Citizen BASIC >
100 OPEN #1 : NAME
110 OPEN #2 : NAME
120 DO
130 INPUT #2 : A$
140 INPUT #2 : B$
150 PRINT #1 : "~$
160 PRINT #1 : M$
170 PRINT #1 : X$
180 PRINT #1 : T$
190 PRINT #1 : N$

Programming Manual
for
Citizen BASIC Interpreter

Citizen BASIC Interpreter

Contents

1. BASIC Interpreter Environment ............................................... 6
  1.1 Overview........................................................................... 6
  1.2 Start Basic Interpreter.................................................. 6
  1.3 BASIC Interpreter Mode................................................ 6
  1.4 End BASIC Interpreter.................................................. 7
  1.5 Example of BASIC Program.............................................. 7

2. BASIC Language ....................................................................... 9
  2.1 Syntactic Element........................................................... 9
    2.1.1 Basic Syntactic Element .......................................... 9
    2.1.2 Identifier .................................................................. 9
    2.1.3 Reserved Word ....................................................... 10
    2.1.4 Numeral .................................................................. 10
    2.1.5 Line Number ............................................................ 11
    2.1.6 String ..................................................................... 11
  2.2 Formula ............................................................................. 11
    2.2.1 Operator .................................................................. 11
    2.2.2 Arithmetic Operator.................................................. 12
    2.2.3 Logic Operator ......................................................... 12
    2.2.4 Logic Operator (Bit Operator) ..................................... 12
    2.2.5 String Operator ....................................................... 13
    2.2.6 Relational Operator ................................................. 13
    2.2.7 Priority of Operator .................................................. 13
    2.2.8 Function Call ........................................................... 14
  2.3 Statement ......................................................................... 14
    2.3.1 Assignment Statement .............................................. 14
    2.3.2 Function Call ........................................................... 14
    2.3.3 GOTO Statement ....................................................... 14
    2.3.4 Structured Statement ................................................. 15
    2.3.5 IF Statement ............................................................ 15
    2.3.6 Control Loop ......................................................... 16
    2.3.7 REPEAT Statement ................................................... 16
    2.3.8 WHILE Statement ..................................................... 16
    2.3.9 DO Statement .......................................................... 17
    2.3.10 FOR Statement ....................................................... 17
  2.4 Data Type and Variable ....................................................... 18
    2.4.1 Type .................................................................... 18
    2.4.2 Integer Type ............................................................ 18
    2.4.3 String Type ............................................................. 18
    2.4.4 Array Type ............................................................. 18
  2.5 Variable .......................................................................... 19
    2.5.1 Variable Declaration ................................................. 19
  2.6 Constant .......................................................................... 19
    2.6.1 Integer Type Constant ............................................... 19
    2.6.2 String Type Constant ............................................... 19
  2.7 Function .......................................................................... 19

3. Command ............................................................................. 19
  3.1 File Name ...................................................................... 20
    3.1.1 Program Store Drive ................................................. 20
    3.1.2 File Name Convention .............................................. 20
  3.2 Input-Output Device ......................................................... 20
    3.2.1 List of Devices ......................................................... 20
    3.2.2 Console Device ....................................................... 20
3.2.3 ZPL Device ................................................21
3.2.4 SER Device ................................................21
3.2.5 PAR Device ................................................21
3.2.6 USB Device ................................................21
3.3 List of Commands ..............................................21
3.4 List of Statements ............................................22
3.5 List of Functions .............................................23
3.6 Program Execution .............................................24
3.6.1 RUN Command ...............................................24
3.6.2 STEP Command ..............................................24
3.6.3 CONT and RESTART Commands .................................25
3.7 End Interpreter ...............................................25
3.7.1 BYE, QUIT, ~JQ, ZPL and SYSTEM Commands ...................25
3.8 Edit BASIC Program ............................................25
3.8.1 NEW Command ...............................................25
3.8.2 EDIT Command ..............................................25
3.8.3 AUTO and AUTONUM Commands ................................26
3.8.4 RENUM Command .............................................26
3.8.5 LIST Command ..............................................26
3.8.6 DELETE Command ............................................27
3.9 Save BASIC Program ............................................27
3.9.1 LOAD Command ..............................................27
3.9.2 SAVE and STORE Commands ...................................28
3.9.3 DIR and FILES Commands ....................................28
3.9.4 DELETE Command ............................................29
3.10 BASIC Program Execution Control Command .................29
3.10.1 END Statement ............................................29
3.10.2 GOTO Command ............................................29
3.10.3 STOP Statement ...........................................29
3.10.4 IF Statement .............................................30
3.10.5 Structured IF Statement ..................................30
3.10.6 FOR – NEXT Loop ..........................................31
3.10.7 REPEAT Statement – UNTIL Loop ............................31
3.10.8 WHILE – WEND Loop ........................................32
3.10.9 DO – LOOP Loop ...........................................32
3.10.10 EXIT Statement ..........................................33
3.10.11 GOSUB Statement .........................................34
3.10.12 RETURN Statement ........................................34
3.10.13 ON – GOTO Statement ..................................34
3.10.14 ON – ERROR – GOTO Statement .............................35
3.10.15 RESUME Statement ........................................36
3.10.16 SLEEP Statement .........................................36
3.11 Comment ......................................................36
3.11.1 REM Statement ............................................36
3.11.2 Comment in Line ..........................................37
3.12 Assignment Statement .........................................37
3.12.1 LET Assignment Statement .................................37
3.13 Data Input-Output ............................................37
3.13.1 PRINT Statement ..........................................37
3.13.2 INPUT Statement ..........................................39
3.13.3 READ Statement ..........................................39
3.13.4 DATA Statement ...........................................40
3.13.5 RESTORE Statement ........................................40
3.13.6 OPEN Statement ..........................................41
3.13.7 CLOSE Statement ...........................................42
1. BASIC Interpreter Environment

1.1 Overview

The BASIC interpreter can operate the BASIC language program under CLP/CL-S environment. Describing the BASIC language program independently, the user can print labels solely by CLP/CL-S. This enables communication with the host and peripheral equipment connected to CLP/CL-S, allowing them to be applied to various system environments.

1.2 Start Basic Interpreter

The BASIC interpreter becomes valid by sending the CLP/CL-S command from one of the communication ports, which becomes the console as the BASIC interpreter. In Windows, this is used by communicating with CLP/CL-S by such a program as hyper terminal.

To activate the BASIC interpreter, there are two types of the command.

(1) ~JI(ZPL)/[STX][ESC]I(DMX): With this command, the BASIC interpreter operates in the conversation mode. When this command is received, CLP/CL-S returns as the response the header and input prompt (>) including the program version.

(2) ^JI(ZPL)/[STX]L[ESC]I(DMX): This command directly starts operation of the BASIC language application program. The format of the parameters is as follows:

^JI command format: ^ JId:o.x,b,c,d

d: Drive of executive BASIC language application program
   Effective value E:, R: (ZPL) / A:, B: (DMX)

o: File name of executive BASIC language application program

x: File name extension of executive BASIC language application program
   ‘.BAS’ fixed

b: Console control
   Y: with console control
   N: without console control
   Cannot be designated

c: Console echo control
   Y: with console echo control
   N: without console echo control
   Cannot be designated

d: BASIC application memory size
   Designate memory size allocated for BASIC application between 20K byte and 1024K byte.
   Default: 50K byte
   Designating example: 32K

1.3 BASIC Interpreter Mode

For handling the BASIC interpreter, there are the following two modes:

(1) Conversation mode
The conversation mode is directly processed once a command is input. A command line input without a line number is immediately executed when received by the printer.

【E.g.】
>NEW

(2) Program mode
The program mode executes the command line stored in the memory in sequence of line numbers by inputting the RUN command.

【E.g.】
>10 A=10
>20 B=3
The program is executed from the command line of the initial line number by inputting RUN command. While the program is executing, the execution of the program can be interrupted by the Control-C (^C) command sent from the console to the printer. The interrupted program can be restarted from the interrupted position by inputting the RESTART command.

1.4 End BASIC Interpreter

The BASIC interpreter can end by either of the following two methods:
(1) Input ZPL or BYE command in prompt (>) state
(2) Input ~ JQ command in prompt (>) state

1.5 Example of BASIC Program

(1) Example 1

This is the program to print the weight measured by electric scale on the label.

```
1000 OPEN #1 : NAME "ZPL"
1010 OPEN #2 : NAME "SER"
1020 DO
1030 SLEEP 1
1040 PRINT #2 : "W"
1050 INPUT #2 : A$
1060 PRINT A$
1070 IF A$ = "EXIT" THEN
1080 CLOSE #1
1090 CLOSE #2
1100 END
1110 END IF
1120 PRINT A$
1130 LOOP WHILE POS(A$,"000.00")=1 OR POS(A$,"?")=1
1140 PRINT #1 : "^SD15^XA^MTD^FS^PW400^FS"
1150 PRINT #1 : ^LH0,0^FS
1160 PRINT #1 : ^FO56,47^A0N,69,58^FDThis weighs^FS";
1170 PRINT #1 : "^FO56,150^A0N,69,58^FD"&A$&" lbs^FS";
1180 PRINT #1 : "^FP1,0,0,N";
1190 PRINT #1 : "^XZ"
1200 GOTO 1020
```

(2) Example 2

The following example reads 5 strings from the console to print it on a printer label. Using 5 strings, the placement must be declared by DECLARE.

```
1000 OPEN #1 : NAME "ZPL"
1010 DECLARE STRING MYARRAYS(5)
1020 FOR INDEX=1 TO 5 STEP 1
1030 INPUT MYARRAYS(INDEX)
1040 NEXT INDEX
1050 PRINT #1 : "^XA^FO20,20^AON,69,58^FD"&MYARRAYS(1)&"^FS";
1060 PRINT #1 : "^FO20,80^AON,69,58^FD"&MYARRAYS(2)&"^FS";
1070 PRINT #1 : "^FO20,140^AON,69,58^FD"&MYARRAYS(3)&"^FS";
1080 PRINT #1 : "^FO20,200^AON,69,58^FD"&MYARRAYS(4)&"^FS";
1090 PRINT #1 : "^FO20,260^AON,69,58^FD"&MYARRAYS(5)&"^FS^XZ"
```

The program is executed from the command line of the initial line number by inputting RUN command. While the program is executing, the execution of the program can be interrupted by the Control-C (^C) command sent from the console to the printer. The interrupted program can be restarted from the interrupted position by inputting the RESTART command.

