point.

```
PRINT USING "###.##", 123.45  
123.45
```

- If the number of the digits on the left of the decimal point is less than the number of #, the data is displayed with the right-most digit justified and the remaining spaces are kept blank. If it is more than the number of #, the numeric data is displayed as it is typed, preceded by "^^"

```
10 PRINT USING "###", 1  
20 PRINT USING "###", 10  
30 PRINT USING "###", 100  
40 PRINT USING "###", 1000  
RUN  
1  
10  
100  
^^1000
```

```
10 PRINT USING "###", 1, 2, 3  
20 PRINT USING "###"; 1; 2; 3  
RUN  
1 2 3  
1 2 3
```

```
10 PRINT USING "### ##", 123, 456  
20 PRINT USING "item 1=### item  
2=### ", 123, 456, 789, 12  
run  
123 456  
item 1=123 item 2=456 item 1=789  
item 2= 12
```

- If the number of the digits on the right of the decimal point is less than the number of #, spaces will succeed. If it is more than the number of #, they will be rounded.

```
PRINT USING "###.##",123.4,123.45,123.456
123.40123.45123.46
```

- A plus sign of the numeric data is ignored. A minus sign is counted as one digit.

```
10 PRINT USING "###",123.4
20 PRINT USING "###",123.5
30 PRINT USING "####",-123.5
40 PRINT USING "####",-123.6
RUN
123
124
-123
-124
```

+

- Specify before or after the format characters.
- "+" for a positive data or "-" for a negative data is displayed before or after the data corresponding to the specification.

```
10 PRINT USING "+#####",1234,-5678
20 PRINT USING "#####+",1234,-5678
RUN
+1234 -5678
1234+ 5678-
```

P

-

- Specify before or after the format characters.
- A space for a positive data or "-" for a negative data is displayed before or after the data corresponding to the specification.

```
10 PRINT USING "-#####",1234,-5678
20 PRINT USING "#####-",1234,-5678
RUN
1234 -5678
1234 5678-
```

*

- Specify at the beginning of the format characters.
- The leading spaces are filled with "*".
- One "*" is counted as one digit to be displayed.

```
PRINT USING "#####", 1234, -5678
***1234**-5678
```

\$

- Specify immediately before the format characters "#".
- The data is succeeded by a "\$".
- One "\$" is counted as one digit to be displayed.

```
10 PRINT USING "$##### ", 123456, -345678
20 PRINT USING "-$##### ", 123456, -345678
30 PRINT USING "+$##### ", 123456, -345678
40 PRINT USING "$#####- ", 123456, -345678
RUN
**$123456 *-$345678
***$123456 **-$345678
***$123456 **-$345678
**$123456 ***$345678-
```

,

(comma)

- Specify a comma at any place between quotation marks.
- Commas are displayed on the left of the every third digits from the decimal point. A comma is not displayed on the right of the decimal point.
- One "," is counted as one digit to be displayed.

```
PRINT USING "###,### ", 12345, -67890
12,345 -67,890
PRINT USING ",,##### ", 123456, 1234567
123,456 ^^1234567
```

P

PUT (put)

Writes data of one record in a random access file according to the RECORD statement specification.

Format

PUT # file number [,record number]

$0 \leq \text{file number} \leq 127$

$0 \leq \text{record number} \leq 65535$

Application

Specification method

Specify the same file number as in the OPEN statement. When the file number is omitted in the OPEN statement, specify #0.

In record number, specify which record is to be written.

If the specification of the record number is omitted, record number following that of the record accessed by the last PUT or GET statement will be selected. When the PUT statement is to be executed right after an OPEN statement, 0 is specified as the record number.

Data write operation

Before writing data with a PUT statement, an OPEN and a RECORD statement must be executed.

Execution example

Execution of

```
PUT #1,3
```

Puts the values of the variables in the format specified by RECORD as the 4th record onto a file device.

When, for example,

```
RECORD #1,A%,LJUST 5 AS S$
```

has been executed and the value of A% is &H4634 and the value of S\$ is "ABC" at this time, a record with the record number 3 is written on to a file device. The contents of the record are as follows.

&H34	&H46	&H41	&H42	&H43	&H20	&H20
------	------	------	------	------	------	------

&H4634 A B C space space

RANDOMIZE (randomize)

Changes the generation sequence of random numbers obtained by an RND function.

Format

RANDOMIZE [N]

N: Numeric-constant, variable, array variable, or expressions including these

Application

Random numbers obtained by an RND function are generated with the same sequence.

For example, the following program generates 5 random numbers in the same order every time the program is executed.

```
10 FOR N=1 TO 5
20 PRINT RND
30 NEXT N
```

```
RUN
.93631576461867
.83096996030588
.27666054804669
.4005443924171
.56569845015311
```

A RANDOMIZE statement changes the random number generation sequence.

Specification

Specify the random number generation sequence in N.

Execution example

```
10 RANDOMIZE 1
20 FOR N=1 TO 5
30 PRINT RND
40 NEXT N
RUN
.14861598197605
.76657390864406
.53053579229367
.49584616598088
.85696141172169
```

```
10 RANDOMIZE 2
20 FOR N=1 TO 5
30 PRINT RND
40 NEXT N
RUN
.24861598197605
.86657390864406
.63053579229367
.59584616598088
.95696141172169
```

```
10 RANDOMIZE -1
20 FOR N=1 TO 5
30 PRINT RND
40 NEXT N
RUN
.14861598197605
.76657390864406
.53053579229367
.49584616598088
.85696141172169
```

```
10 RANDOMIZE -2
20 FOR N=1 TO 5
30 PRINT RND
40 NEXT N
RUN
.24861598197605
.86657390864406
.63053579229367
.59584616598088
.95696141172169
```

R $N=1$ and $N=-1$, $N=2$ and $N=-2$, ... specify the same sequence.

When the specification is omitted

Random numbers are generated with a different sequence with each program execution.

```
10 RANDOMIZE
20 FOR N=1 TO 5
30 PRINT RND
40 NEXT N
```

RUN

```
.64684454424222
.75587826519743
.56399310108624
.34361037728465
.03079323932866
```

RUN

```
.17966278276044
.01234006883085
.83935524026686
.55554921156447
.00355017657968
```

RUN

```
.79205095569559
.45501541289
.11350037529001
.28160433179162
.09256895465083
```

Note

- The random number generation sequence can also be changed by specifying a minus value in N of an RND(N) function. In this case, RND (-1) and RANDOMIZE -1, RND (-2) and RANDOMIZE -2, ... specify the same generation sequence.



READ

(read)

Reads the data specified in a DATA statement, and then assigns it to a variable.

Format

READ variable [,variable] [,variable]...

variable : Numeric or string-variable or array variables

Application

When a READ statement is executed, the first data of the DATA statement having the lowest line number in the program is assigned to the variable.

Execution example

```
10 READ A
.
.
.
50 DATA 5
```

Numeric data 5 is assigned to numeric variable A.

Multi variable in a READ statement

When listing more than one numeric or string variable in a READ statement, separate variables with commas (,).

Variables are assigned data sequentially beginning with the first data in the DATA statement. When there is no data to be read from the DATA statement, data is read from the DATA statement having the second lowest line number.

For example when the programs below are executed, data are assigned as indicated.

```
10 READ A,B,C
.
.
.
50 DATA 5,10,20
```

Variables	A	B	C
Assigned data	5	10	20


```

10 READ A,B,C,D$,E$
.
.
.
50 DATA 5,10,20
.
.
70 DATA ABC,XYZ

```

Variables	A	B	C	D\$	E\$
Assigned data	5	10	20	ABC	XYZ

When there is more than one READ statement in a program

The first data of the DATA statement having the lowest line number is assigned to the first variable of the READ statement executed first. The next data is assigned to the next variable, ... then the next READ statement reads from the data following the data read by the preceding READ statement.

Example

When the program below is executed,

```

10 READ A,B,C,D$,E$
20 READ F$,G
.
.
.
.
100 DATA 5,10,20
110 DATA ABC,XYZ,SSS
120 DATA 30,50,100

```

data are assigned as follows :

Variables	A	B	C	D\$	E\$	F\$	G
Assigned data	5	10	20	ABC	XYZ	SSS	30

Type of variable and data

The data type must match the related variable type.

If an attempt is made to assign string data to a numeric variable, "Non numeric" will be displayed.

When an attempt is made to assign numeric data to a string variable, the data is assumed to be of the character type.

Insufficient or excessive number of data sets for variables

When data in a DATA statement is insufficient for the variables in the READ statement, "Out of DATA" will be displayed.

When there are excessive data, they are ignored.

When a RESTORE statement is executed

The next READ statement reads the DATA statement having the lowest line number after the line number specified in the RESTORE statement.

READ or DATA statement in the direct command mode

When a READ statement is executed in the direct command mode if there is a DATA statement in the program currently in memory, the data in this DATA statement will be read.

If there are no DATA statements, "Out of DATA" will be displayed. A DATA statement executed in the direct command mode is ignored.

RECEIVE (receive)

Reads data sequentially from a file opened with an OPEN/R statement, then assigns the data to the variables.

Format

RECEIVE [# file number,] variable [,variable] [,variable]...

$0 \leq \text{file number} \leq 127$

variable : Numeric or string variable

Application

This command is used for a file which is generated by an OPEN/S and a SEND statements.

Before data are read from a sequential file with this command, the file must be opened with an OPEN/R statement.

```
OPEN/R #1, "DATA.BAS"
RECEIVE #1, X, Y, Z
```

Specify the same file number in RECEIVE as in OPEN/R. If the specification of the file number is omitted in the OPEN/R statement, omit the file number specification or specify 0 in the RECEIVE statement.

The values read are sequentially assigned to the variables listed in the RECEIVE statement. After the read operation is completed, execute a CLOSE statement.

Execution example

To read 10 values which have been saved with the name DATA. BAS, execute the following program :

```
10 DIM A(9)
20 OPEN/R #1, "DATA.BAS"
30 FOR I=0 TO 9
40   RECEIVE #1, A(I)
50 NEXT I
60 CLOSE
```

The values will be assigned to variables A(0) to A(9).
When reading ends, the prompt mode will be generated.

Matching of variable and data

The type of the variable and data to be received must match.

If the read data type does not match the variable type in the RECEIVE statement, a value is assigned assuming that the data is of the same type as the variable type. If, however, the data is received as an floating-point data format but it cannot be a floating point datum in the computer (for the floating-point data format, see Appendix 3 in Part 1.) an error will occur.

Error messages

- If the last datum of the file has been read and you attempt to read another data,

```
Error(17) : Read past EOF
```

will be displayed.

- If a RECEIVE statement is executed before an OPEN/R statement has been executed or different file numbers have been specified in the OPEN and RECEIVE statements,

```
Error(155) : File not open
```

will be displayed.

- If a RECEIVE statement is executed for a file which has been opened with an OPEN statement other than OPEN/R,

```
Error(151) : Bad file access
```

will be displayed.

- If an error occurs during reading of data,

```
Error(14) : File read error
```

will be displayed.

RECORD

(record)

One record of an random access file is allocated to the variables.

Format

RECORD # file number, field specification

$0 \leq \text{file number} \leq 127$

field specification : See below.

Application

Before accessing a random access file, a RECORD statement must be executed.

Specification method

In the file number, specify the same number which has been used to open a file in the OPEN statement.

When the file number is omitted in the OPEN statement, specify #0. In the field specification, numeric variables are simply listed separated by commas. An integer value uses 2 bytes and a floating point value uses 8 bytes. String variables are written as follows :

LJUST
RJUST number of bytes AS string variable

This statement specifies

- String variables whose values are written in/read from a random access file
- Number of bytes (characters) to be allocated to each variable
- The writing of the value in the allocated field

Use of array variables

Array variables can be written in a RECORD statement.

When

```
RECORD #1,A%(5)
```

is executed, integer array variables A%(0) to A%(5) are declared. If the variables have been declared by DIM and they are declared in a RECORD statement,

```
Error(133) : Duplicate definition
```

will be displayed.

Execution program

```
OPEN #1, "TEST.DAT"
```

specifies 128 bytes for one record. If this statement has been executed, variables up to 128 bytes can be allocated.

When

```
RECORD #1,A%(63)
```

is executed, 64 integer array variables of 2 bytes each are allocated.

When

```
RECORD #1,A,B,RJUST 20 AS A$(2)
```

is executed, floating point variables A and B of 8 bytes each and 3 string array variables of 20 bytes each are specified.

Error messages

- If a RECORD statement was executed before an OPEN statement has been executed or different file numbers are specified in the OPEN and RECORD statements,

```
Error(155) : File not open
```

will be displayed.

- If the total bytes allocated in the RECORD statement is greater than the record length specified in the OPEN statement,

```
Error(158) : RECORD too long
```

will be displayed.

REM

(remark)

Adds a comment to a program list.

Format

REM comment

Application

A REM statement makes it possible to add a comment so that the program listing can be easily read.

Enter an arbitrary comment after the space following REM.

The REM statement is output by listing, but it is ignored during execution.