1.4 End BASIC Interpreter

The BASIC interpreter can end by either of the following two methods:
(1) Input ZPL or BYE command in prompt (>) state
(2) Input ~ JQ command in prompt (>) state

1.5 Example of BASIC Program

(1) Example 1

This is the program to print the weight measured by electric scale on the label.

```
1000 OPEN #1 : NAME "ZPL"
1010 OPEN #2 : NAME "SER"
1020 DO
1030 SLEEP 1
1040 PRINT #2 : "W"
1050 INPUT #2 : A$
1060 PRINT A$
1070 IF A$ = "EXIT" THEN
1080 CLOSE #1
1090 CLOSE #2
1100 END
1110 END IF
1120 PRINT A$
1130 LOOP WHILE POS(A$,"000.00")=1 OR POS(A$,"?")=1
1140 PRINT #1 : "^SD15^XA^MTD^FS^PW400^FS"
1150 PRINT #1 : ^LH0,0^FS
1160 PRINT #1 : ^FO56,47^A0N,69,58^FDThis weighs^FS";
1170 PRINT #1 : "^FO56,150^A0N,69,58^FD"&A$&" lbs^FS";
1180 PRINT #1 : "^FP1,0,0,N";
1190 PRINT #1 : "^XZ"
1200 GOTO 1020
```

(2) Example 2

The following example reads 5 strings from the console to print it on a printer label. Using 5 strings, the placement must be declared by DECLARE.

```
1000 OPEN #1 : NAME "ZPL"
1010 DECLARE STRING MYARRAYS(5)
1020 FOR INDEX=1 TO 5 STEP 1
1030 INPUT MYARRAYS(INDEX)
1040 NEXT INDEX
1050 PRINT #1 : "^XA^FO20,20^AON,69,58^FD"&MYARRAYS(1)&"^FS";
1060 PRINT #1 : "^FO20,80^AON,69,58^FD"&MYARRAYS(2)&"^FS";
1070 PRINT #1 : "^FO20,140^AON,69,58^FD"&MYARRAYS(3)&"^FS";
1080 PRINT #1 : "^FO20,200^AON,69,58^FD"&MYARRAYS(4)&"^FS";
1090 PRINT #1 : "^FO20,260^AON,69,58^FD"&MYARRAYS(5)&"^FS^XZ"
```
(3) Example 3

This example employs AUTOEXEC. It automatically activates the BASIC interpreter by AUTOEXEC.ZPL to execute the START.BAS program found in the E: drive.

```
^XA
^DFE:AUTOEXEC.ZPL^FS
^JIE:START.BAS,Y,N,50K^FS
^XZ
```

(4) Example 4

This example explains the BASIC program that extracts data. The BASIC interpreter is used by collecting data from the program not based on the CLP/CL-S command.

```
30 OPEN #2 : NAME "SER"
40 OPEN #1 : NAME "ZPL"
50 DO
60 INPUT #2 : A$
70 IF A$ = "EXIT" THEN
80 CLOSE #2
90 CLOSE #1
95 END
100 END IF
110 LOOP WHILE POS(A$,"START") = 0
120 DECLARE STRING BS$(17)
130 LET INDEX = 1
140 DO WHILE INDEX <= 17
150 INPUT #2 : B$
160 IF POS(B$,"DATA") <> 0 THEN
170 LET BS$(INDEX) = EXTRACT$(B$","",";")
180 LET INDEX = INDEX + 1
190 END IF
200 LOOP
210 PRINT #1 : "~SD15^XA^MTD^FS^PW400^FS";
220 PRINT #1 : "^LH0,0^FS";
230 PRINT #1 : "^FO20,20^AON,69,58^FD"&BS$(1)"^FS";
240 PRINT #1 : "^FO20,80^AON,69,58^FD"&BS$(2)"^FS";
250 PRINT #1 : "^FO20,140^AON,69,58^FD"&BS$(3)"^FS";
260 PRINT #1 : "^FO20,200^AON,69,58^FD"&BS$(4)"^FS";
270 PRINT #1 : "^FO20,260^AON,69,58^FD"&BS$(5)"^FS";
280 PRINT #1 : "^FO20,320^AON,69,58^FD"&BS$(6)"^FS";
290 PRINT #1 : "^FO20,380^AON,69,58^FD"&BS$(7)"^FS";
300 PRINT #1 : "^FO20,440^AON,69,58^FD"&BS$(8)"^FS";
310 PRINT #1 : "^FO20,500^AON,69,58^FD"&BS$(9)"^FS";
320 PRINT #1 : "^FO20,560^AON,69,58^FD"&BS$(10)"^FS";
330 PRINT #1 : "^FO20,620^AON,69,58^FD"&BS$(11)"^FS";
340 PRINT #1 : "^FO20,680^AON,69,58^FD"&BS$(12)"^FS";
350 PRINT #1 : "^FO20,740^AON,69,58^FD"&BS$(13)"^FS";
360 PRINT #1 : "^FO20,800^AON,69,58^FD"&BS$(14)"^FS";
370 PRINT #1 : "^FO20,860^AON,69,58^FD"&BS$(15)"^FS";
380 PRINT #1 : "^FO20,920^AON,69,58^FD"&BS$(16)"^FS";
390 PRINT #1 : "^FO20,980^AON,69,58^FD"&BS$(17)"^FS";
400 PRINT #1 : "^PQ1,0,0,N";
```
410 PRINT #1 : "^XZ";
420 GOTO 50

(5) Example 5

This example explains how a different port communicates by another channel. Note that the commands of INPUT and PRINT are used.

20 OPEN #1 : NAME "SER"
30 OPEN #2 : NAME "ZPL"
40 INPUT #1 : A$
50 PRINT #2 : "^XA^FO20,20^A0N,69,58^FD"&A$&"^FS^XZ"
60 CLOSE #1
70 CLOSE #2
80 END

Line 20 opens the serial port. Line 30 opens the printer engine. Line 40 receives data. In this example, channel 1 is the serial port. If no channel number is given, console will be the standard. Line 50 sends data to the printer engine designated by channel 2.

2. BASIC Language
2.1 Syntactic Element

The BASIC language uses a set of ASCII characters including alphabets from A to Z and a to z, numbers from 0 to 9, and other standard characters. Formulae, declarations and statements are constructed in combination with basic syntactic elements called token. A statement refers to processing on a feasible algorithm in a program. A formula is a syntactic unit constructing part of a statement, referring to a given value. A declaration defines identifiers such as function names and variable names. The identifiers can be used in formulae and statements. A declaration could allocate memory for an identifier as required.

2.1.1 Basic Syntactic Element

In the simplest view, a program consists of a series of tokens separated by the separator. A token is the smallest unit with meaning in the text of a program. Strictly speaking, however, there are some cases where the two tokens need not be separated by the separator. For instance, the following is a correct BASIC code:

1000 Size=20:Price=10

However, taking convention and readability into consideration, this needs to be described as follows:

1000 Size=20
1010 Price=10

Tokens are classified as special symbols, identifiers, reserved words, commands, numerals, line numbers and strings. The separator can be part of a token only when such token is a string. At least one separator is required between adjacent identifiers, reserved words, numerals and line numbers.

2.1.2 Identifier

The identifier refers to a variable or function. Although its length is arbitrary, only the first 16 characters have meaning. An identifier must begin with an alphabet or an underscore (_) with no space in it. For the second character onwards, alphabets,
2.1.3 Reserved Word

The following reserved words are not definable or applicable as an identifier:

<table>
<thead>
<tr>
<th>Reserved Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPEAT</td>
</tr>
<tr>
<td>REMLIST</td>
</tr>
<tr>
<td>EDITINPUT</td>
</tr>
<tr>
<td>NEXT</td>
</tr>
<tr>
<td>AUTO</td>
</tr>
<tr>
<td>OPEN</td>
</tr>
<tr>
<td>EXIT</td>
</tr>
<tr>
<td>REPEAT</td>
</tr>
<tr>
<td>RENUM</td>
</tr>
<tr>
<td>AUTONUM</td>
</tr>
<tr>
<td>OFF</td>
</tr>
<tr>
<td>STEP</td>
</tr>
<tr>
<td>ENDF</td>
</tr>
<tr>
<td>OUTBYTE</td>
</tr>
<tr>
<td>RIGHT</td>
</tr>
<tr>
<td>STR</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>LTRIM</td>
</tr>
<tr>
<td>INKEY</td>
</tr>
<tr>
<td>ASC</td>
</tr>
<tr>
<td>LOG</td>
</tr>
<tr>
<td>SIN</td>
</tr>
<tr>
<td>RND</td>
</tr>
<tr>
<td>ISERROR</td>
</tr>
<tr>
<td>MAXLEN</td>
</tr>
<tr>
<td>TIME</td>
</tr>
<tr>
<td>OUTPUT</td>
</tr>
<tr>
<td>STRING</td>
</tr>
<tr>
<td>INPUT</td>
</tr>
<tr>
<td>LET</td>
</tr>
<tr>
<td>CLEAR</td>
</tr>
<tr>
<td>GOSUB</td>
</tr>
<tr>
<td>READ</td>
</tr>
<tr>
<td>CLOSE</td>
</tr>
<tr>
<td>DEF</td>
</tr>
<tr>
<td>UNTIL</td>
</tr>
<tr>
<td>RECOVER</td>
</tr>
<tr>
<td>DIR</td>
</tr>
<tr>
<td>DECLARE</td>
</tr>
<tr>
<td>TRACE</td>
</tr>
<tr>
<td>SLEEP</td>
</tr>
<tr>
<td>JQ</td>
</tr>
<tr>
<td>LEFT</td>
</tr>
<tr>
<td>SPACE</td>
</tr>
<tr>
<td>JOC</td>
</tr>
<tr>
<td>STRING</td>
</tr>
<tr>
<td>ERM</td>
</tr>
<tr>
<td>EXP</td>
</tr>
<tr>
<td>INSTR</td>
</tr>
<tr>
<td>SGN</td>
</tr>
<tr>
<td>EOF</td>
</tr>
<tr>
<td>EVAL</td>
</tr>
<tr>
<td>LEN</td>
</tr>
<tr>
<td>ATAN</td>
</tr>
<tr>
<td>TIME</td>
</tr>
<tr>
<td>APPEND</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>PRINT</td>
</tr>
<tr>
<td>SAVE</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
<tr>
<td>DATA</td>
</tr>
<tr>
<td>MERGE</td>
</tr>
<tr>
<td>DIM</td>
</tr>
<tr>
<td>ON</td>
</tr>
<tr>
<td>QUIT</td>
</tr>
<tr>
<td>RESTORE</td>
</tr>
<tr>
<td>POKE</td>
</tr>
<tr>
<td>WEND</td>
</tr>
<tr>
<td>SYSTEM</td>
</tr>
<tr>
<td>POKE</td>
</tr>
<tr>
<td>INBYTE</td>
</tr>
<tr>
<td>SYSTEM</td>
</tr>
<tr>
<td>WIDTH</td>
</tr>
<tr>
<td>ECHO</td>
</tr>
<tr>
<td>DEBUG</td>
</tr>
<tr>
<td>RESTART</td>
</tr>
<tr>
<td>INBYTE</td>
</tr>
<tr>
<td>MID</td>
</tr>
<tr>
<td>CHR</td>
</tr>
<tr>
<td>MF</td>
</tr>
<tr>
<td>UCASE</td>
</tr>
<tr>
<td>OCT</td>
</tr>
<tr>
<td>VAL</td>
</tr>
<tr>
<td>SQRT</td>
</tr>
<tr>
<td>PEEK</td>
</tr>
<tr>
<td>MKF</td>
</tr>
<tr>
<td>MAX</td>
</tr>
<tr>
<td>DATE</td>
</tr>
<tr>
<td>MOD</td>
</tr>
<tr>
<td>TO</td>
</tr>
<tr>
<td>ACCESS</td>
</tr>
<tr>
<td>INOUT</td>
</tr>
</tbody>
</table>

2.1.4 Numeral

An integer constant and real constant are described as the numeral of decimal notation which consists of a series of numerals with no commas or space included, and are led by the operator of + or - representing a sign. A value turns positive by default, and thus 67258 is equivalent to +67258, for instance. Further, a constant value must be included in the range of the defined real type maximum value and integer type maximum value.

Numbers with the decimal point or exponential part represent real numbers, and the others refer to integer numbers. The real number with E or e is interpreted as “...th power of ~ multiplied by 10”. For instance, 7E-2 is interpreted as 7×10^-2, and both 12.25e+6 and 12.25e6 as 12.25×10^6.

However, the variables of the BASIC interpreter support integer numbers only; therefore real numbers cannot be substituted.

The prefix of an ampersand such as &H8F represents hexadecimal. A hexadecimal value
without the monadic operator – is handled as a positive value.

2.1.5 Line Number

In the BASIC program to be programmed, a line must begin with a line number, which can be chosen from the range between 0 and 99999999. The line number is used in goto statements and if statements.

2.1.6 String

The string is also called the string constant, consisting of a string with quotation marks. The string is a set of extended ASCII characters with the maximum number of characters of 255. It is described in quotation marks within a line. A quoted string without anything between quotation marks is a blank string. Its examples are:

"DOG"
"CAT"

Strings can be combined by &. For instance, assume there is a combined string shown below:

"HELLO"&"WORLD"

This corresponds to a string with quotation marks:

"HELLOORD"

The length of a string represents the number of characters in the string.

2.2 Formula

A formula is a syntactic element that returns a value. The following is some examples:

X Variable
15 Constant
InterestRate Variable
MAX(X,Y) Function call
X*Y Product of X and Y
Z/(1-Z) Quotient of Z and (1-Z)

The simplest formulae are the variable and constant. More complex formulae consist of simple formulae using operators, function call and subscription.

2.2.1 Operator

The operator performs as a defined function incorporated into the BASIC language. For instance, the formula (X+Y) is formed by adding the + operator to the variables X and Y called the operand. If X and Y stand for integral numbers, then (X+Y) returns its total. Operators include NOT, *, /, MOD, AND, +, -, OR, XOR, =, >, <, <> and >=.

The NOT operator, a monadic operator, takes one operand. All the other operands are binominal operators (taking two operands). However, + and - can perform as a monadic or binominal operator. Always put the monadic operator before the operand as -B. Put the binominal operator between operands as A=7.

Operation of some operators varies depending on the data type received. For example, NOT performs negation operation per bit for integer operands, and logical negation
operation for logic operands. The subsequent tables describe these operators in several classifications.

2.2.2 Arithmetic Operator

The arithmetic operator takes integer operands including +, -, *, / and MOD.

Table 1: Binomial arithmetic operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>Integer type</td>
<td>X+Y</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Integer type</td>
<td>Result-1</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Integer type</td>
<td>P*InterestRate</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Integer type</td>
<td>X/2</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>Residue</td>
<td>Integer type</td>
<td>Y MOD 6</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Monadic arithmetic operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Sign identity</td>
<td>Integer type</td>
<td>+7</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Sign negation</td>
<td>Integer type</td>
<td>-X</td>
<td></td>
</tr>
</tbody>
</table>

2.2.3 Logic Operator

Logic operators NOT, AND, OR and XOR take an arbitrary integer operand.

Table 3 Logic operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>Negation</td>
<td>Integer type</td>
<td>not(MySet=1)</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>Logic product</td>
<td>Integer type</td>
<td>(Total&lt;100)AND(Total&gt;0)</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Logic sum</td>
<td>Integer type</td>
<td>(A=1)OR(B=1)</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive logic sum</td>
<td>Integer type</td>
<td>(A=1)XOR(B=1)</td>
<td></td>
</tr>
</tbody>
</table>

These operations follow the typical Boolean logic.

2.2.4 Logic Operator (Bit Operator)

The following logic operators perform operation per bit for integer operands. For instance, assume that the binary expression of the value stored in X is 001101 while that in Y is 100001.

Z = X OR Y

This formula assigns the value 101101 to Z.

Table 4: Logic operator (bit operator)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
</table>
2.2.5 String Operator

All relational operators =, <>, <, >, <= and >= take string type operands. The operators + and & combine two strings.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, &amp;</td>
<td>Combination</td>
<td>String type</td>
<td>String type</td>
<td>S$&quot;.&quot; &amp; &quot;world&quot;</td>
</tr>
</tbody>
</table>

2.2.6 Relational Operator

The relational operator is used to compare two operands.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand</th>
<th>Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
<td>Integer and string types</td>
<td>Integer type</td>
<td>I=Max</td>
</tr>
<tr>
<td>&lt;</td>
<td>Smaller than</td>
<td>Integer and string types</td>
<td>Integer type</td>
<td>X&lt;Y</td>
</tr>
<tr>
<td>&gt;</td>
<td>Larger than</td>
<td>Integer and string types</td>
<td>Integer type</td>
<td>Len&gt;0</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Or less</td>
<td>Integer and string types</td>
<td>Integer type</td>
<td>Cnt&lt;=I</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Or more</td>
<td>Integer and string types</td>
<td>Integer type</td>
<td>I&gt;=1</td>
</tr>
</tbody>
</table>

Most simple types have no complicated rules regarding comparison. For example, I=J becomes true only when I and J share the same value. Otherwise, I<>J becomes true. To the relational operator, the following rules are applied:

- The operand must be the type with compatibility.
- The string is compared in accordance with the sequence value creating characters to construct a string.

2.2.7 Priority of Operator

In a complex formula, the execution sequence of operation in accordance with the rules for priority.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>1st (highest)</td>
</tr>
<tr>
<td>*, / , MOD</td>
<td>2nd</td>
</tr>
<tr>
<td>+, -</td>
<td>3rd</td>
</tr>
<tr>
<td>=, &lt;, &gt; , &lt; =, &gt; =</td>
<td>4th</td>
</tr>
<tr>
<td>AND, OR, XOR</td>
<td>5th (lowest)</td>
</tr>
</tbody>
</table>
Operators with higher priority are assessed before those with lower priority while those with equal priority are assessed from the left. Therefore in the formula \(X+Y \times Z\), \(Y\) is multiplied by \(Z\) to add \(X\) to its result. However, in the formula \(X-Y+Z\), \(Y\) is subtracted from \(X\) to add \(Z\) to its result. Since \(-\) and \(+\) are equal in their priority, the left operation is executed first.

Use of brackets allows the rules of priority to override. The formula in brackets is assessed first to be handled as the single operand subsequently. For instance, in the formula \((X+Y) \times Z\), the sum of \(X\) and \(Y\) is multiplied by \(Z\).

### 2.2.8 Function Call

Function call is a type of a formula since a function returns values. For example, when MAX returns an integer taking two integer arguments, the function call of \(\text{MAX}(24,47)\) is the formula of the integer type.

### 2.3 Statement

The statement defines algorithmic processing in a program. Combination of simple statements such as assignment and procedure call allows creating loops, conditional statements and other structured statements. To separate several statements in a line, use colons.

#### 2.3.1 Assignment Statement

An assignment statement has the following form:

\[
\text{Variable} = \text{formula}
\]

An assignment statement is replaced by the value of the current value of the variable. An example includes:

\[
I=3
\]

This assigns the value 3 to the variable \(I\). Variable reference in the left member of an assignment statement is available for the right-hand formula as shown below:

\[
I=I+1
\]

This increments the value of \(I\).

#### 2.3.2 Function Call

A function is structured by a function name and, if required, a parameter list. An instance is:

\[
\text{ISERROR} \\
\text{DATAREADY}(1) \\
\text{LEN}(\text{A}$)
\]

#### 2.3.3 GOTO Statement

A GOTO has the following form:

\[
\text{GOTO line number}
\]

A GOTO statement forwards program execution to the statement with the designated
2.3.4 Structured Statement

A structured statement, a statement constructed by other statements, is used to execute statements in order, with conditions or repeatedly.

- A conditional statement (IF statement) executes at most one statement structured based on designated criteria
- Repetition statements include the REPEAT loop, WHILE loop, DO loop and FOR loop, and repeatedly execute statements structuring those loops in order

2.3.5 IF Statement

IF statement has two types: IF...THEN...ELSE depicted in the single line and IF...ELSE...ENDIF depicted in several lines. The following shows syntax depicted in the single line IF...THEN...ELSE:

```
IF formula THEN statement 1 ELSE statement 2
```

The formula returns the logic value. When the formula is true, statement 1 is executed while statement 2 is executed in other cases. The ELSE clause can be omitted. An example is shown below:

```
IF J<>0 THEN Result=I/J
```

The syntax depicted in several lines IF...THEN...ELSE...ENDIF is shown below:

```
IF formula THEN
{statement 1}
ELSE
{statement 2}
ENDIF
```

The formula returns a logic value. When the formula is true, {statement 1} is executed while in other cases {statement 2} is executed as shown below:

```
IF J=0 THEN
Result=0
ELSE
Result=I/J
ENDIF
```

THEN and ELSE clauses include one or more statements respectively, for which structured statements are also used. The following is an example:

```
IF J<>0 THEN
Result=I/J
Count=Count+1
ELSE IF Count=Last THEN
```
2.3.6 Control Loop

Use of a loop repeatedly executes a series of statements to stop execution using control conditions or control variables. BASIC includes four types of control loops: REPEAT statement, WHILE statement, DO statement and FOR statement. Use of the EXIT procedure controls the flow of the REPEAT statement, WHILE statement, DO statement or FOR statement. EXIT ends execution of a statement including them.