Use of apostrophe (')

A comment can be entered after entering an apostrophe (') instead of REM.

```
10 REM MUSIC
20 .....
30 .....
```

```
10 ' MUSIC
20 .....
30 .....
```

When an apostrophe is used after a statement in a line, colon (:) need not be written to separate these two statements.

```
10 PRINT X :REM Output X
10 PRINT X 'Output X
```

REM DEBUG

(remark debug)

Statements in the REM DEBUG are executed only in the debug mode.

Format

REM DEBUG statement : statement :.....

statement : Sony disk BASIC statement

Application

When a program is executed, a REM DEBUG statement is simply regarded as a REM statement. The statements in REM DEBUG are regarded as comments. In the debug mode, however, the statements in REM DEBUG become executable.

RENAME

(rename)

Changes a file name and type of a file saved on a disk.

Format

RENAME "[drive name:] old file name [.type]" TO "[drive name:] new file name [.type]"

drive name : a character from A to P

file name : character string of up to 8 characters

type : character string of up to 3 characters

Application

Specification method

Use the RENAME command to change the name of a particular file while keeping the file contents as they are.

In "drive name", specify the name of the disk drive (A - P) in which the disk containing the specified file is inserted.

If the disk in question is the currently logged disk, the "drive name" specification can be omitted.

Execution example

```
RENAME "A:OLDNAME.BAS" TO "NEWNAME.BAS"
```

The above command will rename OLDNAME.BAS as NEWNAME.BAS on the disk in drive A. If the drive A is the currently logged disk, the "drive name" specification can be omitted ; simply execute :

```
RENAME "OLDNAME.BAS" TO "NEWNAME.BAS"
```

If the specified new name already exists on the disk,

```
Error(159) : File already exists
```

will be displayed.

R

RENUM (renumber)

Renumbers the specified line in the stored program.

Format

RENUM [new line number], [old line number], [increment]

$0 \leq \text{line number} \leq 65529$	} Integers
$1 \leq \text{increment} \leq 65529$	

Application

A RENUM statement is used to renumber the lines after program modification.

Execution example

Assume the following program :

```
3 WIPE
5 RANDOMIZE
10 X=RND*319
20 Y=RND*199
25 FOR I=0 TO 199
28 C=RND*6+1
30 LINE (X,Y)-(0,I),C
35 LINE (X,Y)-(319,199-I),C
40 NEXT I
```

Execution of

```
RENUM 10,3,10
```

changes line number 3 to 10, and renumber each successive line numbers at increment of 10.

After execution of this statement, the above program is listed as follows :

```
10 WIPE
20 RANDOMIZE
30 X=RND*319
40 Y=RND*199
50 FOR I=0 TO 199
60 C=RND*6+1
70 LINE (X,Y)-(0,I),C
80 LINE (X,Y)-(319,199-I),C
90 NEXT I
```

When the specification is omitted

When the "new line number" is omitted, 10 is assumed to have been specified; when the "old line number" is omitted, the first program line is assumed to have been specified. When the "increment" is omitted, 10 is assumed to have been specified.

For example,

RENUM specifies changing all line numbers to line numbers having increments of 10, beginning with 10.

RENUM 100,,100 specifies changing all line numbers to line numbers having increments of 100, beginning with 100.

RENUM 100 specifies changing all line numbers to line numbers having increments of 10, beginning with 100.

Line numbers specified in GOTO and GOSUB statement etc.

The line numbers specified in these types of statements in the program are also changed by RENUM execution. If the line specified in these types of statements does not exist before RENUM statement execution, the RENUM statement will be executed. However, the line numbers in these types of statements will remain unchanged, and the "Undefined line number XXX" will be displayed.

For example, assume the following program :

```
40 INPUT A
50 ON SGN(A)+2 GOTO 100,200,300
```

When

RENUM

is executed, the following are displayed :

```
Error(132) in 20: Undefined line number 100
Error(132) in 20: Undefined line number 200
Error(132) in 20: Undefined line number 300
```

And the list after execution of RENUM is as follows :

```
10 INPUT A
20 ON SGN(A)+2 GOTO 100,200,300
```



If the line number specified in these types of statements does not exist before RENUM statement execution and the line having the same line number as that specified in these types of statements is supposed to be generated by RENUM execution, RENUM will be executed, and the "Undefined line number XXX...Over-defined" will be displayed.

For example, assume the following program :

```
10 INPUT A
20 ON SGN(A)+2 GOTO 60,70,80
30 PRINT "A<0"
40 PRINT "A=0"
50 PRINT "A<0"
```

When

RENUM 40

is executed, the following are displayed :

```
Error(132) in      50 : Undefined line
number          60 ... Over-defined
Error(132) in      50 : Undefined line
number          70 ... Over-defined
Error(132) in      50 : Undefined line
number          80 ... Over-defined
```

And the list of the program will be as follows :

```
40 INPUT A
50 ON SGN(A)+2 GOTO 60,70,80
60 PRINT "A<0"
70 PRINT "A=0"
80 PRINT "A<0"
```

If the line sequence is changed by RENUM execution,
"Illegal function call" will be displayed.

For example, assume the following program :

```
100 INPUT A
200 ON SGN(A)+2 GOTO 300,400,500
300 PRINT "A<0"
400 PRINT "A=0"
500 PRINT "A>0"
```

If

```
RENUM 10,300
```

is executed to change the line numbers 300, 400 and 500 to 10, 20, and 30 respectively, the line sequence will be changed ; therefore an error will occur.

```
Error(3) : Illegal function call
```

RESTORE (restore)

Returns the data to be read with the next READ statement to the first data of the DATA statement having the lowest line number after the specified line number.

Format

RESTORE [line number]

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

Use a RESTORE statement to read the same data in the DATA statement more than once. If there is more than one READ and DATA statements in a program, data will be read sequentially from the DATA statement having the lowest line number.

When a RESTORE statement is executed, the next READ statement reads beginning with the first data in the DATA statement having the lowest line number after the specified line number.

Specification method

In "line number", specify the line number of a line in the program.

Execution example

```
10 READ A,B,C$,D$
20 READ E,F
30 RESTORE 200
40 READ G,H
```

.

```
100 DATA 1,2,X,Y
200 DATA 3,4
300 DATA 5,6
```

When the above program is executed, data is assigned as follows :

Variables	A	B	C\$	D\$	E	F	G	H
Assigned data	1	2	X	Y	3	4	3	4

When the specification is omitted

The DATA statement having the lowest line number is read.

When a nonexistent line is specified

Error(132) : Undefined line number

is displayed.

RESUME (resume)

Returns control to the main routine after the execution of the error correction routine.

Format

RESUME [line number or NEXT]

line number : Integer in the range between 0 and 65529 or a label specifying a line

Application

Specification method

The specification after RESUME specifies the line to return.

When the line number or the label of the line is specified, the control returns to that line.

When NEXT is specified, the operation returns to the first statement after the error.

When both line number and NEXT are omitted, the operation returns to the statement where the error occurred.

If RESUME is executed without an error having occurred

Error(146) : RESUME without error

will be displayed and the program will stop.

RJUST (right justify)

Used in a RECORD statement and specifies the right justification of a string variable value setting.

Format

RJUST number of bytes AS string variable

number of bytes : Numeric-constant, variable, array variable, or expressions including these

Application

Execution example

When

```
RECORD #1,RJUST 100 AS A$
```

is executed, 100 bytes are allocated for the variable A\$ and the value is set into the allocated area with right justification. Execution of

```
RECORD #1,RJUST 10 AS B$(10)
```

specifies 11 array variables B\$(0) or B\$(10) with 10 bytes each. The values of these array variables are set with right justification.

When the length of the variable values are longer than the length specified in RECORD, the extra characters are ignored.

Program example

The following program is to write data in a file "TEST.DAT"

```
10 OPEN #1,"TEST.DAT"  
20 LENG=10  
30 RECORD #1,RJUST LENG AS A$(2),RJUST LENG*2 AS B$  
40 A$(0)="ABC"  
50 A$(1)="0123456789abc"  
60 A$(2)="defgh"  
70 B$="SONY BASIC"  
80 PUT #1  
90 CLOSE
```

After the execution of this program, the file contents is displayed by a TYPE command.

```
TYPE "TEST.DAT"
```

```
ABC0123456789
```

```
defgh
```

```
SONY BASIC
```

Each variable value is written into the specified allocation with right justification. The last three characters of the variable A\$(1) value are ignored.

RUN (run)

- 1 Executes the program from the specified line.
- 2 Loads the specified Sony BASIC program from the device and executes it.

Format 1

RUN [line number]

line number : Integer in the range of 0 to 65529 or a label specifying a line

Application

When a RUN statement is executed, all numeric variables are set to 0s, and string variables are set to null strings. Array variables are also placed in the undefined state, and then the program is executed. After the program execution is completed, the prompt mode is set.

Specification method

In "line number", specify the starting line number in the program or a label indicating the starting line.

When the specification is omitted

The program is executed from the first line.

To temporarily stop program execution

Press the **ESC** key. Program execution is restarted by executing a CONT statement in the direct command mode.

Format 2

RUN "[device name:] file name [.type]"

device name : DSK (disk), CTR (cassette tape) or PAC (ROM pack)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

Application

When

RUN "[device name:] file name [.type]"

is executed, the program in memory and the variables are cleared and the files on the specified file device are closed.

Then the specified file is loaded into memory and is executed.

Specification method

Specify DSK, CTR or PAC for the device name according to the file device. To select the disk which is not the currently logged in disk, specify the drive name instead of "DSK:".

Specify the same file name and type as those when the file was saved.

When the specification is omitted

When the device name specification is omitted, the current logged in disk is assumed to have been specified. When the type specification is omitted, "BAS" is assumed to have been specified.

When the specified program is not found

A "file not found" message will appear. When a tape cassette is being used, the computer continues to search for the specified file until you press the **ESC** key. After you press the **ESC** key, the message will appear. The memory contents before the RUN statement is executed will be held.

When an error occurs during loading

"Check-sum error" or "File read error" is displayed. When the program to be loaded was saved with SAVE or CSAVE, the memory contents will be completely cleared when the error occurs. When the program to be loaded was saved in the ASCII format with SAVE/A or CSAVE/A, the lines loaded before the error will not be cleared.

SAVE

(save)

The Sony BASIC program in memory is saved onto the specified device.

Format

SAVE [/A or /P] [/C] "[device name:] file name [.type]" [,start address, end address]

device name : DSK (disk), CTR (cassette tape), CON (console), LST (list device) or CHN (RS-232C)

file name : Character string or string variable of eight or fewer characters

type : Character string or string variable of three or fewer characters

0 ≤ start address, end address ≤ 65535 :

Numeric-constant, variable, or expressions including these

Application

To save the Sony BASIC program onto a currently logged disk, execute

SAVE "DSK : file name [.type]"

If "DSK :" is omitted, the result will be the same.

Specification of the device name, file name and type

Specify as the device name the device onto which the program is to be saved. If the specification of the device name is omitted, the disk on the current disk drive is selected. To specify a disk on another drive, write the drive name instead of "DSK :".

The file name can be any character string of eight or fewer characters. If nine or more characters are used, the excess characters will be ignored.

The type can be any character string of three or fewer characters. When the type is omitted, "BAS" is specified.

Note on specification of the file name and type

If the specified file name and type are the same as those of a program already saved on the disk, the program to be saved will be replaced with the program on the disk.

If the specified file name and type includes special characters such as % or ?,

Error(13) : Bad filename

will be displayed.

Execution example

Execute

```
SAVE "TEST.BAS"
```

File name "TEST.BAS" is generated on the current disk and the Sony BASIC program currently in memory is saved. When the saving has been completed, the prompt mode will be re-entered. In this case, the memory contents will be held.

To save a Sony BASIC program with ASCII character strings

Execute

```
SAVE/A "[device name:] file name [.type]"
```

For example, execute

```
SAVE/A "TEST1.ASC"
```

A program file named "TEST1.ASC" is generated on the current disk with ASCII codes. Any type may be specified, but "ASC" is often used to reference the type of the file later.

The program to be merged with a MERGE command or a CHAIN/M command must be saved with the ASCII format.

To save a program so that it will not be listed when loaded

Execute

```
SAVE/P "[device name:] file name [.type]"
```

For example, execute

```
SAVE/P "SAMPLE.BAS"
```

A program file named "SAMPLE.BAS" is generated on the current disk. Later, the program can be loaded with a LOAD or CLOAD command, but cannot be listed with a LIST command. The SAVE/P statement is used to prevent another person from gaining access to the program list.

To save commands at the same time as the program is saved (so that, for instance, sound is played to tell when the program loading is completed.)

Execute

```
SAVE [/A or /P] /C "[device name:] file name [.type]"
```

For example, execute

```
SAVE/C "SAMPLE.BAS"
```

Then,

Enter Startup Command Line(s)

will be displayed on the screen.

Enter any commands. For example, enter

```
:PLAY "N1N2N3N4N5"
```

and then press **RETURN**. Two or more commands can be entered; commands up to 128 bytes can be entered. To stop entry, press **RETURN**. The input commands are saved together with the Sony BASIC program. If the program is later loaded, these commands will automatically be executed after completion of the loading.

To save a machine language subroutine simultaneously

Execute

```
SAVE [/A or /P] [/C] "[device name:] file name [.type]",  
start address, end address
```

The contents of memory from the start address to the end address will be saved together with the Sony BASIC program. The low and high memory addresses will also be saved simultaneously. When a program is loaded, the low and high memory addresses are automatically set and the subroutine is loaded together with the program.