2.3.7 REPEAT Statement

The following shows the syntax of the REPEAT statement:

```
REPEAT
{statement}
UNTIL formula
```

The formula returns a logic value. The REPEAT statement executes a series of statements constructing it to test the formula every time repetition ends. When the formula returns true, the REPEAT statement ends. Since the formula is not assessed until the first execution is completed, a series of statements will always be executed at least once.

An example of the REPEAT statement includes:

```
REPEAT
K=I MOD J
I=J
J=K
UNTIL J=0
REPEAT
PRINT"Enter a value(0..9):";
INPUT I
UNTIL (I>=0) AND (I<=9)
```

2.3.8 WHILE Statement

Although the WHILE statement is similar to the REPEAT statement, it is different in a sense that the control condition is assessed before a series of statements are executed for the first time. Therefore, if the condition is false, a series of statements will never be executed.

```
WHILE formula
{statement}
WEND
```

The formula returns a logic value. The WHILE statement executes a series of statements constructing it to test the formula before starting each repetition. Such execution continues as long as the formula returns true.

An example of the WHILE statement includes:

```
WHILE I<>X
I=I+1
WEND
```
2.3.9 DO Statement

The DO statement is similar to the REPEAT statement and the WHILE statement. The syntax of the DO statement is shown below:

Syntax 1
DO WHILE formula
{ statement}
LOOP

Syntax 2
DO UNTIL formula
{ statement}
LOOP

Syntax 3
DO
{ statement}
LOOP WHILE formula

Syntax 4
DO
{ statement}
LOOP UNTIL formula

The formula returns a logic value. The Do statement executes a series of statements constructing it. Syntax 1 and 2 test the formula before starting each repetition. Syntax 3 and 4 test the formula after the statement is executed. Syntax 1 and 3 continue execution as long as the formula returns true. Syntax 2 and 4 repeatedly execute the execution of statements until the formula becomes true. An example of the DO statement includes:

DO WHILE I<>X
  I=I+1
LOOP

2.3.10 FOR Statement

Unlike the REPEAT statement, the WHILE statement or the DO statement, the FOR statement needs to explicitly designate the number of times to execute loop. The following is an example of the syntax of the FOR statement:

FOR counter = initial value TO final value
{ statement}
NEXT counter

Or

FOR counter = initial value TO STEP increment of final value
{ statement}
NEXT counter

The elements of the syntax are as follows:

- The counter is a local variable of the order type which has no qualifier declared in the block with the FOR statement.
- The initial and final values are the formula compatible with the counter in assignment.
- The formula is a simple or structured statement without alteration of the counter value.
After the initial value is assigned to the counter, the FOR statement repeatedly executes the statement to increment or decrement the counter every time repetition occurs. The syntax FOR...TO increments the counter while the syntax FOR...TO...STEP adds increment to the counter. When the counter returns the same value as the final value, the FOR statement ends after a statement is executed once again. Therefore, a statement is executed once for every value between the initial and final values. When the initial value is the same as the final, a statement is executed once only. When the initial value is greater than the final in the statement FOR...TO, no statement is executed.

To control execution of loop, each formula of the initial and final values is executed once only before execution of loop.

2.4 Data Type and Variable

A type is the name of the type of data. It determines the value storable in a variable and the operation executable to a variable. Similar to the function, the formula always returns the specific types of data. Most functions claim the specified types of parameters.

2.4.1 Type

The data types of BASIC are as follows:

- Integer type
- Character type

2.4.2 Integer Type

The integer type, showing part of integers, is typically used since it allows CPU to obtain optimal processing efficiency. The variable description only in alphabets is regarded as the integer type variable.

2.4.3 String Type

A string shows a series of characters. The string type variable always comes with $ at the end. Between 0 and 255, the length of a string dynamically changes while 256 byte is allocated to memory.

2.4.4 Array Type

The array type is represented by the syntax of the following form:

DIM variable name (array quantity, ..., array quantity)

The number of elements storable in array is determined by the product of the sizes of each array type since a subscript is used to subscript an array. An integer is used for the subscript.

The number of the maximum dimension definable is 3. The subscript type is one only for the simplest one-dimensional array.

DIM MyArray(100)

For instance, DIM MyArray(100) declares the variable MyArray that stores a hundred integer type arrays. In such declaration, MyArray(3) stands for the third MyArray. If no value is assigned to part of the created array, memory is allocated to unused elements. However, similar to variables not initialized, the content of these
elements becomes random data.
With $ at the end of a variable name, the array variable becomes the string type. The array of a string is as follows.

**DIM MyString$ (10)**

The array storing 10 strings is defined under the name MyString$.
Multi-dimensional array is the array of array as shown below:

**DIM MyMatrix(10, 10)**

MyMatrix refers to the integer type array of 100×100. The way to subscript the variable MyMatrix is MyMatrix(2,45).

### 2.5 Variable

A variable is an identifier that can change a value at execution. In other words, it is a name to give to a position in memory, which enables read and write to the position. A variable acts as a data container.

#### 2.5.1 Variable Declaration

The BASIC language does not require variable declaration. A variable appearing in a statement is automatically recognized. The variable name with $ at its end means the string type variable. Otherwise they are the integer type.

### 2.6 Constant

The constant has the integer type and string type.

#### 2.6.1 Integer Type Constant

Description of a value in a formula makes a integer-type constant.
E.g. 1234

#### 2.6.2 String Type Constant

Description of a string double quoted in a formula makes a string type constant.
E.g. "Helloword"

### 2.7 Function

The BASIC language has an intrinsic function already prepared, which is used in a formula to return a value.

### 3. Command

This chapter describes the commands of the BASIC interpreter. This chapter has the following items for explanation:

Explanation: explains how to use the commands.
Format: explains the parameter format included in the command.
Parameter: explains the meaning of the parameter.
E.g.: draws the example/s of command usage.

The symbols of the format have the following meaning:

Omission [a]  a can be omitted.
Repetition <a>  a can be repeated to designate.
Selection {a|b|c|d} one of a, b, c or d is selected to designate.

3.1 File Name

3.1.1 Program Store Drive

Storage of the BASIC program occurs in the following drives, in which the BASIC program can be stored using the SAVE command. To read the stored BASIC program, use the LOAD command.

<table>
<thead>
<tr>
<th>Drive</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>E: (ZPL) / B: (DMX)</td>
<td>Permanent storage by non-volatile memory</td>
</tr>
<tr>
<td>R: (ZPL) / A: (DMX)</td>
<td>Temporary storage by volatile memory</td>
</tr>
<tr>
<td></td>
<td>Content disappears when powered off</td>
</tr>
</tbody>
</table>

3.1.2 File Name Convention

The BASIC program stored in a drive is managed under a file name, designated as the following format. Designate it with the number of characters of 13 or less in combination with a file name and extension.

Drive:file name.extension

Drive: E:, R: (ZPL) / A:, B: (DMX)
File name: arbitrary character
Extension: .BAS fixed

E.g.
E:PROGRAM.BAS
R:TEMP.BAS

3.2 Input-Output Device

3.2.1 List of Devices

Data input-output allows the BASIC program to communicate with the printer engine and the external devices connected to communication interface as shown in table 8.

<table>
<thead>
<tr>
<th>Device name</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZPL</td>
<td>Printer engine</td>
</tr>
<tr>
<td>SER</td>
<td>Serial interface</td>
</tr>
<tr>
<td>PAR</td>
<td>Parallel interface</td>
</tr>
<tr>
<td>USB</td>
<td>USB interface</td>
</tr>
</tbody>
</table>

3.2.2 Console Device
The console device displays the result of commands input by the BASIC interpreter. It is intended for input-output when a device channel is omitted by the INPUT and PRINT statements. The device sending the BASIC interpreter activation command automatically becomes the console device.

### 3.2.3 ZPL Device

This is a device for the printer engine. For the ZPL device, output the printer command. The printer executes operation according to the command. Input to the ZPL device can read status returned by the printer engine.

### 3.2.4 SER Device

The SER device is the serial interface of a printer. To communicate with the devices connected to the serial interface, data input-output is executed through the SER device.

### 3.2.5 PAR Device

The PAR device is the parallel interface of a printer. To communicate with the devices connected to the parallel interface, data input-output is executed through the PAR device.

### 3.2.6 USB Device

The USB device is the USB interface of a printer. To communicate with the devices connected to the USB interface, data input-output is executed through the USB device.

### 3.3 List of Commands

Table 9 shows the list of commands.

<table>
<thead>
<tr>
<th>No.</th>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RUN</td>
<td>Execute BASIC program</td>
</tr>
<tr>
<td>2</td>
<td>STEP</td>
<td>Step execute BASIC program</td>
</tr>
<tr>
<td>3</td>
<td>CONT</td>
<td>Interrupt and restart BASIC program</td>
</tr>
<tr>
<td></td>
<td>RESTART</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BYE</td>
<td>End BASIC interpreter</td>
</tr>
<tr>
<td></td>
<td>QUIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~JQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ZPL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSTEM</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NEW</td>
<td>Initialize BASIC program</td>
</tr>
<tr>
<td>6</td>
<td>EDIT</td>
<td>Edit lines of BASIC program</td>
</tr>
<tr>
<td>7</td>
<td>AUTO</td>
<td>Automatically input BASIC program</td>
</tr>
<tr>
<td></td>
<td>AUTONUM</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>RENUM</td>
<td>Reallocate line numbers of BASIC program</td>
</tr>
<tr>
<td>9</td>
<td>LIST</td>
<td>Display BASIC program</td>
</tr>
<tr>
<td>10</td>
<td>DELETE</td>
<td>Delete lines of BASIC program, delete files of BASIC program</td>
</tr>
<tr>
<td>11</td>
<td>LOAD</td>
<td>Read BASIC program</td>
</tr>
</tbody>
</table>
3.4 List of Statements

Table 10 shows the list of statements.

Table 10: List of statements

<table>
<thead>
<tr>
<th>No.</th>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>END</td>
<td>End execution of BASIC program</td>
</tr>
<tr>
<td>2</td>
<td>GOTO</td>
<td>Jump executing lines</td>
</tr>
<tr>
<td>3</td>
<td>STOP</td>
<td>Stop execution of BASIC program</td>
</tr>
<tr>
<td>4</td>
<td>IF</td>
<td>Judge condition</td>
</tr>
<tr>
<td>5</td>
<td>FOR</td>
<td>Execute repetition</td>
</tr>
<tr>
<td>6</td>
<td>REPEAT</td>
<td>Execute repetition</td>
</tr>
<tr>
<td>7</td>
<td>WHILE</td>
<td>Execute repetition</td>
</tr>
<tr>
<td>8</td>
<td>DO</td>
<td>Execute repetition</td>
</tr>
<tr>
<td>9</td>
<td>EXIT</td>
<td>Escape from loop</td>
</tr>
<tr>
<td>10</td>
<td>GOSUB</td>
<td>Call sub-routine</td>
</tr>
<tr>
<td>11</td>
<td>RETURN</td>
<td>Return from sub-routine</td>
</tr>
<tr>
<td>12</td>
<td>ON GOTO</td>
<td>Jump condition</td>
</tr>
<tr>
<td>13</td>
<td>RESUME</td>
<td>Return from error handling routine</td>
</tr>
<tr>
<td>14</td>
<td>SLEEP</td>
<td>Suspend execution</td>
</tr>
<tr>
<td>15</td>
<td>REM</td>
<td>Comment</td>
</tr>
<tr>
<td>16</td>
<td>LET</td>
<td>Assign variable</td>
</tr>
<tr>
<td>17</td>
<td>PRINT</td>
<td>Output data display</td>
</tr>
<tr>
<td>18</td>
<td>INPUT</td>
<td>Input data</td>
</tr>
<tr>
<td>19</td>
<td>READ</td>
<td>Input data</td>
</tr>
<tr>
<td>20</td>
<td>DATA</td>
<td>Define data variable</td>
</tr>
<tr>
<td>21</td>
<td>RESTORE</td>
<td>Change data input position</td>
</tr>
<tr>
<td>22</td>
<td>OPEN</td>
<td>Device open</td>
</tr>
<tr>
<td>23</td>
<td>CLOSE</td>
<td>Device close</td>
</tr>
<tr>
<td>24</td>
<td>LINPUT</td>
<td>Input data line</td>
</tr>
<tr>
<td>25</td>
<td>CLRERR</td>
<td>Release printer error</td>
</tr>
<tr>
<td>26</td>
<td>SETERR</td>
<td>Set printer error</td>
</tr>
<tr>
<td>27</td>
<td>INBYTE</td>
<td>Input 1 byte of device data</td>
</tr>
<tr>
<td>28</td>
<td>OUTBYTE</td>
<td>Output 1 byte of device data</td>
</tr>
<tr>
<td>29</td>
<td>DECLARE</td>
<td>Define array</td>
</tr>
<tr>
<td>30</td>
<td>DIM</td>
<td>Define array</td>
</tr>
<tr>
<td>31</td>
<td>BASE</td>
<td>Define array radix</td>
</tr>
<tr>
<td>32</td>
<td>DEF</td>
<td>Define function</td>
</tr>
<tr>
<td>33</td>
<td>BREAK</td>
<td>Interrupt BASIC program</td>
</tr>
<tr>
<td>34</td>
<td>ERROR</td>
<td>Display error message</td>
</tr>
<tr>
<td>35</td>
<td>RANDOMIZE</td>
<td>Initialize random numbers</td>
</tr>
</tbody>
</table>
3.5 List of Functions

Table 11 shows the list of functions.

Table 11: List of functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MID$</td>
<td>Extract string</td>
</tr>
<tr>
<td>2</td>
<td>RIGHT$</td>
<td>Extract leading string</td>
</tr>
<tr>
<td>3</td>
<td>LEFT$</td>
<td>Extract end of string</td>
</tr>
<tr>
<td>4</td>
<td>STRING$</td>
<td>Repeat string</td>
</tr>
<tr>
<td>5</td>
<td>CHR$</td>
<td>Stringize numeric value</td>
</tr>
<tr>
<td>6</td>
<td>STR$</td>
<td>Convert numeric value to string</td>
</tr>
<tr>
<td>7</td>
<td>SPACE$</td>
<td>Blank string</td>
</tr>
<tr>
<td>8</td>
<td>MKI$</td>
<td>Stringize 16 bit of integer value</td>
</tr>
<tr>
<td>9</td>
<td>DATE$</td>
<td>Obtain date</td>
</tr>
<tr>
<td>10</td>
<td>TIME$</td>
<td>Obtain time</td>
</tr>
<tr>
<td>11</td>
<td>EXTRACT$</td>
<td>Extract string</td>
</tr>
<tr>
<td>12</td>
<td>LCASE$</td>
<td>Lower-case conversion</td>
</tr>
<tr>
<td>13</td>
<td>UCASE$</td>
<td>Upper-case conversion</td>
</tr>
<tr>
<td>14</td>
<td>LTRIM$</td>
<td>Delete first space character</td>
</tr>
<tr>
<td>15</td>
<td>RTRIM$</td>
<td>Delete trailing space character</td>
</tr>
<tr>
<td>16</td>
<td>HEX$</td>
<td>Hexadecimal conversion</td>
</tr>
<tr>
<td>17</td>
<td>OCT$</td>
<td>Octal conversion</td>
</tr>
<tr>
<td>18</td>
<td>REPEAT$</td>
<td>Repeat string</td>
</tr>
<tr>
<td>19</td>
<td>MOD</td>
<td>Residue</td>
</tr>
<tr>
<td>20</td>
<td>ORD</td>
<td>Digitalize characters</td>
</tr>
<tr>
<td>21</td>
<td>LEN</td>
<td>Set up string length</td>
</tr>
<tr>
<td>22</td>
<td>VAL</td>
<td>Convert numeric string to numeric value</td>
</tr>
<tr>
<td>23</td>
<td>ASC</td>
<td>Digitalize characters</td>
</tr>
<tr>
<td>24</td>
<td>INSTR</td>
<td>Search string</td>
</tr>
<tr>
<td>25</td>
<td>MKI</td>
<td>Convert 2 byte of string to 16 bit of integer</td>
</tr>
<tr>
<td>26</td>
<td>DATE</td>
<td>Obtain date</td>
</tr>
<tr>
<td>27</td>
<td>TIME</td>
<td>Obtain time</td>
</tr>
<tr>
<td>28</td>
<td>MAX</td>
<td>Obtain maximum value</td>
</tr>
<tr>
<td>29</td>
<td>MIN</td>
<td>Obtain minimal value</td>
</tr>
<tr>
<td>30</td>
<td>MAXLEN</td>
<td>Obtain the number of maximum storable characters in string variable</td>
</tr>
<tr>
<td>31</td>
<td>MAXIMUM</td>
<td>Obtain maximum integer value</td>
</tr>
<tr>
<td>32</td>
<td>INPUT$</td>
<td>Input data from device</td>
</tr>
<tr>
<td>33</td>
<td>GET$</td>
<td>Input data from device</td>
</tr>
<tr>
<td>34</td>
<td>EOF</td>
<td>Judge EOF</td>
</tr>
<tr>
<td>35</td>
<td>INKEY$</td>
<td>Input 1 character from console</td>
</tr>
<tr>
<td>36</td>
<td>SEARCHTO$</td>
<td>Input data from device to search string</td>
</tr>
<tr>
<td>37</td>
<td>ISERROR</td>
<td>Obtain error condition of printer</td>
</tr>
<tr>
<td>38</td>
<td>ISWARNING</td>
<td>Obtain warning error condition of printer</td>
</tr>
<tr>
<td>39</td>
<td>DATAREADY</td>
<td>Obtain state of device data preparation</td>
</tr>
<tr>
<td>40</td>
<td>ERL</td>
<td>Obtain line number at error</td>
</tr>
</tbody>
</table>
3.6 Program Execution

3.6.1 RUN Command

【Explanation】
This command executes the BASIC program.

【Format】
① RUN
② RUN line number
③ RUN file name

① Executes the BASIC program currently loading to memory.
② Executes from the line number position designated by the BASIC program currently loading to memory.
③ Loads and executes the program designated by a file name saved in the drive. The BASIC program in a command other than the line number designation is executed from the beginning of the BASIC program.

【Parameter】
Line number: execution started.
file name: designate a file name to execute in a string.

【E.g.】
Ready
>LOAD "E:SER.BAS"
Ready
>RUN

3.6.2 STEP Command

【Explanation】
This command executes a statement of the BASIC program to get back to the state of command input. It can restart continuous execution with the CONT command in the state where the program execution is suspended.

【Format】
STEP

【E.g.】
3.6.3 CONT and RESTART Commands

**Explanation**
These commands restart execution of the BASIC program whose execution is suspended by the STEP or BREAK command, from the suspended spot.

**Format**
CONT
RESTART

**E.g.**
Ready
>STEP
Breaking at line 1010
Ready
>STEP
Breaking at line 1020
Ready
>CONT
Ready

3.7 End Interpreter

3.7.1 BYE, QUIT, ~JQ, ZPL and SYSTEM Commands

**Explanation**
These commands end the BASIC interpreter.

**Format**
BYE
QUIT
~JQ
ZPL
SYSTEM

**E.g.**
Ready
>ZPL

3.8 Edit BASIC Program

3.8.1 NEW Command

**Explanation**
This command clears all the BASIC programs currently read in memory.

**Format**
NEW

**E.g.**
Ready
>NEW
Ready

3.8.2 EDIT Command

**Explanation**
This command changes a line of the BASIC program. Since command input leads to displaying the line of the designated line number, it timely changes programs.

**[Format]**
EDIT line number

**[Parameter]**
Line number: designate the line number of a line to change.

**[E.g.]**
Ready
>EDIT 1010

### 3.8.3 AUTO and AUTONUM Commands

**[Explanation]**
This command automatically displays the line number to set up the state to input the BASIC program.

**[Format]**
AUTO [line number[, increment]]
AUTONUM [line number[, increment]]

**[Parameter]**
Line number: starting line number of a line to input
  Default: value previously designated Initial value: 10
Increment: line number increment
  Default: value previously designated Initial value: 10

**[E.g.]**
Ready
>AUTO 1000,10
1000

### 3.8.4 RENUM Command

**[Explanation]**
This command reallocates the line number of the BASIC program.

**[Format]**
RENUM [line number[, increment]]

**[Parameter]**
Line number: starting line number Default: 10
Increment: line number increment Default: 10

**[E.g.]**
Ready
>RENUM 1000,10
Ready

### 3.8.5 LIST Command

**[Explanation]**
This command displays the list of the BASIC program on the console. When a line number is designated, it displays the list of the BASIC program within the range of the line numbers designated.

**[Format]**
LIST [initial line number[-final line number]]
Citizen BASIC Interpreter

【Parameter】
Initial line number: initial line number
Final line number: final line number

The entire omission of the parameters displays all the list of the BASIC program. Designation of the initial line number only displays the BASIC program from the initial line number to the final.

【E.g.】
Ready
>LIST
1000 A=1
1010 B=2
1020 C=3
Ready
>LIST 1010
1010 B=2
Ready

3.8.6 DELETE Command

【Explanation】
This command deletes the source line of the BASIC program of the line number designated as a parameter.