SEND (send)

Writes data sequentially in a file opened with an OPEN/S statement.

Format

SEND [# file number,] X[,Y] [,Z]...

$0 \leq \text{file number} \leq 127$

X, Y, Z... : Numeric- or string constant, variable, array variable or expressions including these

Application

Before data can be written in a sequential file with this command, the file must be opened with an OPEN/S statement.

```
OPEN/S #1,"DATA.BAS"  
SEND #1,X,Y,Z
```

Specify the same file number in SEND which is used to open the file in OPEN/S. If the specification of the file number is omitted in the OPEN/S statement, omit the file number specification or specify 0 in the SEND statement.

The values listed in the SEND statement are written sequentially in the format of internal representation. After the writing operation is completed, execute a CLOSE statement.

Execution example

The following program generates a file having the name DATA.BAS on the current disk and then writes 10 data entered from the keyboard.

```
10 DIM A(9)  
20 FOR I=0 TO 9  
30 INPUT A(I)  
40 NEXT I  
50 OPEN/S #1,"DATA.BAS"  
60 FOR N=0 TO 9  
70 SEND #1,A(N)  
80 NEXT N  
90 CLOSE
```

When writing ends, the prompt mode will be re-entered.

Error messages

- If a SEND statement is executed before an OPEN/S statement has been executed or if different file numbers have been specified in the OPEN/S and SEND statements,

Error(155) : File not open

will be displayed.

- If a SEND statement is executed for a file which has been opened with an OPEN statement other than OPEN/S,

Error(151) : Bad file access

will be displayed.

- If an error occurs during writing of data,

Error(12) : File write error

will be displayed.

Note

A PRINT # statement writes data in the ASCII format while a SEND statement writes in the internal representation format of the data.

SET DATE (set date)

Sets the built-in timer.

Format

SET DATE "date", "time", "day of week" [, "mode"]

data : Six numeric characters
 indicating the year, month, and
 day
time : Six numeric characters
 indicating the hour, minute,
 and second
day of week : Three characters
 indicating the day of the
 week
mode : 0 or AM...in the morning
 1 or PM...in the afternoon

} String-constant, variable, or expressions
including these

Application

Specification method

In "date", specify the month, and the day of the month with two digits each and the last two digits of the current Gregorian calendar year (MMDDYY).

In "time", specify the hour in the 24-hour system or in the 12-hour system with "AM" or "PM" specified, minute and second with two digits each (HHMMSS).

In "day of week", specify the day of the week with three letters as follows :

Sunday : SUN
Monday : MON
Tuesday : TUE
Wednesday : WED
Thursday : THU
Friday : FRI
Saturday : SAT

Execution example

To set December 1, 1982, Wednesday, 5:30:30 p.m., specify one of the followings :

```
SET DATE "120182", "173030", "WED"  
SET DATE "120182", "053030", "WED",  
"PM"
```

When the **RETURN** key is pressed, the timer starts counting from the set time.

Note

Since the built-in timer is backed up with a rechargeable battery, it continues counting even when the computer's power is off.

SET TIME (set time)

Sets the time of the built-in timer.

Format

SET TIME "time" [,"mode"]

time : Six numeric characters indicating
hour, minute, and second

mode : 0 or AM...in the morning
1 or PM...in the afternoon

} string constant, variable, array
variable, or expressions including
these

Application

Specification method

In "time", specify the hour in the 24-hour system or in the 12-hour system with "AM" or "PM" specified, minute and second with two digits each (HHMMSS).

Execution example

To set 5:00:00 p.m., execute as follows :

```
SET TIME "170000"
```

or

```
SET TIME "050000", "PM"
```

When the **RETURN** key is pressed, the timer starts counting from the set time.

Note

Since the built-in timer is backed up with a rechargeable battery, it continues counting even when the computer's power is off.

STOP

(stop)

Stops program execution and sets the prompt mode.

Format

STOP

Application

When a STOP statement is executed, program execution stops, and the prompt mode is set. "Break in XXX" is displayed indicating the interrupted line number.

Restarting program execution

When a CONT statement is executed in the direct command mode, execution restarts from the statement after the interrupted statement.

SWAP

(swap)

Exchanges the values of the two variables.

Format

SWAP variable, variable

variable : Numeric or string type variable or array variable

Application

Execution example

When

```
10 A=3:B=5
20 SWAP A,B
30 PRINT A,B
```

is executed, the values are exchanged and the value of A will be 5 and the value of B will be 3.

Type of the variables

The type of the two variables must be the same. If the values specified are not of the same type, then a

```
Error(5) : Type mismatch
```

will be displayed.

Program example

Program to arrange ten numbers from the smallest to the largest:

```
10 DIM A(9)
20 RANDOMIZE
30 FOR I=0 TO 9
40   A(I)=RND*100
50   PRINT "A(";I;")=";A(I)
60 NEXT
70 PRINT
80 /
90 FOR I=8 TO 0 STEP -1
100   FOR J=0 TO I
110     IF A(J)>A(J+1)
120       THEN SWAP A(J),A(J+1)
120   NEXT J
130 NEXT I
140 /
150 FOR I=0 TO 9
160   PRINT "A(";I;")=";A(I)
170 NEXT
```

} Generate ten random numbers.

} Sorting

} Displays the results.

In this program, numbers are arranged by so called "Bubblesort".

The FOR-to-NEXT loop from line 100 to 120 gets the largest number and assigns the number to the array with the largest subscript, then the second largest number to the array variable with the second largest subscript and so on.

I is used as the loop counter.

"Quicksort" or "Heapsort" can also be used for arranging numbers depending of the number of data.

SYSTEM (system)

Transfers control to the CP/M.

Format

SYSTEM

Application

When a SYSTEM command is executed while Sony disk BASIC is in the prompt mode, the control is transferred to the CP/M mode. The Sony disk BASIC programs in the memory are all cleared.

TYPE (type)

Displays the contents of a file saved on a disk with ASCII format.

Format

TYPE "[drive name:] file name [.type]"

drive name : a character from A to P

file name : character string of up to 8 characters

type : character string of up to 3 characters

Application

Specification method

In "drive name", specify the name of the disk drive (A - P) in which the disk containing the file to be displayed is inserted.

If the disk in question is the currently logged disk, the "drive name" specification can be omitted.

Execution example

To display contents of a file existing on the disk in drive A, execute

```
TYPE "A:KEYBOARD.ASC"
```

The contents of the file with ASCII format under the file name and type "KEYBOARD.ASC" will be displayed on the display screen.

```
TYPE "A:KEYBOARD.ASC"
```

Note

- If you try to use the TYPE command to display a file which has not been saved with ASCII format, a meaningless character string will be displayed.
- The TYPE command only reads the specified file from the disk and displays it on the display screen. The file is not loaded in memory after execution of the TYPE command. If another file has been loaded in memory, this file will remain in memory after a TYPE command execution.

USER (user number)

Changes the logical user area of the CP/M on the currently logged disk.

Format

USER user number

user number : Integer from 0 to 15

Application

It is possible to use a disk as if it was divided into 16 different areas, (logical user area) numbering each of them from 0 to 15.

When Sony disk BASIC is activated, the user area is automatically set to 0. Normally it is not necessary to keep this user area number in mind. However, once you use the USER command to set another user area (for example user area 3), the disk space is divided into two areas.

"No file" will be displayed by the DIR command after changing the user area from 0 to 3 by USER 3 command, if this USER 3 command is first applied to the disk. Make and save a file at this stage; this file will be saved in the user area 3.

The DIR command will display this file name. Return to the user area 0 by USER 0 command; the DIR command at this stage will not display the file saved in the user area 3.

By switching the user area, one disk can be handled as several disks.

Execution example

Execute

DIR

PROGA	.BAS	5K
PROGA	.ASC	7K
PROGB	.XYZ	1K
PROGC	.BAS	4K
DATA	.NO1	1K
DATA	.NO2	1K

All files are displayed by the DIR command. When

USER 3

is executed, then

DIR

is executed,

Error(10) : File not found

will be displayed. There are no files in the user area 3. If files have been saved in the user area 3, the file names are displayed.

USER 0

DIR

PROGA	.BAS	5K
PROGA	.ASC	7K
PROGB	.XYZ	1K
PROGC	.BAS	4K
DATA	.NO1	1K
DATA	.NO2	1K

After returning to the user area 0, it is possible to display the files and apply any disk operating commands to the files.



VOUT

(video RAM out)

Outputs data to the specified video RAM address.

Format

VOUT address, data

$0 \leq \text{address} \leq 65535$
 $0 \leq \text{data} \leq 255$

} Numeric-constant, variable, array variable,
or expressions including these

Application

This is a command to output data to the SMC-70 video RAM. See Appendix 8 in Part 1 for Video RAM address, and refer to the Hardware Reference Manual for details.

Execution example

```
VOUT 0,65
```

displays character A at the upper-left corner on the screen.

```
VOUT &H800,&H47
```

blinks character A.

```
VOUT &H8000,&H70
```

displays a white point at the upper-left corner on the screen.

When an address outside the range allocated to the video RAM is specified

Data is output to the I/O port. For example, the execution of VOUT &H23FF,4 will give the same result of the execution of OUT &H23,4.

Note

Related to a VOUT command, there is a VINP function to read data at an arbitrary video RAM address.

Program example

The following program defines an arbitrary character pattern for an arbitrary character code. (This is equivalent to a DEF FONT statement.)

```

10 INPUT "CH=",CH
20 INPUT "PATTERN=",P$
30 AD=&H1000+CH*8
40 FOR I=0 TO 7
50 DA$="&H"+MID$(P$,I*2+1,2)
60 D=VAL(DA$)
70 VOUT AD+I,D
80 NEXT I

```

```

RUN
CH=45
PATTERN=0C0A09090A387870

```

```

* End *
Ready
PRINT CHR$(45)
)

```

WHILE...WEND

(while...wend)

Executes the statements between WHILE and WEND statements as long as the expression in the WHILE statement is true (not 0).

Format

WHILE expression

:

WEND

Application

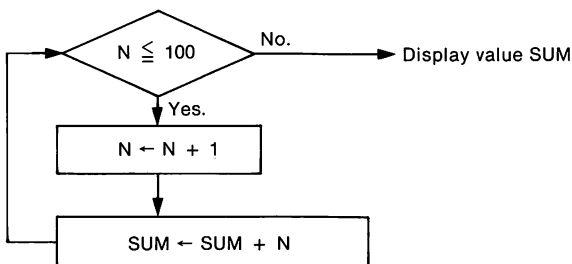
The expression may be a relational, logical, or arithmetic expression if the result is of numeric type.

When a WHILE statement is executed, the value of the expression is obtained. If the result is true (not 0), the statements up to the WEND statement are executed and the control returns to the WHILE statement; if the result is false (0), the statement succeeding WEND will be executed.

Execution example

```
10 WHILE N<=100
20   N=N+1
30   SUM=SUM+N
40 WEND
50 PRINT SUM
```

This program will be executed as follows :



The WHILE and WEND statements must be used in pairs. This repeated execution procedure is called a loop.



Error messages

- If a WHILE statement does not have a corresponding WEND statement,

Error(143) : WHILE without WEND

will be displayed.

- If a WEND statement does not have a corresponding WHILE statement,

Error(144) : WEND without WHILE

will be displayed.

If control is transferred within the loop with a statement such as GOTO, the program will be executed from the line to which control was transferred; however, the WEND statement does not have a corresponding WHILE statement.

Error(144) : WEND without WHILE

will be displayed. Control can be transferred from the inside to the outside of the loop.

- If a WHILE statement is executed in the direct command mode,

Error(143) : WHILE without WEND

will be displayed. When WHILE and WEND statements are used in the direct command mode, write them in one line and separate them with colons (:).

Multi-loop

The WHILE-to-WEND loop can contain another WHILE-to-WEND loop creating a multi-loop.

The internal loop must be completely within the external loop.

```
WHILE .... }
  WHILE .... }
  :          }
  :          }
  WEND      }
WEND      }
```

WIPE

(wipe)

Initializes the screen display.

Format

WIPE

Application

A WIPE statement has the following functions :

- Erasing characters from the character display screen.
- Erasing graphics from graphic display screen.
- Initializing the CCOLOR and GCOLOR statement specifications.
- Expanding the scrolling area to the whole screen.
- Setting scrolling on.
- Placing the cursor at the upper-left corner on the screen.

CHAPTER 2

SONY DISK BASIC FUNCTIONS

This chapter explains the Sony disk BASIC functions in alphabetical order. The following format is used for explanation :

Function name
Function (value which is given)

Format

Indicates the correct syntax.

The specification enclosed in brackets [] may be omitted.

Application

The method for writing a function and applications are explained in detail.

Program example

Gives a program example using the function.

ABS

(absolute)

A

Gives the absolute value X ($|X|$) of numeric data X .