【Format】
DELETE initial line number [-final line number]

【Parameter】
Initial line number: initial line number
Final line number: final line number

Designation of the initial and final line numbers deletes the line of the BASIC program from the initial line number to the final. Designation of the initial line number only displays its line number.

【E.g.】
Ready
>LIST
1000 A=1
1010 B=2
1020 C=3
Ready
>DELETE 1020
Ready
>LIST
1000 A=1
1010 B=2
Ready

3.9 Save BASIC Program

3.9.1 LOAD Command

【Explanation】
This command reads the BASIC program from the drive to memory.

【Format】
LOAD file name

【Parameter】 File name: file name of the BASIC program to read

【E.g.】
3.9.2 SAVE and STORE Commands

[Explanation]
These commands save the BASIC program on the memory to the drive.

[Format]
①SAVE file name
②STORE file name

[Parameter]
File name: BASIC program file of destination to save

[E.g.]
Ready
>SAVE "R:TEST.BAS"
Ready

3.9.3 DIR and FILES Commands

[Explanation]
These commands display on the console the list of the BASIC program saved in the drive.

[Format]
①DIR [file name wildcard]
②FILES [file name wildcard]

[Parameter]
File name wildcard: file name wildcard of the BASIC program to display

It displays the list of the BASIC program corresponding to the file name wildcard. 
“*” and “?” have the following meaning:

*: arbitrary string
?: arbitrary characters

Omission of the file name wildcard as a parameter displays all drives and files.

[E.g.]
Ready
>DIR "E:*.*.BAS"

- DIR E:*.*.BAS
  * R:TEST.BAS 643
  * E:EXAMPLE1.BAS 643
  * E:EXAMPLE2.BAS 408

-4193660 byte free R: RAM
-358148 byte free E: ONBOARD FLASH
Ready
3.9.4 DELETE Command

**Explanation**
This command deletes the BASIC program file saved in the drive designated as a parameter.

**Format**
DELETE file name

**Parameter**
File name: file name of the BASIC program to delete

**E.g.**
Ready
>DELETE "E:TEST.BAS"
Ready

3.10 BASIC Program Execution Control Command

3.10.1 END Statement

**Explanation**
This statement ends execution of the BASIC program.

**Format**
END

**E.g.**
1000 PRINT 123
1010 END

3.10.2 GOTO Command

**Explanation**
This command moves execution to the line of the designated line number.

**Format**
GOTO line number

**E.g.**
1000 C=1
1010 IF C=10 THEN GOTO 2000
1020 C=C+1
1030 GOTO 1010
2000 END

3.10.3 STOP Statement

**Explanation**
This statement suspends execution of the BASIC program to set command input ready. Restart of execution is allowed by the CONT and RESTART commands.

**Format**
STOP

**E.g.**
>LIST
1000 C=1
1010 IF C=10 THEN GOTO 1040
Citizen BASIC Interpreter

1020 C=C+1
1030 GOTO 1010
1040 STOP
1050 PRINT "FINISH"
2000 END

Ready
>RUN
Stopped at line 1040
Ready
>RESTART
FINISH
Ready

3.10.4 IF Statement

【Explanation】
Judging the conditional expression, this statement executes the statement with the THEN clause if it is true, and that with the ELSE clause if false.

【Format】
IF conditional expression THEN{ statement| line number} [ELSE{ statement| line number}]

When designating a line number to the THEN or ELSE clause, jump to the line. The ELSE clause can be omitted. When the ELSE clause is omitted and the conditional expression is false, then the execution moves to the next line.

【E.g.】
1010 IF C=10 THEN GOTO 1040
If the value of the variable C is 10, jump to the line number 1040.

1010 IF C=127 THEN C=32
if the value of the variable C is 127, rewrite it to 32.

3.10.5 Structured IF Statement

【Explanation】
Judging the conditional expression, this statement executes the statement with the THEN block if the result is true, and that with the ELSE block if false. After the THEN and ELSE blocks are executed, start execution subsequent to ENDIF.

【Format】
IF conditional expression THEN
Statement
[ELSE IF conditional expression THEN
Statement]
[ELSE
Statement]
ENDIF

ELSE and ELSE IF clauses can be omitted.

【E.g.】
1000 INPUT "(XX)";A$1010 IF MID$(A$,1,1 )="0" THEN
1020 PRINT 0;
1030 IF MID$(A$,2,1 )="0" THEN
1040 PRINT 0;
1050 ELSE
1060 PRINT 1;
1070 END IF
1080 PRINT "-";
3.10.6 FOR – NEXT Loop

【Explanation】
This loop executes the lines from the FOR statement to the NEXT statement by repeating them several times. Firstly assign the initial value to the variable. Then, execute the statements in the FOR loop. If in the NEXT statement the variable exceeds the final value, execution moves to the one next to the NEXT statement. If not, add the increment value to the variable to once again execute the statements within the FOR loop. Repeating these, execute the statements within the FOR loop the designated number of times repeatedly.

【Format】
FOR variable =initial value TO final value [STEP increment value]
Statement
NEXT variable

When the STEP clause is omitted, the increment becomes 1.

【Parameter】
Initial value: Initial value to assign to the variable
Final value: Conditional value for exiting the FOR loop
Increment value: Increment value for incrementing the value of the variable in the NEXT statement

【E.g.】
>LIST
1000 T=0
1010 FOR C=1 TO 10
1020 T=T+C
1030 NEXT C
1040 PRINT T
Ready
>RUN
55
Ready

The sum from 1 to 10 is calculated.

3.10.7 REPEAT Statement – UNTIL Loop

【Explanation】
These execute the lines from the REPEAT statement to the UNTIL statement until the conditional expression becomes true.

【Format】
REPEAT
Statement
UNTIL conditional expression

[S.e.g.]
>LIST
1000 T=0
1010 C=1
1020 REPEAT
1030 T=T+C
1040 C=C+1
1050 UNTIL C>10
1060 PRINT T
Ready
>RUN
55
Ready

3.10.8 WHILE – WEND Loop

[Explanation]
This loop executes the statements between the WHILE and WEND loops repeating from the WHILE statement to the WEND statement while the result of the conditional expression is true.

[Format]
WHILE conditional expression
Statement
WEND

[S.e.g.]
>LIST
1000 T=0
1010 C=1
1020 WHILE C<=10
1030 T=T+C
1040 C=C+1
1050 WEND
1060 PRINT T
Ready
>RUN
55
Ready

3.10.9 DO – LOOP Loop

[Explanation]
This loop repeatedly executes the lines from the DO statement to the LOOP statement until their conditions are fulfilled.

[Format]
① DO WHILE conditional expression
Statement
LOOP

② DO UNTIL conditional expression
Statement
LOOP

③ DO
Statement
Citizen BASIC Interpreter

LOOP WHILE conditional expression

④
DO
Statement
LOOP UNTIL conditional expression

① Repeatedly executes the statements between DO and LOOP loops while the conditional expression is true. The conditional expression is judged before the statements within the loop are executed.
② Repeatedly executes the statements between the DO and LOOP loops until the conditional expression becomes true. The conditional expression is judged before the statements within the loop are executed.
③ Repeatedly executes the statements between the DO and LOOP loops while the conditional expression is true. The conditional expression is executed after the statements within the loop are executed.
④ Repeatedly executes the statements between the DO and LOOP loops until the conditional expression becomes true. The conditional expression is judged after the statements within the loop are executed.

【E.g.】
>LIST
1000 T=0
1010 C=1
1020 DO
1030 T=T+C
1040 C=C+1
1050 LOOP UNTIL C>10
1060 PRINT T
Ready
>RUN
55
Ready

3.10.10 EXIT Statement

【Explanation】This statement exits from the WHILE – WEND loop, FOR – NEXT loop, REPEAT – UNTIL loop and DO – LOOP loop.

【Format】
EXIT [( WHILE| FOR| REPEAT| DO)]

【Parameter】
WHILE: Exits from the innermost WHILE – WEND loop
FOR: Exits from the innermost FOR – NEXT loop
REPEAT: Exits from the innermost REPEAT – UNTIL loop
DO: Exits from the innermost DO – LOOP loop

When a parameter is omitted, exit from one of the WHILE – WEND loop, FOR – NEXT loop, REPEAT – UNTIL or DO – LOOP loop.

【E.g.】
>LIST
1000 T=0
1010 C=1
1020 DO WHILE 1
Citizen BASIC Interpreter

1030 T=T+C
1040 C=C+1
1041 IF C>10 THEN EXIT
1050 LOOP
1060 PRINT T
Ready
>RUN
55
Ready

3.10.11 GOSUB Statement

[Explanation]
This statement moves execution to the line designated by the line number to get the execution back to the one immediately after the GOSUB statement with the RETURN statement.

[Format]
GOSUB line number

[Parameter]
Line number: designate the line number of the line to move its execution

[E.g.]
>LIST
1000 T=0
1010 C=1
1020 FOR C=1 TO 10
1030 GOSUB 1070
1040 NEXT C
1050 PRINT T
1060 END
1070 T=T+C
1080 RETURN
Ready
>RUN
55
Ready

3.10.12 RETURN Statement

[Explanation]
When called with a GOSUB statement, this statement gets execution back to the one immediately after the GOSUB statement.

[Format]
RETURN

3.10.13 ON – GOTO Statement

[Explanation]
This statement jumps to the designated line according to the value expressed by the formula.

[Format]
ON formula { GOTO| GOSUB} line number 1 [, line number 2[, line number 3][, line number n]} ...

[Parameter]
Formula: Must be the formula that becomes an integral value
Jumps to the line number corresponding to the value expressing by the formula
Jumps to the line number 1 when the value of the formula is 1
Jumps to the line number n when the value of the formula is n

GOTO: Jumps to the line number by operation of the GOTO statement
GOSUB: Jumps to the line number by operation of the GOSUB statement
Line number n: Designates the line number of the destination to jump depending on the value of the formula

【E.g.】
>LIST
1000 FOR C=1 TO 10
1010 ON (C MOD 2)+1 GOSUB 2000,3000
1020 NEXT C
1030 END
2000 PRINT C,"EVEN"
2010 RETURN
3000 PRINT C,"ODD"
3010 RETURN
End
>RUN
1 ODD
2 EVEN
3 ODD
4 EVEN
5 ODD
6 EVEN
7 ODD
8 EVEN
9 ODD
10 EVEN
End

3.10.14 ON – ERROR – GOTO Statement

【Explanation】
This statement sets the destination to jump by interrupting when an error occurs in execution of the BASIC program. When an error occurs in execution by the BASIC program, execution of the spot concerned is interrupted to move execution to the line designated by the line number.

【Format】
ON ERROR GOTO line number

【Parameter】
line number: designates the destination to jump for the case where an error occurs in execution

【E.g.】
>LIST
1000 ON ERROR GOTO 9000
1010 A=1000/0
1011 PRINT A
1020 END
9000 PRINT "RUN-TIME ERROR on ";ERL
9010 END
End
>RUN
RUN-TIME ERROR on 1010
End
3.10.15 RESUME Statement

**Explanation**
This statement recovers from error handling routine by the ON ERROR GOTO statement.

**Format**
RESUME [line number]

**Parameter**
Line number: designates the line number to recover. When omitting the line number, this gets back to the line from which an error derives.

**E.g.**

```plaintext
>LIST
1000 ON ERROR GOTO 2000
1010 A=1000/0
1020 PRINT "FINISH A=";A
1030 END
2000 PRINT "DIVIDE BY ZERO"
2010 RESUME 1020
Ready
>RUN
DIVIDE BY ZERO
FINISH A=0
Ready
```

3.10.16 SLEEP Statement

**Explanation**
This statement suspends execution of the BASIC program for a certain period of time.

**Format**
SLEEP latency time

**Parameter**
Latency time: designates the time to wait execution by seconds.

**E.g.**

```plaintext
1000 SLEEP 10
Stop execution for 10 seconds.
```

3.11 Comment

3.11.1 REM Statement

**Explanation**
This statement is the line to describe comments. They can be described after REM onwards which does not affect execution of the BASIC program.

**Format**
REM [comment statement]

**E.g.**

```plaintext
>LIST
1000 REM Hello world
1010 REM The quick brown fox jumps over the lazy dog
1011 REM
1020 PRINT "FINISH"
Ready
>RUN
FINISH
Ready
```
3.11.2 Comment in Line

[Explanation]
When the character ’’ (single quotation) or ’!’ appears in the middle of a line, the subsequent line is the comment.

[E.g.]
>LIST
1000  T=0    'Clear TOTAL
1010  FOR C=1 To 10  !LOOP C=1 to 10
1020  T=T+C    !Add C to TOTAL
1030  NEXT C
1040  PRINT T   !Output TOTAL Result
Ready
>RUN
55
Ready

3.12 Assignment Statement

3.12.1 LET Assignment Statement

[Explanation]
This statement assigns the value, the calculation results etc. on the right of the equal to the left-hand variable.

[Format]
LET  <assignment statement>
<assignment statement>:= variable = formula

[E.g.]
>LIST
1000  LET A=123
1010  PRINT A
Ready
>RUN
123
Ready

3.13 Data Input-Output

3.13.1 PRINT Statement

[Explanation]
The PRINT statement outputs the value of a formula to the console or the device. It is output to the device of the designated file identifier while it is output to the console when omitting the identifier. Use of the USING phrase can formulate the output form.

[Format]
PRINT [file identifier,] [USING phrase;] <[[formula( ,;)]]>.....

File identifier: Designates the device of the destination to output
Outputs the device data of the channel number expressed in #n
Outputs to the console when omitted

USING phrase: Designates the output form
Formula: Data to output
Formula with the values of string type and integer type
Separator: Separation of a formula by “;” or “,”
Designates separation between data expressed by a formula
";" does not insert space

【USING phrase】
The USING phrase defines the output format by the following format.

The USING format designates the output format of the value by the string. The definitions of the string are as follows:

! First 1 character of the string
& Output string
\ \ Output string of the designated digits
\ The number of digits is determined by the number from \ to \"\"
# Numeric value. The number of characters of "#" is the number of digits.
. Decimal point. There is "." on the right of the point.
"0" is added if not fulfilled
+ Display positive and negative. Display "+" for positive,
- for negative
**$ Display positive and negative. Display space for
positive, "+" for negative
** Number of digits from "+" and "+" are displayed in the blank of the left digit.
E.g.: "**$##", result: **$123
** "**" is displayed in the blank of the left digit. E.g.:
"**##", result: **123
$$ Insert the separating symbol "," every 3 digits in the integer part
E.g. "###,###,###", result: 12,345,678
^^^^ Display exponents. E.g.: ".###^^^^", result: 1.234E+02

【E.g.】
PRINT 123;456
Result
123456

PRINT 123,456
Result
123 456

PRINT USING"######## ??????";123;456
Result
123 456

PRINT USING"**$###*$$###";123;456
Result
**$123**$456

PRINT USING"!!";"ABC";"XYZ"
Result
AX

PRINT USING"&-&";"ABC";"XYZ"
Result
ABC-XYZ
3.13.2 INPUT Statement

【Explanation】
This statement inputs data from the device.

【Format】 INPUT [channel number:][prompt 1 ]variable<[,variable]>>

【Parameter】
Channel number: Designates the channel number of the device to input data. If omitted, data are input from the console.
Prompt: Designates the prompt message to display to the device by string before data input. The prompt can be designated only when omitting the channel number to input data from the console.
Variable: Designates the variable to which stores the input data. When designating several variables, separate them with commas (,).

【E.g.】
INPUT A
INPUT S$
INPUT "How many?";N

INPUT A,B
Input data
1, 2
Result
A=1
B=2

3.13.3 READ Statement

【Explanation】
This statement reads the to the variable the data defined in the DATA statement.

【Format】 READ variable <[, variable]>•

【Parameter】 Variable: designates the variable to which stores read data.

【E.g.】
>LIST
1000 WHILE 1
1010 READ A,B
1020 IF A=9999 THEN EXIT
1030 PRINT A,B
1040 WEND
1050 END
1060 DATA 1,2,3,4,5,6,7,8,9,10
1080 DATA 9999,0
Ready
3.13.4 DATA Statement

**Explanation**
This statement defines the data to read to the variable by the READ statement.

**Format**
DATA data <[, data]>

**Parameter**
Data: defines the numeric constant or the string constant. The value defined here is read to the variable of the READ statement. The data type must correspond to the variable type of the READ statement.

**E.g.**
>LIST
1000 WHILE 1
1010 READ A, B$
1020 IF A=9999 THEN EXIT
1030 PRINT A;"-"; B$
1040 WEND
1050 END
1060 DATA 1,"PEN"
1070 DATA 2,"DESK"
1080 DATA 3,"NOTE"
1090 DATA 9999,""
Ready
>RUN
1-PEN
2-DESK
3-NOTE
Ready

3.13.5 RESTORE Statement

**Explanation**
This statement designates the position of the DATA statement to read by the READ statement.

**Format**
RESTORE [line number]

**Parameter**
Line number: Designates the line number of the DATA to start reading
Omission of the line number starts reading from the initial DATA statement.

**E.g.**
>LIST
1000 DIM A$(5), B$(5)
1010 FOR C=1 TO 5
1020 READ A$(C)
1030 NEXT C
1040 RESTORE
1050 FOR C=1 TO 5
1060 READ B$(C)
1070 NEXT C
1080 FOR C=1 TO 5
1090 PRINT A$(C);"-";B$(C)
1100 NEXT C
1110 END
1120 DATA "PEN"
1130 DATA "DESK"
1140 DATA "NOTE"
1150 DATA "INK"
1160 DATA "ERASER"
Ready
>RUN
PEN-PEN
DESK-DESK
NOTE-NOTE
INK-INK
ERASER-ERASER
Ready

3.13.6 OPEN Statement

【Explanation】
This statement opens the device.

【Format】
① OPEN channel number: NAME device name [,ACCESS{ IN| OUT| INOUT} ]
② OPEN device name [FOR{INPUT| OUTPUT| INOUT}]AS channel number

【Parameter】
Channel number: designates in #n format the channel number designated in the input-output statement.
Device name: designates the following devices to open:
"ZPL" printer engine
"SER" serial interface
"PAR" parallel interface
"USB" USB interface
IN: opens the device in input mode.
OUT: opens the device in output mode.
INOUT: opens the device in input-output mode.

Omission of the ACCESS phrase or FOR phrase opens the device in input-output mode. The device opened in input mode allows input only while that opened in output allows output only. That opened in input-output mode allows both input and output.

【E.g.】
>LIST
1000 OPEN "SER" FOR INOUT AS #1
1010 WHILE 1
1020 A$=INPUT$(1,#1)
1030 IF A$=CHR$(27) THEN EXIT
1040 PRINT #1,A$;
1050 WEND
1060 PRINT
1070 CLOSE #1
1080 END
Ready
>RUN
3.13.7 CLOSE Statement

[Explanation]
This statement closes the device opened by the OPEN statement.

[Format]
CLOSE channel number

[Parameter]
Channel number: Designates in #n format the channel number of the device opened by the OPEN statement

[E.g.]
CLOSE #1

3.13.8 LINPUT Statement

[Explanation]
This statement inputs the data for a line from the console or device to store the result to the variable. The end of a line in the data to input is judged by the CR code.

[Format]
LINPUT [prompt 1]string type variable
LINPUT [channel number,] string type variable
① Inputs data from the console to store them to the variable.
② Inputs data from the device expressed by the channel number to store them to the variable.

[Parameter]
Prompt: Designates the message to display on the console before data input.
When omitted, "?" is displayed
Channel number: Designates the channel number of a device in #n format when inputting data from such device.
String type variable: Designates the variable to which stores data for a line input. Designation of the string type variable is required.

[E.g.]
>LIST
1000 LINPUT "DATA?";A$
1010 PRINT A$
Ready
>RUN
DATA?123
123
Ready

>LIST
1000 OPEN "SER" FOR INPUT AS #1
1010 LINPUT #1,A$
1020 PRINT A$
1030 CLOSE #1
Ready
>RUN
3.13.9 CLRERR Statement

**Explanation**
This statement cancels the error condition of the printer.

**Format**
CLRERR

**E.g.**
>CLRERR

3.13.10 SETERR Statement

**Explanation**
This statement sets the printer in the error condition.

**Format**
SETERR

**E.g.**
>LIST
1000 SETERR
1010 LINPUT A$
1020 CLRERR
Ready

3.13.11 INBYTE Statement

**Explanation**
This statement reads 1 byte of data from the designated device to store the value to the variable. When data are not ready in the device, it is stored as the value of 0.

**Format**
INBYTE [channel number:] variable

**Parameter**
- **Channel number:** Designates in #n format the channel number of the device to read.
  When omitted, #1 is deemed designated.

- **Variable:** Designates the variable to which 1 byte of data read from the device.
  The integer type and string type variable can be designated for the variable.
  When data are not ready in the device, it is stored as the value of 0.

**E.g.**
>LIST
1010 OPEN "SER" FOR INPUT AS #1
1020 DO
Citizen BASIC Interpreter

1030 INBYTE #1:A
1040 LOOP UNTIL A<>0
1050 PRINT A
1060 CLOSE #1
Ready
>RUN
(input "1")
49
Ready

3.13.12 OUTBYTE Statement

【Explanation】
This statement outputs 1 byte of data for the designated device.