Format

ABS (X)

X: Numeric-constant, variable, array variable, or expressions including these

Application

The ABS function is as follows:

When $X \geq 0$: $\text{ABS}(X) = X$

When $X < 0$: $\text{ABS}(X) = -X$

Example

```
PRINT ABS(2)
```

```
2
```

```
PRINT ABS(-2)
```

```
2
```

```
PRINT ABS(10-3)
```

```
7
```

```
PRINT ABS(3-10)
```

```
7
```

ANN (ann)

Gives the value obtained by executing the machine language subroutine beginning with the address defined in a DEF ANN statement.

Format

ANN [X] (I)

X: Integer in the range of 0 to 9.

I: Numeric or string- constant, variable, or array variable

Application

Specification method

In "X", specify the value defined in a DEF ANN statement.

In "I", specify the variable or constant indicating the value to be transferred from the Sony disk BASIC program to the subroutine.

Example

Assume that these statements are executed.

```
DEF ANN0=&HC000
```

```
DEF ANN1=&HC500
```

To execute the subroutines from &HC500 and to return the value to the Sony disk BASIC program, execute the following:

```
X=ANN1 ( I )
```

See Appendix 5 in Part 1 for DEF ANN statement and ANN function operations and how to transfer values to the subroutine.

When specification is omitted

When X is omitted, 0 is assumed to have been specified.

APOINT

(attribute point)

Gives the attribute (character color, background color, and attribute code) for the character displayed at the specified location (X,Y) on the character display screen.

Format

APOINT (X,Y)

0 ≤ X ≤ 79 (80-character display)	} Numeric-constant or variable, array variable, or expressions including these
0 ≤ X ≤ 39 (40-character display)	
0 ≤ Y ≤ 24	

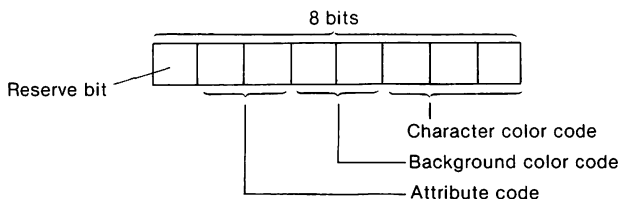
Application

Specification method

Specify the coordinates on the character display screen.

Attribute

An attribute is a list of the binary character color, background color, and attribute codes for the specified location on the character display screen. The format is as follows :



See Reference Chart for the codes.

The APOINT function converts this binary code to decimal values for display.

Example

Assume that the execution of

```
PRINT APOINT(10,10)
```

gives 75.

The binary notation 01001011 of 75 indicates the codes of the specified location (10,10) as follows :

	Binary	Decimal	
Character color code :	011	→ 3	→ Turquoise
Background color code :	01	→ 1	→ White
Attribute code :	10	→ 2	→ Blinking

The value given with the APOINT function is

an integer in the range of 0 to 255.

When an integer between 128 and 255 is given, the MSB bit of the attribute data is set. In this case, the lower 7 bits are ignored and the attribute is set as follows:

character color: White

background color: Transparent

attribute: Not blinking

Program example

The following program obtains the attribute of a character displayed at an arbitrary location (X,Y) entered from the keyboard:

```
10 INPUT "X,Y=";X,Y
20 C=APOINT(X,Y)
30 ATR=C\32
40 C=C MOD 32
50 BACK=C\8
60 CHR=C MOD 8
70 PRINT "ATTRIBUTE=";ATR
80 PRINT "BACKGROUND COLOR=";BACK
90 PRINT "CHARACTER COLOR=";CHR
RUN
X,Y=10,5
ATTRIBUTE=2
BACKGROUND COLOR=2
CHARACTER COLOR=2
```

ASC

(ASCII)

Gives the character code of the first character of string data X\$.

Format

ASC (X\$)

X\$: String-constant, variable, array variable, or expressions including these

Application

Example

```
10 X$="S"  
20 A1=ASC(X$)  
30 A2=ASC("S")  
40 A3=ASC("STU")  
50 PRINT A1,A2,A3  
RUN  
83      83      83
```

Be sure to enclose the character constant in quotation marks (") as shown above; otherwise, an error will occur.

```
PRINT ASC(S)
```

Error(5) : Type mismatch

The value given with an ASC function is
an integer in the range of 0 to 255.

When character data is a null string

An error occurs.

```
PRINT ASC("")
```

Error(3) : Illegal function call

Note

See Appendix 9 in Part 1 for the characters and the character codes.

ATN (arc tangent)

The value of the inverse trigonometric function arc tangent, $\tan^{-1}(X)$ of numeric data X, is given.

Format

ATN (X)

X: Numeric-constant, variable, array variable, or expressions including these

Application

The ATN function gives an floating-point type numeric data, indicating that the trigonometric tangent of the data is X. Its unit is a radian.

Example

```
PRINT ATN(1)
.7853981633974
```

To convert the ATN function result to degrees

Multiply $180/\pi$.

```
PRINT ATN(1)*180/PI
44.9999999999996
```

Mathematically, $\tan^{-1}(1) = \pi/4 \text{ rad} = 45^\circ$

In this example, an error occurs in the internal operation. It is because the internal operation is performed using a floating-point representation and the 15th digit is rounded.

CADR

(convert from address data)

Address data X is converted to the integer type.

Format

CADR (X)

$0 \leq X \leq 65535$: Integer-constant, variable, array variable, or expressions including these

Application

When numeric data X is of the integer type, the CADR function will give the data as it is. When X is of the floating-point type, it will be rounded to the nearest whole number, and then converted to an integer type. If X is from 32768 to 65535 which exceeds the integer type range (−32768 to 32767), no error will occur ; the value of CADR is between &H8000 and &HFFFF internally and is displayed as a negative value (−32768 to −1). CADR function is used when memory addresses are handled.

C

CFLT (convert to floating)

Numeric data X is converted to the floating-point type.

Format

CFLT (X)

X: Numeric-constant, variable, array variable, or expressions including these

Application

When numeric data X is of the floating-point type, the CFLT function will give the data as it is. When X is of the integer type, it will be converted to the floating-point type.

Example

To send data to the disk as a floating point type, execute

```
SEND CFLT(1)
```

To transfer floating-point data to a machine language subroutine, execute

```
X=ANN0(CFLT(1))
```


CHR\$ (character dollar)

The string data whose character code is X is given.

Format

CHR\$(X)

$0 \leq X \leq 255$: Numeric-constant, variable, array variable, or expressions including these

Application

Example

```
10 FOR I=65 TO 90
20 PRINT CHR$(I);
30 NEXT I
RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The number of the character given by the CHR\$ function is one.
If the specification of X is not an integer type, it will be rounded to the nearest whole number.

```
PRINT CHR$(65.3)
A
PRINT CHR$(65.5)
B
```

Special character codes

Of character codes 0 to 31, there are special character codes having special functions such as screen control.

Example:

```
PRINT CHR$(7)      ... Sound an audible beep sound
PRINT CHR$(9)      ... Moves the cursor to the tab
                    position.
PRINT CHR$(13)     ... Performs a carriage return.
```

These can be used as follows:

```
PRINT CHR$(9);:PRINT "A"
A
DEF KEY 1, "RUN"+CHR$(13)
```

Note

See Appendix 9 in Part 1 for the character codes and characters.

CINT

(convert to integer)

Numeric data X is converted to integer type.

Format

CINT (X)

$-32768 \leq X \leq 32767$: Numeric-constant, variable, array variable, or expressions including these

Application

When numeric data X is of the integer type, the CINT function will give the data as it is. When X is of the floating-point type, it will be rounded to the nearest whole number, then given as an integer type.

Example

To send numeric data after conversion to an integer type, specify the following :

F=1.234

·
·
·

SEND CINT(F)

To transfer integer data to a machine language subroutine, specify the following :

F=1.234

·
·
·

I=ANN0 (CINT(F))

COND (condition)

Obtains the value of the first expression. If the result is true (not 0), COND gives the result of the second expression; if false (0), COND gives the result of the third expression.

Format

COND ([expression1], [expression2], [expression3])

expression 1: a relational, logical or arithmetic expression if the result is of a numeric type.

expression 2 and 3: an arithmetic or string expression.

Application

The value given by this function will be the numeric or the string type depending on the type of expression 2 or 3.

When the specification is omitted

When the expression 1 is omitted, the expression 2 is always evaluated.

When the result of the expression 1 is always true, the expression 3 can be omitted; when the result is always false, the expression 2 can be omitted.

In other cases, if the specification of an expression is omitted,

Error(130) : Syntax error

will be displayed.

Example

```
10 A$="A":B$="B"
20 INPUT X
30 PRINT COND(X>0,A$+B$,3+4)
40 GOTO 20
RUN
?1
AB
?-2
7
?0
7
```

In this example, when a positive number is entered as a value of X, the first expression "X>0" is true and the string value of A\$ + B\$ is obtained. When a negative number including 0 is entered, the first expression is false and the numeric value of 3 + 4 is obtained.

Program example

The factorial value of a number can be obtained with the following program.

```
10 DEF FNFACT(N)=COND(N=0,1,
    N*FNFACT(N-1))
20 INPUT N
30 PRINT FNFACT(N)
RUN
?5
120
```

For details about the factorial value see "DEF FN".

COS

(cosine)

The value of the trigonometric function cosine of numeric data is given.

Format

COS (X)

X: Numeric-constant, variable, array variable, or expressions including these
(Unit: radian)

Application

Value given by COS function

This value is floating-point numeric data in the following range:

$$-1 \leq \text{COS}(X) \leq 1$$

The internal operation is performed using a floating-point representation.

Example

```
PRINT COS(PI/3)  
.500000000000007
```

When degree is used as the unit of angle

COS (D°) can be obtained from COS (D*PI/180)

```
PRINT COS(60*PI/180)  
.500000000000007
```

Note

Mathematically, $\text{COS } \pi/3 = \text{COS } 60^\circ = 0.5$

In this example, an error occurs because the internal operation is performed and the 15th digit is rounded.

CPOINT

(character point)

Gives the character code of the character displayed at the specified position (X, Y) on the character display screen.

Format

CPOINT (X, Y)

$0 \leq X \leq 79$ (80-character mode)	}	Numeric-constant, variable, array variable, or expressions including these
$0 \leq X \leq 39$ (40-character mode)		
$0 \leq X \leq 24$		

Application

Specification method

Specify the coordinate (X, Y) on the character display screen.

Example

When

```
PRINT CPOINT(39,24)
```

is executed when character A is displayed at the lower-right corner of the screen in the 40-character mode, 65 will be given.

Value given by CPOINT function

Integer from 0 to 255

When a position where no character is displayed is specified

32 (space code) is given.

Note

See Appendix 9 in Part 1 for the character codes and characters.

CSRCLM (cursor column)

The current horizontal position of the cursor is given.

Format

CSRCLM

Application

The value given by the CSRCLM function is the X-coordinate on the character display screen.

Value given by CSRCLM function

0 to 79 (80-character mode)	}	Integer
0 to 39 (40-character mode)		

CSRLIN (cursor line)

The current vertical position of the cursor is given.

Format

CSRLIN

Application

The value given by the CSRLIN function is the Y-coordinate on the character display screen.

Value given by CSRLIN function

0 to 24 (integer)

DATE\$ (date dollar)

The current date (month, day and year) can be given in the specified format.

D

Format

DATE\$ [(N)]

$0 \leq N \leq 2$: Integer type-constant, variable, array variable or expressions including these

Application

Specification method

The current month, day and year will be given as 2 digits each (MM, DD, and YY) according to the value N:

Value N	Data format
0	MMDDYY
1	MM/DD/YY
2	YY/MM/DD

When specification of N is omitted

0 is assumed to have been specified.

Example

Assume that today is December 25, 1982.

```
PRINT DATE$(0)
122582
PRINT DATE$(1)
12/25/82
PRINT DATE$(2)
82/12/25
PRINT DATE$
122582
```

Note

There is a SET DATE command for setting the built-in timer.

DAYS\$ (day dollar)

The current day of the week can be given with string data of three characters or string data of 0 to 6.

Format

DAYS\$ [(N)]

$0 \leq N \leq 1$: Integer type-constant, variable, array variable or expressions including these

Application

Specification method

The current day of the week will be given according to the value of N :

data format		day of week
When N = 0	When N = 1	
SUN	0	Sunday
MON	1	Monday
TUE	2	Tuesday
WED	3	Wednesday
THU	4	Thursday
FRI	5	Friday
SAT	6	Saturday

When the specification of N is omitted

0 is assumed to have been specified.

Example

Assume that today is Tuesday.

```
PRINT DAY$(0)
TUE
PRINT DAY$(1)
2
PRINT DAY$
TUE
```

Note

There is a SET DATE command for setting the built-in timer.

EOF

(end of file)

Gives -1 (true) when the last datum in the file has been read ; otherwise, 0 (false) is given.

Format

EOF [(file number)]

$0 \leq \text{file number} \leq 127$: Numeric-constant, variable, array variable,
or expressions including these

Application

This statement is used to determine whether the last datum of a sequential file has been read.

If another datum is to be read after the last datum has been read,

```
Error(17) : Read past EOF
```

will be displayed.

Example

```
IF EOF(1) THEN CLOSE 1
```

statement closes the file 1 when the last datum in the file 1 has been read.

When EOF is used for a file which has not been opened

```
Error(155) : File not open
```

will be displayed.

When the specification of the file number is omitted

0 is assumed to have been specified.

ERL

(error line)

Gives the number of the line in which an error has occurred.

Format

ERL

Application

The ERL is the reserved variable to which the number of the line where an error has occurred is assigned automatically.

If an error has not occurred, 0 is given as the variable value and if an error has occurred in the direct command mode, 65535 is given as the variable value.

The value of ERL is cleared when the value of the variables are cleared by executing the CLEAR command and so on or another error occurs.

E

ERR

(error)

1. Gives the number of the error which has occurred.
2. Simulates the occurrence of an error of the error number specified.

Format 1

ERR

Application

The ERR is the reserved variable to which the number of the error is assigned.

If an error has not occurred, 0 is given as the variable value. The value of ERR is cleared when the RESUME statement is executed, or the value of the variables are cleared.

```
PRINT 10/0
Error(2) : Division by zero
PRINT ERR
2
```

```
A="ABC"
```

```
Error(5) : Type mismatch
PRINT ERR
5
```

Note

See Chapter 3 for the numbering of errors.

Format 2

ERR(N)

$1 \leq N \leq 255$:

Numeric constant, variable, array variable or expressions including these

Application

This function is used to occur an error using a function.

When an integer between 1 and 127 is specified as N, the control can be transferred by ON ERROR GOTO statement.

When an integer between 128 and 255 is specified, the execution is interrupted even if an ON ERROR GOTO statement has been executed and the prompt mode is re-entered.

Example

```
10 DEF FNFONT$(C)=COND(C<0 OR 255<C,ERR(3),PROG(SETQ(N,&H1000+C*8),SETQ(L,7),FNVINP$(N,L)))
20 DEF FNVINP$(N,L)=COND(L<0,"",FNVINP$(N,L-1)+HEX$(VINP(N+L),2))
30 INPUT C
40 PRINT FNFONT$(C),CHR$(C)
50 GOTO 30
```

The function FNFONT\$ is defined to give the font pattern of the given character code C. If a negative value or a value over 255 is given as the character code C, "Illegal function call" will be displayed by executing ERR(3). The function FNVINP\$ is defined to give the contents of the video RAM at addresses N + L to N. N is the start address of the PCG storing the data for the character of code C. L is used as the loop counter.

EXP

(exponential)

Exponential function e^x of numeric data X is given.

Format

EXP (X)

$X \leq 147.36544595161$: Numeric-constant, variable, array variable, or expressions including these

Application

e is the base of a natural logarithm.

The value given by the EXP function is a floating-point numeric data.

Example

```
PRINT EXP(1)
2.718281828459
```

```
PRINT EXP(100)
2.6881171418211E+43
```

```
PRINT EXP(0)
1
```

FIX

(fix)

The integer part of numeric data X is given.

Format

FIX (X)

X: Numeric-constant, variable, array variable, or expressions including these

Application

The FIX function gives a value obtained by omitting the figure after the decimal point of numeric data X, so that an integer value is given.

Example

```
PRINT FIX(3),FIX(-3)
3      -3
```

```
PRINT FIX(3.58),FIX(-3.58)
3      -3
```

Difference between FIX (X) and INT (X)

When $X \geq 0$, $\text{FIX}(X) = \text{INT}(X)$

When $X < 0$, $\text{FIX}(X) = \begin{cases} \text{INT}(X) & (\text{Figures after the decimal point are 0s.}) \\ \text{INT}(X) + 1 & (\text{Others}) \end{cases}$

```
10 PRINT FIX(3),FIX(-3)
20 PRINT INT(3),INT(-3)
30 PRINT FIX(3.58),FIX(-3.58)
40 PRINT INT(3.58),INT(-3.58)
```

```
RUN
3      -3
3      -3
3      -3
3      -4
```

F

FN

(function)

Gives the result obtained by the user defined function.

Format

FN function name [(X,Y,Z,...)]

X,Y,Z... : Numeric or string-constant, variable, array variable or expressions including these

Application

Specification method

Specify the function name to be evaluated. The function should have been defined in the DEF FN statement. Specify the values in X,Y,Z... corresponding to the parameters in the DEF FN statement. The values will be transferred to the defined function and the result will be obtained as the function value.

Example

```
10 A=3:B=4:C=5
20 DEF FNY(X)=A*X^2+B*X+C
30 PRINT FNY(3)
RUN
44
```

When a function which has not been defined is specified

Error(148) : Undefined user function

will be displayed.

FRE (free)

Gives the number of bytes of the unused area in the usable Sony disk BASIC memory.

Format

FRE [(X)]

X: Numeric or string- constant, variable, array variable, or expressions including these

Application

Specification method

To obtain the unused area with the FRE function after garbage collection¹⁾, specify a null string (" ") for X. If garbage collection is not required, specify an arbitrary numeric or string data (except a null string) for X.

```
10 A$="AAAAAAAAAA"  
20 A$=" "  
30 PRINT FRE(0)  
40 PRINT FRE(" ")  
RUN  
30617  
30627
```

If the specification of X is omitted

The number of bytes of the unused memory area is given without garbage collection.

¹⁾ For Sony disk BASIC, old string data remains in the memory if a string variable value is altered during program execution.

If arbitrary data X is given to the FRE function, the memory area containing the old data will be regarded as a used area; the number of bytes of the other area will be given.

If a null string is given as data X, all the old data in the memory will be cleared (garbage collection) and then the number of bytes of the unused area will be given.

Note

Garbage collection is performed when FRE (" ") is used or the program uses the entire memory area.

GPOINT

(graphic point)

The color code of the specified point on the graphic display screen is given.

Format

GPOINT (coordinates)

$-16384 \leq \text{X-coordinate} \leq 16383$	}	Numeric-constant, variable, array variable, or expressions including these
$-16384 \leq \text{Y-coordinate} \leq 16383$		

Application

Specification method

Specify coordinates on the graphic display screen.

Example

If

```
PRINT GPOINT(100,100)
```

is executed when a red dot is marked at point (100,100) on the screen,
4 will be given.

Value given by GPOINT function

An integer between 0 to 15 is given.

When the point outside of the displayed screen is specified, the value
-1 is given.

Note

See the Reference Chart for the color codes and colors.

GPOSX GPOSY

(graphic position X)

(graphic position Y)

Gives the coordinates specified in the last graphic command.

Format

GPOSX

GPOSY

Application

The GPOSX gives the last x-coordinate and the GPOSY gives the last y-coordinate in the preceding graphic command or the current point of the DRAW statement.

If there has been no graphic command or a DRAW statement, 0 will be given for each function's value.

Execution example

When

```
PRINT GPOSX,GPOSY
```

is executed after execution of

```
BOXF (0,0)-(50,70),3
```

the values 50 and 70 will be given.

when

```
DRAW "Z0m100,50S1C3U10L20D30"
```

is executed, then

```
PRINT GPOSX,GPOSY
```

is executed, 80 and 70 will be given.

Value given by the GPOSX or GPOSY function

An integer between -16384 to +16383 is given.

HEX\$

(hexadecimal dollar)

Representation of numeric data X is converted from decimal to hexadecimal notation, and the converted data is given as N-digit string data.

Format

HEX\$ (X [, N])

X: $-32768 \leq X \leq 32767$

N: $0 \leq N \leq 4$

} Numeric- constant, variable, array variable, or expressions including these

Application

Specification method

X is a decimal notation to be converted to a hexadecimal notation. N indicates the number of last digits (1 to 4) to be displayed. If 0 is specified for N, 0's in the higher-order digits will be omitted.

Example

```
PRINT HEX$(3500,4)
0DAC
```

```
PRINT HEX$(3500,3)
DAC
```

```
PRINT HEX$(3500,2)
AC
```

```
PRINT HEX$(3500,1)
C
```

```
PRINT HEX$(3500,0)
DAC
```

If the specification is omitted

0 is assumed to have been specified for N.

```
PRINT HEX$(16)
10
```

If specification is not an integer

The value will be rounded to the nearest whole number.

The value given is the same when $N = 0$ and $N = 2$ except when a key is kept pressed. The computer checks the B strobe (a flag reset when the data has been read) when $N = 0$ and the key data is given and then null strings are given until the B strobe is set again. When $N = 2$ and a key is kept pressed, the computer checks the A strobe (a flag set while a key is pressed) and the key data are read.

When N = 0 or is omitted, the data of the key pressed last is given.
If no key has been pressed, a null string is given.
When N = 1, the computer waits until a key is pressed and then gives the data of the pressed key.
When N = 2, the data of the key currently pressed is given. If no key is pressed when the INKEY\$ statement is executed, a null string is given.
If the specification is not an integer, it will be rounded to the nearest whole number and the value will be given.

If a key other than SHIFT, CTRL, and CTRL + P is pressed, the INKEY\$ function gives the character data corresponding to the pressed key.

INKEY\$(N))
0 ≤ N ≤ 2
Numeric-constant, variable, array variable,
or expressions including these

INKEYS
(inkey dollar)

Gives a string data from the keyboard.

- When a key is pressed, the key code is entered and the character corresponding to the key code is given. When a function key or a cursor key is pressed, the code shown below is entered.

Key	Code		Key	Code	
	Decimal	Hexadecimal		Decimal	Hexadecimal
F1	129	81	SHIFT+F1	149	95
F2	130	82	SHIFT+F2	152	98
F3	132	84	SHIFT+F3	146	92
F4	134	86	SHIFT+F4	133	85
F5	139	8B	SHIFT+F5	131	83
INS	143	8F	SHIFT+INS	144	90
DEL	145	91	SHIFT+DEL	147	93
CLR	142	8E	SHIFT+CLR	141	8D
HOME	148	94	SHIFT+HOME	140	8C
←	150	96	SHIFT+←	135	87
↑	151	97	SHIFT+↑	136	88
→	153	99	SHIFT+→	137	89
↓	156	9C	SHIFT+↓	138	8A
H	157	9D	SHIFT+H	157	9D

For other keys and their corresponding codes, see Appendix 9 in Part 1.

Program example

The program waits until any key is pressed. When a key is pressed, it displays the character as the INKEY\$ function value:

```

10 PRINT "PRESS ANY KEY"
20 X$=INKEY$(1)
30 PRINT X$;
40 GOTO 20
RUN
PRESS ANY KEY
ASDFG

```

INP (input)

Data is input from the specified I/O port.

Format

INP (N)

$0 \leq N \leq 255$: Numeric-constant, variable, array variable, or expressions including these

Application

This command is used to input data from the specified I/O port of the SMC-70.

See Appendix 7 in Part 1 for I/O port assignment. Refer to the Hardware Reference Manual for further details.

Note

An OUT command is used to output data to the specified I/O port.



INSTR (in string)

The position of character data Y\$ found after the Nth character of character data X\$ is given.

Format

INSTR ([N,] X\$, Y\$)

$0 \leq N \leq 255$: Numeric-constant, variable, array variable, or expressions including these

X\$ and Y\$: String-constant, variable, array variable, or expressions including these

Application

Example

When

```
X=INSTR(3,"WHAT IS THIS?","IS")
```

is specified, "IS" is found after the third character A of character data "WHAT IS THIS?". Since "IS" is the sixth and seventh characters, numeric data 6 is given as the function value.

This specification may be written as follows :

```
X$="WHAT IS THIS?"
Y$=" IS"
PRINT INSTR(3,X$,Y$)
6
```

Value given by an INSTR function

As the maximum character string length is 255, the given value is in the range of 0 to 255 (integer).

If there is no Y\$ after the Nth character (example 1), if the value of N is greater than the length of character data X\$ (example 2), or if X\$ is a null string (example 3), the INSTR function value is 0.

```
X$=" JAPANUSACANADA"
Y$=" JAPAN"
PRINT INSTR(6,X$,Y$) } Example 1
0
PRINT INSTR(20,X$,Y$) } Example 2
0
PRINT INSTR(1," ",Y$) } Example 3
0
```


If Y\$ is a null string, the INSTR function value will be N when the value of N is less than the length of character data X\$, and it will be 0 when the value of N is greater than the length of X\$.

```
PRINT INSTR(2,"JAPAN","")  
2
```

```
PRINT INSTR(100,"JAPAN","")  
0
```

If N is not an integer

This value will be rounded to the nearest whole number.

If N is 0 or N is omitted

Character data will be searched from the leftmost character of character data X\$.

```
X$="IS THIS A PEN?"
```

```
Y$="IS"
```

```
PRINT INSTR(0,X$,Y$)  
1
```

```
PRINT INSTR(1,X$,Y$)  
1
```

```
PRINT INSTR(X$,Y$)  
1
```



INT

(integer)

The maximum integer value under numeric data X is given.

Format

INT (X)

X: Numeric- constant, variable, array variable, or expressions including these

Application

Example

```
PRINT INT(3), INT(-3)
3      -3
```

```
PRINT INT(3.58), INT(-3.58)
3      -4
```

Difference between INT (X) and FIX (X)

When $X \geq 0$, $\text{INT}(X) = \text{FIX}(X)$

When $X < 0$, $\text{INT}(X) = \begin{cases} \text{FIX}(X) & (\text{Figures after decimal point are 0's.}) \\ \text{FIX}(X) - 1 & (\text{Others}) \end{cases}$

```
10 PRINT INT(3), INT(-3)
20 PRINT FIX(3), FIX(-3)
30 PRINT INT(3.58), INT(-3.58)
40 PRINT FIX(3.58), FIX(-3.58)
RUN
3      -3
3      -3
3      -4
3      -3
```

IRND (integer random)

Gives a random integer.

Format

IRND [(N)]

N : Numeric constant, variable, array variable, or expressions including these

Application

When N is specified, a random integer between 0 and N is given.

If N is not an integer, N will be rounded to the nearest whole number and then the function value will be given.

If N is from 32768 to 65535, N is regarded as a negative number (-32768 to -1; refer to Chapter 1 in Part 1) and the IRND function gives random integer between -32768 and N or between 0 and 32767. IRND function gives a random integer from a sequence of 65535 numbers. Using of IRND(0) gives the first number of a random sequence.

If the specification of N is omitted,

65535 (&HFFFF) is assumed to have been specified.

LEFT\$ (left dollar)

The left N characters of string data X\$ are given as string data.

Format

LEFT\$ (X\$, N)

X\$: String-constant, variable, array variable, or expressions including these

$0 \leq N \leq 255$: Numeric-constant, variable, array variable, or expressions including these

Application

Example

If

```
L$=LEFT$("123-4567",3)
```

is specified, the left three characters 123 of string data 123-4567 will be given as string data. Accordingly, if

```
PRINT L$
```

is executed,

```
123
```

will be displayed.

Value given according to N

If N is not an integer, N will be rounded to the nearest whole number and then the function value will be given.

If N is 0, a null string will be given.

```
PRINT LEFT$("ABC",0)
```

(Null string)

If N is greater than the length of string data X\$, the function value will be X\$.

```
PRINT LEFT$("ABC",10)
ABC
```

LEN (length)

The length (number of characters) of string data X\$ is given as numeric data.

Format

LEN (X\$)

X\$: String-constant, variable, array variable, or expressions including these

Application

Example

If

```
PRINT LEN("CHRISTMAS")
```

is specified, 9 will be displayed because character data CHRISTMAS consists of nine characters.

Value given by LEN function

When character contains spaces or a CHR\$ function, the space or CHR\$ function is counted as one character.

```
PRINT LEN("MERRY CHRISTMAS")
```

15 (The space is counted as one character.)

```
PRINT LEN(STR$(1.58))
```

4 (STR\$(1.58) gives character data 1.58 and its length is 4 because the decimal point is counted as one character.)

```
PRINT LEN(CHR$(7))
```

1 (The CHR\$ function always gives string data of one character.)

If character data is a null string

The LEN function value will be 0.

```
PRINT LEN(" ")
```

0

L

LN

(natural logarithm)

The value of natural logarithm $\log_e X$ is given.

Format

LN (X)

X > 0: Numeric-constant, variable, array variable, or expressions including these

Application

The LN function gives the value of the natural logarithm whose base is e (= 2.718281828459).

An internal operation is performed using floating-point representation.

Example

```
PRINT LN(10)
2.302585092994
```

```
10 A=14
20 B=EXP(A)
30 C=LN(B)
40 PRINT A,B,C
RUN
14      1202604.284165    14
```

Value of logarithm $\log_a b$ ($b > 0$) whose base is positive number a ($\neq 1$)

This value can be obtained from $\text{LN}(b)/\text{LN}(a)$.

LOG

(common logarithm)

The value of common logarithm $\log_{10}X$ is given.

Format

LOG (X)

X > 0: Numeric-constant, variable, array variable, or expressions including these

Application

The LOG function gives the value of the common logarithm whose base is 10. An internal operation is performed using a floating-point representation.

Example

```
PRINT LOG(90)
1.9542425094393
```

```
10 A=14
20 B=10^A
30 C=LOG(B)
40 PRINT A,B,C
RUN
14          1.E+14    14
```

Value of $\log_a b$ ($b > 0$) whose base is positive number a ($\neq 1$)

This value can be obtained from $\text{LOG}(b)/\text{LOG}(a)$.

L

MID\$ (middle dollar)

N characters beginning with the Mth character from the left of string data X\$ are given as string data.

Format

MID\$ (X\$, M, N)

X\$: String-constant, variable, array variable, or expressions including these

$1 \leq M \leq 255$

$0 \leq N \leq 255$

Numeric-constant, variable, array variable, or expressions including these

Application

Example

When

M\$=MID\$(" JAPANUSACANADA" , 6 , 3)

is specified, three characters (USA) beginning with U, the sixth character of JAPANUSACANADA, are given as string data.

Accordingly, if

PRINT M\$

is executed,

USA

will be displayed.

Value given by the MID\$ function

If M or N is not an integer, M or N will be rounded to the nearest whole number, and the value will be given.

If the number of characters on the right of the Mth character is less than N, all the characters on the right of the Mth character will be given.

PRINT MID\$(" JAPANESE" , 6 , 10)
ESE

If M is greater than the length of string data X\$ (example 1) or N is 0 (example 2), the function value will be a null string.

X\$= " JAPANUSA"

PRINT MID\$(X\$, 12 , 5)

(Null string)

} Example 1

PRINT MID\$(X\$, 6 , 0)

(Null string)

} Example 2

PEEK

(peek)

The data written at the specified memory address is given.

Format

PEEK (N)

$0 \leq N \leq 65535$: Numeric-constant, variable, array variable, or expressions including these

Application

Example

To read the data at address 2000, write the following :

```
PRINT PEEK(2000)
0
```

Value given by PEEK function

As the data is one byte, the value given is in the range of 0 to 255 (integer)

Note

A POKE command is used to write data at the desired memory address.

Program example

The following program displays the contents of the specified memory area in hexadecimal notation :

```

10 WIPE
20 CONSOLE 80
30 PRINT TAB(34);"MEMORY DUMP"
40 CCOLOR 1,1:PRINT TAB(24);"Hexadecimal";
  TAB(66);"Ascii";TAB(76);
50 /
60 CONSOLE ,20:CCOLOR 1,0:CCLEAR 20
70 INPUT "from,to [hex] :","M1$,M2$
80 CURSOR OFF
90 CONSOLE ,3,16:CCLEAR 2,17:LOCATE (0,2):CCOLOR 2
100 M1=VAL("&H"+M1$):IF M1<0 THEN M1=M1+65536.
110 M2=VAL("&H"+M2$):IF M2<0 THEN M2=M2+65536.
120 /
130 PRINT :PRINT HEX$(M1,4);" ";
140 FOR I=0 TO 15
150   PRINT HEX$(PEEK(M1+I),2);" ";
160 NEXT I
170 PRINT TAB(60);
180 FOR I=0 TO 15
190   A=PEEK(M1+I)
200   IF (A AND &H7F) < &H20 THEN PRINT "-";
  ELSE PRINT CHR$(A);
210 NEXT I
220 M1=M1+16:IF M1<=M2 THEN 130 ELSE 60
RUN

```

		MEMORY DUMP																	
		Hexadecimal																Ascii	
01A0	81 00 02 02 02 03 81 00 FF 81 00 02 02 02 03 81	-----																	
01B0	00 FF 81 00 02 02 02 03 81 00 FF 81 00 02 02 02	-----																	
01C0	03 81 00 FF 81 00 02 02 02 03 81 00 FF 81 00 02	-----																	
01D0	02 02 03 81 00 FF 81 00 02 02 02 03 81 00 FF 81	-----																	
01E0	00 02 02 02 03 81 00 FF 81 00 02 02 02 03 81 00	-----																	
01F0	FF 81 00 02 02 02 03 81 00 FF 81 00 02 02 02 03	-----																	
0200	81 00 FF 81 00 02 02 03 81 00 FF 81 00 02 02	-----																	

from,to [hex] :1A0,200

PEN

(pen)

Gives the status of the light pen. When the light pen has been pressed to the screen, -1 (true) is returned. Otherwise 0 (false) is returned.

Format

PEN

Application

This function is used to determine if the value of the PENX and PENY functions are valid or not. That is, it is used to determine whether the given value -1 indicates the coordinate or that the light pen has not been pressed.



PENCLM

(pen column)

The horizontal 1 position of the light pen on the character display screen which has been read with a PEN READ statement is given.

Format

PENCLM

Application

Value given by PENCLM function

An integer between 0 to 39 be given when the 40 character display mode is selected and an integer between 0 to 79 will be given when the 80 character display mode is selected.

Note

- If the light pen is not connected, the switch at the light pen tip is not pressed, or the touched place is dark, the PENCLM function value will be -1.
- If the border area is touched with the light pen, meaningless data will be given if the border area is not black, or -1 will be given if the border area is black.

PENLIN

(pen line)

The vertical position of the light pen on the character display screen which has been read with a PEN READ statement is given.

Format

PENLIN

Application

Value given by PENLIN function

An integer between 0 to 24 will be given.

Note

- If the light pen is not connected, the switch at the light pen tip is not pressed, or the touched place is dark, the PENLIN function value will be -1.
- If the border area is touched with the light pen, meaningless data will be given if the border area is not black, or -1 will be given if the border area is black.



PENX

(pen X)

The horizontal position of the light pen on the graphic display screen read with a PEN READ statement is given.

Format

PENX

Application

Value given by PENX function

The given value is the X-coordinate on the graphic display screen and it differs according to the resolution of the graphic display screen. For example, the following value is given on the initial coordinate system.

Low resolution mode : an integer from 0 to 159 incremented by 1

Standard resolution mode : an integer from 0 to 319 incremented by 2

High resolution mode : an integer from 0 to 639 incremented by 4

If a light pen is not connected, if the switch at the light pen tip is not pressed, or if the touched place is dark, the PENX function value will be -1.

If the border area is touched with the light pen, meaningless data will be given when the border area is not black, and -1 will be given when the border area is black.

Note

- Use of a light pen in the super high resolution mode is not guaranteed.
- A light pen may not be used when a CRT display using a fluorescent substance with a long afterglow is connected.

PENY

(pen Y)

The vertical position of the light pen on the graphic display screen read with a PEN READ statement is given.

Format

PENY

Application

Value given by a PENY function

The given value is the Y-coordinate on the graphic display screen and it differs according to the resolution of the graphic display screen. For example, the following value is given on the initial coordinate system.

Low resolution mode: an integer from 0 to 99 incremented by 1

Standard and high resolution mode:

an integer from 0 to 198 incremented by 2

If the light pen is not connected, the switch at the light pen tip is not pressed, or the touched place is dark, the PENY function value will be -1.

If the border area is touched with the light pen, meaningless data will be given when the border area is not black, and -1 will be given when the border area is black.

Note

- Use of a light pen in the super high resolution mode is not guaranteed.
- A light pen may not be used when a CRT display using a fluorescent substance with a long afterglow is connected.

Program example

The following program displays the vertical position touched with the light pen:

```
10 GCLEAR 13
20 PEN READ,1
30 PRINT PENY
40 GOTO 20
```

PI (pi)

Pi ($\pi = 3.1415926535898$) is indicated with a numeric constant.

Format

PI

Application

```
PRINT SIN(PI*3/2)  
-1
```

P

PROG (program)

Evaluates all the expressions listed from left to right and gives the result of the right-most expression.

Format

PROG (expression, expression,.....)

expression : relational, logical or arithmetic expression of numeric type or string expression.

Application

This function is used to evaluate several expressions in a statement. The value given by this function will be the numeric or the string type depending on the type of the last evaluated expression.

Example

```
10 X=10
20 Y=2
30 EVAL PROG(SETQ(A,X+Y),SETQ(B,X-Y),
  SETQ(C,X*Y), SETQ(D,X/Y))
40 PRINT "X+Y=";A
50 PRINT "X-Y=";B
60 PRINT "X*Y=";C
70 PRINT "X/Y=";D
```

The line 30 evaluates the addition, subtraction, multiplication and division in one statement.

A complicated evaluation can be written in one statement using this function.



The following program is to move the first character of A\$ to the last.

```
10 DEF FNRL$(A$)=COND(A$="",ERR(3),  
    PROG(SETQ(A$,A$+LEFT$(A$,1)),RIGHT$(  
        A$,LEN(A$)-1)))  
20 INPUT A$  
30 PRINT FNRL$(A$)  
40 PRINT B$
```

```
RUN  
?ABCD  
BCDA
```

For example, when "ABCD" is entered as the value of A\$, the FNRL\$ gives "BCDA". When a null string is entered,

```
Error(3) : Illegal function call
```

will be displayed by the ERR (3) function.

REC (record)

Gives the record number following that of the record accessed by the last PUT or GET statement.

Format

REC [(file number)]

$0 \leq \text{file number} \leq 127$

Numeric-constant, variable, array variable,
or expressions including these

Application

Specify the file number which has been opened with an OPEN statement.

This function is used for a random access file and the record number following that of the record accessed by the last PUT or GET statement is given.

Value given by REC function

An integer between 0 and 65535 is obtained.

When REC is used right after the file is opened, 0 will be given.

When REC is used for a file which has not been opened,

Error(155) : File not open

will be displayed.

When the specification of the file number is omitted

0 is assumed to have been specified.

REPLACES

(replace dollar)

N characters beginning with the Mth character from the left of the X\$ will be replaced by Y\$ and the result will be given.

Format

REPLACES\$(X\$,M,N,Y\$)

X\$,Y\$: String-constant, variable, array variable, or expressions including these.

$1 \leq M \leq 255$

$0 \leq N \leq 255$

Numeric-constant, variable, array variable, or expressions including these

Application

Example

When

```
10 A$="ABCDEF"
20 B$="XYZ"
30 C$=REPLACE$(A$,2,4,B$)
40 PRINT C$
```

is executed, 4 characters beginning with the 2nd character, that is, string data of "BCDE" will be replaced by "XYZ". Therefore,

AXYZF

will be displayed.

Value given by the REPLACES function

If M or N is not an integer, it will be rounded to the nearest whole number and the value will be given.

If N=0, the string data Y\$ will be inserted in front of the Mth character.

```
PRINT REPLACE$("ABC",2,0,"XYZ")
AXYZBC
```

If Y\$ is a null string, N characters beginning with the Mth character will be deleted.

```
PRINT REPLACE$("ABCDEF",2,2,"")
ADEF
```

If X\$ is a null string or if M is greater than the length of X\$,

```
Error(3) : Illegal function call
```

will be displayed.

```
PRINT REPLACE$("",2,2,"XYZ")
```

```
Error(3) : Illegal function call
```

```
PRINT REPLACE$("ABCDE",10,2,"XYZ")
```

```
Error(3) : Illegal function call
```

RIGHT\$ (right dollar)

The last N characters of string data X\$ are given as string data.

Format

RIGHT\$ (X\$, N)

X\$: String-constant, variable, array variable, or expression including these

$0 \leq N \leq 255$: Numeric-constant, variable, array variable, or expressions including these

Application

Example

When

```
R$=RIGHT$("I LOVE NEW YORK",8)
```

is specified, the last eight characters (NEW YORK) of string data "I LOVE NEW YORK " are given as character data.

Accordingly, if

```
PRINT R$
```

is executed,

```
NEW YORK
```

will be executed.

Value given by RIGHT\$ according to N

If N is not an integer, N will be rounded to the nearest whole number and then the function value will be given.

If N is 0, a null string is given.

```
PRINT RIGHT$("I LOVE NEW YORK",0)  
(Null string)
```

If N is greater than the length of string data X\$, the function value will be X\$.

```
PRINT RIGHT$("I LOVE NEW YORK",30)  
I LOVE NEW YORK
```

RND (random)

Random positive numbers which are less than 1 are generated.
(including 0)

Format

RND [(X)]

X : Numeric-constant, variable, array variable, or expressions including these

Application

Specification method

The random numbers to be obtained by the RND function depending on X are as follows :

When X is greater than 0 or omitted

Random numbers are generated in the same sequence.

When X is negative

The initial value of the sequence corresponding to the value X is generated.

When X is 0

The same value as the last generated value is given.

Example

If either of the following programs is executed :

```
10 FOR N=1 TO 5
20 PRINT RND(5)
30 NEXT N
```

or

```
10 FOR N=1 TO 5
20 PRINT RND(N)
30 NEXT N
```

the result will be as follows :

```
.93631576461867
.83096996030588
.27666054804669
.4005443924171
.56569845015311
```

This result will never change if this program is executed many times.

If the following program is executed :

```
5 PRINT RND(-1)
10 FOR N=1 TO 5
20 PRINT RND(N)
30 NEXT N
```

the result will be as follows :

```
.40615871059364
.14861598197605
.76657390864406
.53053579229367
.49584616598088
.85696141172169
```

In this case, random numbers are generated in the sequence corresponding to $X = -1$.

If the value of X of the RND function in line 5 is changed to another negative number, random numbers will be generated in another sequence. Negative number X corresponds to a random number sequence.

If the following program is executed :

```
10 PRINT RND
20 PRINT RND
30 PRINT RND(0)
```

the result will be as follows :

```
.93631576461867
.83096996030588
.83096996030588
```

If X is 0, the same random numbers as the previous ones will be generated.

To generate random numbers in different sequence every time the program is executed

Use a RANDOMIZE statement.

Note

There is an IRND function to generate an integer type random number.

SETQ

(set quote)

Evaluates the expression and assigns the result to the variable. At the same time, the result of the expression is given as the function value.

Format

SETQ (variable, expression)

expression : relational, logical or arithmetic expression of the numeric type or string type expression

Application

Operation of a SETQ command is the same as a LET command. This command is used to assign a value to a variable in an expression, for example, in a DEF FN statement.

The type of the variable and the type of the result of the expression must match.

The given value type will be the same as the type of the variable.

Example

```
10 A=SETQ(B,10)
20 C#=SETQ(D#,"ABC")
30 PRINT A,B,C#,D#
RUN
10      10      ABC      ABC
```

The next program is to draw a graph of a function $f(X) = \frac{1}{10} X^2$.

```
10 DEF FNCURVE(X)=PROG(SETQ(Y,(X^2)/10)
    ,SETQ(GX,X+180),SETQ(GY,140-Y),Y)
20 FOR I=-30 TO 30 STEP 1
30  PRINT I,FNCURVE(I)
40  GPLOT(GX,GY),4
50 NEXT
```

The DEF FN statement in line 10 defines the function and defines the coordinates GX and GY for plotting the value of X and f(X). When the FNCURVE function is evaluated, the values are assigned for GX and GY.

When the following program is executed and the FNADD3\$ is evaluated, the computer waits until a key is pressed, and the character corresponding to the pressed key is repeatedly added to the value of A\$ three times.

```
10 DEF FNADD3$(A$)=PROG(SETQ(B$,  
    INKEY$(1)),A$+STRING$(3,B$))  
20 A$="ABC"  
30 PRINT FNADD3$(A$)
```

For example, when **[Q]** key is pressed,

ABCQQQ

will be displayed. This is because "Q" is assigned to B\$, "QQQ" is given by STRING\$(3, B\$) and it is added to the value "ABC".

SGN (sign)

1 is given when numeric data X is positive ; 0 when X is 0 ; -1 when X is negative.

Format

SGN (X)

X : Numeric-constant, variable, array variable, or expressions including these

Application

Example

```
PRINT SGN(123)
```

```
1
```

```
PRINT SGN(0)
```

```
0
```

```
PRINT SGN(-123)
```

```
-1
```

Note

The SGN function is used when control branches according to the sign of numeric data X.

```
ON SGN(X)+2 GOTO 100,200,300
```

Control branches to line 100 if X is negative ; line 200 if X is 0 ; line 300 if X is positive.

SIN (sine)

The value of trigonometric function sine (X) of numeric data X is given.

Format

SIN (X)

X: Numeric-constant, variable, array variable, or expressions including these
(Unit: radian)

Application

Value given by SIN function

The value is floating-point numeric data in the following range

$$-1 \leq \text{SIN}(X) \leq 1$$

An internal operation is performed using a floating-point representation.

Example

```
PRINT SIN(PI/6)  
.500000000000001
```

When degree is used as the unit of angle

Sin (D°) can be obtained from $\text{SIN}(D * \text{PI}/180)$

```
PRINT SIN(30*PI/180)  
.500000000000001
```

Note

Mathematically, $\text{SIN } \pi/6 = \text{SIN } 30^\circ = 0.5$

In the above example, an error occurs because the 15th digit is rounded by the internal operation.

SPACE\$

(space dollar)

N spaces are given as a character data.

Format

SPACE\$(N)

$0 \leq N \leq 255$

Numeric-constant, variable, array variable
or expressions including these

Application

Example

When

```
S$=SPACE$(5)
```

is specified, 5 spaces are given as the value of character variable S\$.

When

```
10 PRINT SPACE$(10);STRING$(20,"*")
20 PRINT SPACE$(10);"*";SPACE$(18);"*"
30 PRINT SPACE$(10);"*"          SPACE$      "*"
40 PRINT SPACE$(10);"*";SPACE$(18);"*"
50 PRINT SPACE$(10);STRING$(20,"*")
```

is executed,

```
*****
*                                     *
*          SPACE$                   *
*                                     *
*****
```

will be displayed.

Value given by the SPACE\$ function

If N is not an integer, N will be rounded to the nearest whole number,
and the spaces will be given.

If N is 0, a null string is given.

```
PRINT LEN(SPACE$(5))
5
PRINT LEN(SPACE$(0))
0
```

SQR

(square root)

The square root of numeric data X ($\sqrt{X}$) is given.

Format

SQR (X)

$X \geq 0$: Numeric-constant, variable, array variable, or expressions including these

Application

An internal operation is performed using a floating-point representation.

Example

```
PRINT SQR(999)
31.606961258558
```

STR\$ (string dollar)

The string data indicating numeric data X is given.

Format

STR\$(X)

X: Numeric-constant, variable, array variable, or expressions including these

Application

Numeric data is converted to string data.

The function value is of the string type.

Example

```
PRINT STR$(5)
5
```

```
A$=STR$(30)
```

```
PRINT MID$(STR$(1234),1,3)
123
```

Sign of numeric data X

When numeric data X is negative, X is preceded by a minus sign (-); the first character of STR\$(X) is the minus sign. When numeric data X is 0 or more, the sign of X is omitted; the first character of STR\$(X) is a digit.

```
PRINT STR$(-123)
-123
```

```
PRINT STR$(123)
123
```

```
PRINT STR$(+123)
123
```

When X is floating-point data

The format of the character string given by the STR\$ function is the same as that of the PRINT statement.

STRING\$

(string dollar)

String data created by repeating X\$ for N times is given.

Format

STRING\$(N,X\$)

$0 \leq N \leq 255$

Numeric-constant, variable, array variable,
or expressions including these

X\$: String-constant, variable, array variable, or expressions including these

Application

Example

When

```
PRINT STRING$(5,"ABC")
```

is executed, string data "ABC" will be repeated 5 times and will be given as a string data. Accordingly,

```
ABCABCABCABCABC
```

will be displayed.

Value given by the STRING\$ function

If N is not an integer, N will be rounded to the nearest whole number, and the value will be given.

If the value of N is 0 or the value of X\$ is a null string, a null string will be given as the value of STRING\$.

Program example

```
10 INPUT "A,B,C=";A,B,C
20 PRINT "A";STRING$(A,"*")
30 PRINT "B";STRING$(B,"*")
40 PRINT "C";STRING$(C,"*")
RUN
A,B,C=?10,5,20
A*****
B*****
C*****
```


TAB (tab)

This function is used in the PRINT statement to move the cursor from the left margin X character(s) to the right.

Format

TAB (X)

$0 \leq X \leq 255$:

Numeric-constant, variable, array variable, or expressions including these

Application

The TAB function specified in the PRINT statement moves the cursor as follows (the current cursor is at the Cth character counted from the left margin).

- When X is greater than C, the cursor moves to Xth character from the left margin in the current line.
- If X is greater than the number of characters to be displayed in one line (40 or 80), the cursor will move to the next line.
- When X is less than C, the cursor moves to the Xth character in the next line.

Example

If

```
PRINT TAB(4); "A"
```

is executed, the following will be displayed :

```
      A
0 1 2 3 4 5 6 7 8 9 10
```

And if

```
PRINT TAB(4); "A"; TAB(7); "B"
```

is executed, the following will be displayed :

```
      A      B
0 1 2 3 4 5 6 7 8 9 10
```

T

If

```
PRINT TAB(4); "A", TAB(7); "B"
```

is executed, the following will be displayed :

```
      A      B
0  1  2  3  4  5  6  7  8  9  10
```

In this case, after A is displayed, the cursor moves to the 8th position because of the comma (,), and then the cursor moves to the 7th character in the next line to display B.

TAN (tangent)

The value of trigonometric function $\tan(X)$ of numeric data X is given.

Format

TAN (X)

X: Numeric-constant, variable, array variable, or expressions including these
(Unit: radian)

Application

Value given by TAN function

Internal operation is performed using a floating-point representation; floating-point numeric data is given.

Example

```
PRINT TAN(PI/4)
1
```

Value of TAN (PI/2)

If $X = \pi/2 \pm n\pi$ (n : integer), the value of $\tan X$ will be infinite.

For some X 's, the TAN function value overflows, and "Illegal function call" will be displayed.

When an angle is indicated in degrees

$\tan(D^\circ)$ can be obtained from $\text{TAN}(D * \text{PI}/180)$.

```
PRINT TAN(45*PI/180)
1
```

TIMES

(time dollar)

The current time (hour, minute, and second) can be given with the specified format.

Format

TIMES [(N)]

$0 \leq N \leq 4$: Integer type-constant, variable, array variable or expressions including these

Application

Specification method

The current hour (HH), minute (MM) and second (SS), or "AM" or "PM" will be given according to the value N as follows :

Value N	Data format
0	HHMMSS (in the 24-hour system)
1	HH : MM : SS (in the 24-hour system)
2	HH : MM : SS (in the 12-hour system)
3	AM or PM
4	0 (in the morning) or 1 (in the afternoon)

When the specification of N is omitted

0 is assumed to be specified.

Example

Assume that the current time is 3 : 15 : 15 p.m.

```
PRINT TIMES(0)
151515
PRINT TIMES(1)
15:15:15
PRINT TIMES(2)
03:15:15
PRINT TIMES(3)
PM
PRINT TIMES(4)
1
```

Note

There are SET DATE and SET TIME commands for setting the built-in timer.

VAL (value)

String data X\$ is given as numeric data.

Format

VAL (X\$)

X\$: String-constant, variable, array variable, or expressions including these

Application

String data is converted to numeric data.

The function value is of the numeric type.

Example

```
PRINT VAL("5")  
5
```

```
A=VAL("5")+3  
PRINT A  
8
```

String data X\$ that can be handled by VAL function

String data X\$ must begin with +, -, &H, space, or digit. If it begins with some other character or includes a character or space that cannot be handled as numeric data, an error will occur.

```
PRINT VAL("123")  
123  
PRINT VAL("+123")  
123  
PRINT VAL("-123")  
-123  
PRINT VAL(" 123")  
123  
PRINT VAL("12.34")  
12.34  
PRINT VAL("&HFF")  
255  
PRINT VAL("AB")  
Error(6) : Non numeric
```

VARPTR (variable pointer)

The lowest address of the memory storing the data assigned to variable X is given.

Format

VARPTR (X)

X : Numeric or string-variable or array variable.

Application

Value obtained by VARPTR function is in the range of

$$-32768 \leq \text{VARPTR}(X) \leq 32767 \text{ (integer)}$$

If the value given is positive, it will be used as the address.

If the value given is negative, the value 65536 will be added to it, and the result will be the address.

The VARPTR function can directly change the variable value if it is used together with the PEEK function or POKE statement.

Example

```
10 A%=5:B%=6
20 C%=A%+B%
30 M=VARPTR(C%)
40 PRINT HEX$(M,4),PEEK(M+1)
   *256+PEEK(M)
RUN
A1F7      11
```

When X is a variable to which data has not been assigned or an array variable not been declared

When a variable is specified in the VARPTR function, 0 will be written into the memory if the variable is of the numeric type ; a null string will be written if the variable is of the string type. This memory address at which the data is written is given as a function value.

Note

- If a statement is entered in the direct command mode, the variable data in the memory will be shifted to another location.
Therefore, to change the variable value directly with POKE, obtain the address of the variable with the VARPTR function and write the data at the address with POKE in a program.
- See Appendix 3 in Part 1 for the format of the data starting at the address obtained by the VARPTR function.

VINP

(video RAM input)

The data at the specified address of the video RAM is given.

Format

VINP (N)

$0 \leq N \leq 65535$: Numeric-constant, variable, array variable, or expressions including these

Application

This function is used to input data from the SMC-70 video RAM.

See Appendix 8 in Part 1 for video RAM addresses.

Refer to the Hardware Reference Manual for further details.

Example

When A is displayed in the upper-left corner of the screen and the following is executed :

```
PRINT VINP(&H0)
```

65 will be displayed.

When a red dot is displayed in the upper-left corner of the screen and the following is executed :

```
PRINT HEX$(VINP(&H8000))
```

40 will be displayed

Value given by VINP function

Since the value is 1-byte data, an integer from 0 to 255 is obtained.

$0 \leq \text{VINP}(N) \leq 255$

If an address other than those assigned to video RAM is specified

A meaningless value will be given.

Note

There is a VOUT command used to write data at the desired address of the video RAM.



Program example

The following program enlarges and displays the character pattern of a pressed key.

```
10 INPUT "Character :",CH$
20 PRINT
30 AD=&H1000+ASC(CH$)*8
40 FOR I=AD TO AD+7
50   T=VINP(I)
60   CP=&H80
70   FOR J=1 TO 8
80     IF (T AND CP) THEN PRINT "*";
        ELSE PRINT " ";
90     CP=CP\2
100  NEXT J
110  PRINT
120 NEXT I
130 PRINT :PRINT
140 GOTO 10
```

RUN

Character :A

```
  **
 *  *
*    *
*****
*    *
*    *
*    *
```

Character :\$

```
  *
*****
*  *
****
 *  *
*****
 *
```


CHAPTER 3

ERROR

MESSAGES

If an error occurs, program execution will stop and the prompt mode will be re-entered. Both the program and the variable values are retained. The error messages are displayed on the screen as follows :

Direct command mode

Error(130) : Syntax error

error number error message

Indirect command mode

`Error(130) in 20 : Syntax error`

error number number of line where the error occurred error message

1 Overflow

- The numeric data or operation result has exceeded the range that can be handled by Sony disk BASIC.

Numeric data that can be handled by Sony disk BASIC

Integer type: -32768 to 32767

Floating-point type: $-9.999999999999999E + 63$
to $9.999999999999999E + 63$

- An attempt was made to perform logical operation for values below -32768 or over 32767.
- A value 65530 or over was specified as a line number.

2 Division by zero

- An attempt has been made to divide by zero.

3 Illegal function call

- An attempt was made to obtain the function value of an illegal datum

Wrong

CHR\$(500)
HEX\$(25,5)

Correct

CHR\$(65)
HEX\$(25,4)

- An attempt was made to specify an illegal value in a command.

Wrong

Correct

```
CONSOLE 20.0,50  CONSOLE 40.0,24
CCLEAR 0,500     CCLEAR 0,25
```

4 Subscript out of range

- The specified array subscript was larger than the largest subscript declared in the DIM statement.

If it is necessary to change the size of the array, clear the array, then declare the new array with a DIM statement. If a CLEAR statement is executed, all array variables and other variables will be cleared. A RUN or NEW statement can also clear the array.

- A negative value was given to the array subscript.

5 Type mismatch

- An attempt was made to act upon data of different types.

Wrong

Correct

```
A="A"+2
```

```
A=A+2
```

- An attempt was made to perform logical operation for string data.
- The type of data specified in the function was illegal.

Wrong

Correct

```
VAL(3)
```

```
VAL("3")
```

6 Non numeric

- In a READ or INPUT statement, the data entered for a numeric variable was a nonnumeric character or a datum including spaces.
- The specification of the VAL function was not numeric character string.

Wrong

Correct

```
VAL("A")
```

```
VAL("3")
```

7 String too long

- The character string exceeds 255 characters.

Example

```
10 FOR I=1 TO 100
20 A$=A$+"ABCDE"
30 NEXT I
RUN
```

Error(7) : String too long

- The character string in the DEF KEY statement is too long.

8 Out of screen

- A value 16384 or over, or -16385 or less was specified as a coordinate in a graphic command (LINE, GPLOT, DRAW and so on) or a graphic function.
- A command or a function for character display, such as LOCATE, has specified a point not shown on the screen.

9 Out of DATA

- The number of data to be read with a READ statement was insufficient.
- A RESTORE statement was executed without a DATA statement.

10 File not found

- The specified file was not in the file device.
- The ESC key has been pressed before the specified file is found.

11 Check-sum error

- A program or data was not transferred from the file device.

12 File write error

- An error has occurred during data writing or program storing into a file on a disk.
- Defective or faulty disk or memory.

13 Bad file name

- An illegal file name has been specified in LOAD, SAVE, etc., statements.

14 File read error

- An error has occurred during data reading or program reading from a file on a disk.

16 Can't open file

- An attempt was made to open another new file on a disk where 128 files already exist.

17 Read past EOF

- An attempt was made to read data after the last data in the sequential file was read.

18 Out of disk space

- The disk space has been fully used.

129 Macro language error

- Graphic or music subcommands' syntax in the DRAW or PLAY statement does not conform with the Sony disk BASIC syntax.

130 Syntax error

- The entered statement does not conform with the Sony disk BASIC syntax.

131 Undefined variable

- The subroutine name has not been defined in a CALL statement.
- When an undefined variable was to be detected with the DEBUG command, the undefined variable has been used on the right side of the expression.

132 Undefined line number

- A nonexistent line number or undefined label was specified in a GOTO, GOSUB, EDIT, or DEBUG statement.
- A period (.) was used in an EDIT or LIST statement instead of a line number, but there was no line to be edited or listed.
- A GOTO, or GOSUB statement specified a nonexistent line and a RENUM statement was executed.

133 Duplicate definition

- The same variable names were used in two or more DIM or RECORD statements for array declaration.
- The same array variable of different dimensions was used. For example, A (5) and A (2, 3) were used in the program.
- OPTION BASE was altered.

If it is necessary to change the array size, dimension, or OPTION BASE, execute a CLEAR, RUN, or NEW statement, then execute a DIM or OPTION BASE statement.

When a CLEAR, RUN, or NEW statement is executed, all array variables are set in the undefined state, and all variables become 0's or a null string.

134 Out of memory

- The program is too long.
- Too many variables.
- An array which is too large was declared.
- The external loop contains too many internal loops.
- The function was called recursively too many times.
- The usable memory area specified in the MCLEAR statement is too small.

135 Line buffer overflow

- The buffer size of 255 bytes was exceeded when the entered program line was interpreted into the internal code.

Example

```
? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 :  
? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 :  
? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 :  
? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 :  
? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 : ? .1 :  
Error(135) : Line buffer overflow
```

- During data reading from a file, over 255 characters have been read as a datum.

136 FOR without NEXT

- There was no NEXT statement corresponding to the executed FOR statement.
- FOR-to-NEXT loops in a multiloop were improperly nested.
- The variable of the FOR statement did not match the variable of the corresponding NEXT statement.
- A FOR statement has been executed in the direct command mode without a corresponding NEXT statement.

Wrong

```
FOR I=1 TO 10  
Error(136) : FOR without NEXT
```

Correct

```
FOR I=1 TO 10:PRINT I;:NEXT  
12345678910
```

137 NEXT without FOR

- There was no FOR statement corresponding to the executed NEXT statement.
- Control was transferred within the FOR-to-NEXT loop with a GOTO statement.

138 RETURN without GOSUB

- There was no GOSUB statement corresponding to the executed RETURN statement.
- A RETURN statement was executed with no GOSUB statement in the program.
- There was no STOP or END statement to separate the main program from the subroutine program.

139 Can't CONTinue

- An attempt was made to restart a program with a CONT statement after the program was stopped with an END statement or by an error.
- A CONT statement was used when there was no program.
- A CONT statement was used to continue a program after the program was interrupted with a STOP statement or the **ESC** key and the program was modified.
- A LOAD, CONT statement was used in a program.

140 Invalid format

- A LOAD, CLOAD or OPEN statement was executed for a file which was not generated using the SMC-70.
- A LINK statement has been executed for a file which is not in the proper format.
- A datum with an invalid floating point format was assigned to a variable or an attempt was made to print out of a datum with an invalid floating point format.

141 Device not assigned

- An undefined device name was specified in SAVE, LOAD, etc. statement.

142 Internal error

- Defective or faulty memory or file contents.
- An undefined error has occurred.

143 WHILE without WEND

- There was no WEND statement corresponding to the executed WHILE statement.
- A WHILE statement has been executed in the direct command mode without a corresponding WEND statement.

144 WEND without WHILE

- There is no WHILE statement corresponding to the executed WEND statement.
- The control was transferred within the WHILE-to-WEND loop with a GOTO statement.

145 No RESUME

- An error has occurred in the error correction routine.

146 RESUME without error

- There was no ON ERROR statement corresponding to the executed RESUME statement.
- A RESUME statement has been executed without an error occurrence.

147 Can't MERGE non-ASCII file

- A MERGE or CHAIN/M statement has been executed specifying a program which was not saved in the ASCII format.

148 Undefined user function

- A function without DEF FN has been used.

149 Can't close file

- An error has occurred during closing a file.

150 Verify error

- The program content in memory was different from the program on the specified device.

151 Bad file access

- A data write or read statement such as PRINT # or INPUT # was executed for a file opened with an improper OPEN statement.

152 Function not available

- A command or a function which is not available in this Sony disk BASIC version was used.
- A DEBUG command was used before the file "DEBUGGER.PAC" has been loaded.

153 File already open

- The same file number is specified in different OPEN statements at the same time.

154 Bad file number

- A negative number or a number of 128 or over was specified as the file number in a statement such as OPEN.

155 File not open

- A PRINT #, INPUT #, SEND, RECEIVE, RECORD, GET, or PUT statement was executed before the OPEN statement has been executed.

156 Bad file device

- The specified file device cannot be used.

Example

An attempt was made to read data from the printer.

```
OPEN/R "LST:PROGRAM.ASC"
```

An attempt was made to write data on a ROM pack.

```
OPEN/G "PAC:DATA1.DAT"
```

157 No RECORD declaration

- A GET or PUT statement was executed when a RECORD statement has not been executed.

158 Record too long

- The total bytes allocated in the RECORD statement was over the record length specified in the OPEN statement.

Example

```
OPEN #1,"NAMELIST.DAT",128  
RECORD #1,JUST 50 AS A$(10)
```

159 File already exists

- The name of a file which was on the disk was specified as a new name in a RENAME statement.

160 Bad record number

- A record number specified in GET or PUT is a negative number, or over 65535.

Example

```
PUT #1,-1
```

161 Bad disk sector

- A defective, faulty or dirty floppydisk.
- The floppydisk drive did not operate normally.

162 Disk select error

- An attempt has been made to access the disk drive which is not handled by the CP/M.

Example

LOG IN "P:"

163 Disk R/O

- Data writing on the disk has been inhibited.
- The disk was replaced while using BASIC.

164 File R/O

- An attempt has been made to write data on the file onto which data writing has been inhibited with the STAT command of the CP/M.