【Format】
OUTBYTE [channel number:]formula

【Parameter】
Channel number: Designates in #n format the channel number of device to output data. When omitted, #1 is deemed designated.
Formula: Gives data to output. For the integer type, its value is output as 1 byte of data. The range of the value must stay between 0 and 255. For the string type, the first 1 byte of the string is output as data.

【E.g.】
>LIST
1000 OPEN #1:NAME "SER", ACCESS OUTPUT
1010 A$=INPUT$(1)
1020 OUTBYTE #1:A$
1030 PRINT #1,
1040 CLOSE #1
Ready
>RUN
(input "1" )
1
Ready

3.14 Definition Statement
3.14.1 DECLARE Statement

【Explanation】
This statement defines the array variable.

【Format】
DECLARE data type variable<[, variable]>...
<Data type>: ={NUMERIC|STRING}

【Parameter】
Data type: Designates NUMERIC when defining the integer type array variable.
Designates STRING when defining the string type array variable.
Variable: Designates the number of array elements in brackets after the variable name.
When the data type is the string type, "$" is necessary to be attached at the end of the variable name.

【E.g.】
Citizen BASIC Interpreter

>LIST
1000 DECLARE NUMERIC A(10)
1010 DECLARE STRING X$(10)
1020 FOR I=1 TO 10
1030 READ A(I)
1040 X$(I)=STR$(A(I))
1050 NEXT I
2000 FOR I=1 TO 10
2010 PRINT A(I);"-";X$(I)
2020 NEXT I
9000 DATA 1,2,3,4,5,6,7,8,9,10
Ready
>RUN
1-1
2-2
3-3
4-4
5-5
6-6
7-7
8-8
9-9
10-10
Ready

3.14.2 DIM Statement

【Explanation】
This statement defines the array variable.

【Format】
DIM array variable[, array variable]...

【Parameter】
Array variable: Designates the number of array elements in brackets after the variable name to define the array variable.
When it is the string type array variable, "$" is attached at the end of the variable name.
E.g. A(10), B$(10)

【E.g.】
>LIST
1000 DIM A(10), X$(10)
1020 FOR I=1 TO 10
1030 READ A(I)
1040 X$(I)=STR$(A(I))
1050 NEXT I
2000 FOR I=1 TO 10
2010 PRINT A(I);"-";X$(I)
2020 NEXT I
9000 DATA 1,2,3,4,5,6,7,8,9,10
Ready
>RUN
1-1
2-2
3-3
3.14.3 BASE Statement

【Explanation】
This statement designates whether the basic value of the array subscript starts from 0 or 1. The default is 1.

【Format】
BASE { 0 | 1 }

【Parameter】
0: Sets to start the array subscript from 0.
1: Sets to start the array subscript from 1.

【E.g.】
>LIST
1000 BASE 0
1010 DIM A(10)
1020 FOR I=0 TO 10
1030 A(I)=I
1040 NEXT I
1050 FOR I=0 TO 10
1060 PRINT A(I)
1070 NEXT I
Ready
>RUN
0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
Ready

3.14.4 DEF

【Explanation】
This defines the user function.

【Format】
DEF function name (argument list)=format

【Parameter】
Function name: Designates the function name used to call the function.
Argument list: Defines as a dummy argument the value to pass to the function. Several arguments existing are separated with commas.

Formula: Defines the formula that returns the results as a function.

【How to call a function】To call a function, describe the actual argument put in brackets at the end of the function name. The actual argument is passed to the dummy argument, and the result as a function is returned.

【E.g.】
>LIST
1000 DEF FNM(R,Q)=R*Q
1010 FOR I=1 TO 4
1011 J=I
1020 PRINT I;;"*";J;;”=";FNM(I,J)
1030 NEXT I
Ready
>RUN
1*1=1
2*2=4
3*3=9
4*4=16
Ready

3.15 Debug Function
3.15.1 DEBUG Command

【Explanation】This command switches the debug mode.

【Format】
DEBUG [{ ON| OFF}]

【Parameter】
ON: enables the debug mode
OFF: disables the debug mode

Omission of a parameter displays the current state of the debug mode.

【E.g.】
Ready
>DEBUG ON
Ready
>DEBUG
DEBUG ON
Ready

3.15.2 TRACE Command

【Explanation】
This command switches the trace mode. When the trace mode becomes enabled, the following information is displayed on the console. The tracing result is displayed when the debug mode is enabled.

・Line number of the executed line
・Content of the rewritten variable

【Format】
Citizen BASIC Interpreter

TRACE [(ON| OFF)]

【Parameter】
ON: with the trace mode
OFF: without the trace mode

Omission of a parameter displays the current state of the trace mode.

【E.g.】
>LIST
1000 DEBUG ON
1010 TRACE ON
1020 T=0
1030 FOR I=1 TO 3
1040 T=T+I
1050 NEXT I
Ready
>RUN
<TRACE> 1020
<TRACE> T=0
<TRACE> 1030
<TRACE> I=1
<TRACE> 1040
<TRACE> T=1
<TRACE> 1050
<TRACE> I=2
<TRACE> 1040
<TRACE> T=3
<TRACE> 1050
<TRACE> I=3
<TRACE> 1040
<TRACE> T=6
<TRACE> 1050
<TRACE> I=4
Ready

3.15.3 BREAK Statement

【Explanation】
This statement suspends execution of the BASIC program to set the state of command input. To restart execution of the BASIC program, input the CONT command. To interrupt execution by the BREAK statement, it must be set to the debug mode by the DEBUG command.

【Format】
BREAK

【E.g.】
>LIST
1000 T=0
1010 FOR I=1 TO 3
1020 T=T+I
1030 BREAK
1040 NEXT I
Ready
>DEBUG ON
Ready
>RUN
Breaking at line 1030
Ready
>CONT
Breaking at line 1030
Ready

3.15.4 ERROR Statement

【Explanation】
This statement displays the error message corresponding to the error number on the console.

【Format】
ERROR error number

【Parameter】
Error number: designates the error number to display the error message.

【E.g.】
Ready
>ERROR 10
Illegal string size
Ready

3.16 Other Statements
3.16.1 RANDOMIZE Statement

【Explanation】
This statement initializes the randomizer at the designated value.

【Format】
RANDOMIZE [integer]

【Parameter】
Integer: Designates an arbitrary integral value. When omitted, the BASIC interpreter initializes the randomizer at a random value.

【E.g.】
>RANDOMIZE 123

3.16.2 ZBICOMPATI Statement

【Explanation】
This statement designates ZBI compatibility operation.

【Format】

【Parameter】
ON: enables ZBI compatibility operation.
OFF: disables ZBI compatibility operation. For ZBI compatibility operation, refer to Table 12.

Table 12: ZBI compatibility operation

<table>
<thead>
<tr>
<th></th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output PRINT statement</td>
<td>Do not add space to numeric value</td>
<td>Add space to numeric value</td>
</tr>
<tr>
<td>Variable of numeric value</td>
<td>Integer type only</td>
<td>Floating point type available for use</td>
</tr>
</tbody>
</table>
3.17 String Function

3.17.1 MID$ Function

【Explanation】
MID$ function returns the partial string of the specific number of bytes of the specific part of the string. MID$ function returns the number of bytes, designated by Count, of the partial string starting at the position of the number of bytes designated by Start. If the length of Start exceeds that of Text$, then the MID$ function returns a blank string. If the number of bytes designated by Count exceeds that of the designatable number, then only the designatable number of bytes is returned.

【Format】
MID$(Text$,Start,Count)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String intended for character retrieval</td>
</tr>
<tr>
<td>Start</td>
<td>Integer</td>
<td>Position for character retrieval</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Number of bytes for character retrieval</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 ZBICOMPATI ON
1010 A=123.456
1020 B=234.567
1030 PRINT A;B
1040 ZBICOMPATI OFF
1050 A=123.456
1060 B=234.567
1070 PRINT A;B
Ready
>RUN
123234
123.456 234.567
Ready

3.17.2 RIGHT$ Function

【Explanation】
The RIGHT$ function returns the partial string of the designated length that is displayed at the end of the string. It returns the number of characters within the Count from the end of Text$. For instance, therefore, if the Text is the string of “Programmer” with its Count of 7, RIGHT$ returns the string of “grammar”.

【Format】
RIGHT$(Text$,Count)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String intended for character retrieval</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="Hello world"
1010 B$=MID$(A$,4,2)
1020 PRINT B$
Ready
>RUN
lo
Ready
Citizen BASIC Interpreter

<table>
<thead>
<tr>
<th>Text$</th>
<th>String type</th>
<th>String intended for character retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Integer type</td>
<td>Number of bytes for character retrieval</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="Hello world"
1010 B$=RIGHT$(A$,5)
1020 PRINT B$
Ready
>RUN
world
Ready

3.17.3 LEFT$ Function

【Explanation】
The LEFT$ function returns the partial string of the designated length that is displayed at the beginning of the string. It returns the number of characters within the number of characters designated by Count from the beginning of Text$.

【Format】
RIGHT$(Text$,Count)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String intended for character retrieval</td>
</tr>
<tr>
<td>Count</td>
<td>Integer type</td>
<td>Number of bytes for character retrieval</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="Hello world"
1010 B$=LEFT$(A$,5)
1020 PRINT B$
Ready
>RUN
Hello
Ready

3.17.4 STRING$ Function

【Explanation】
The STRING$ function returns the string made by repeating strings the designated number of times. The function returns the string made by repeating the string designated by Text$ Count times. The following shows an example:

S$=STRING$(10, "A");
In this case, S$ is set to the string AAAAAAAAAAA.

【Format】
STRING$(Count,Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Integer type</td>
<td>Number of times of repetition</td>
</tr>
<tr>
<td>Text$</td>
<td>String type</td>
<td>Characters intended for repetition</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 PRINT STRING$(10,"ABC")
Ready
3.17.5 CHR$ Function

【Explanation】
The CHR$ function returns, as the string of a character, the character corresponding to the designated ASCII value.

【Format】
CHR$(X)$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>ASCII code value to convert to string</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 OPEN #1:NAME"SER"
1010 DO
1020 A$=INPUT$(1,#1)
1030 LOOP UNTIL A$=CHR$(27)
1040 CLOSE #1
Ready
>RUN
(ESC key input)
Ready

3.17.6 STR$ Function

【Explanation】
The STR$ function converts the integer to the string.

【Format】
STR$(X)$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Integer value to convert to string</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="TEST"+STR$(123)
1010 PRINT A$
Ready
>RUN
TEST123
Ready

3.17.7 SPACE$ Function

【Explanation】
The SPACE$ function returns the designated number of spaces as the string.

【Format】
SPACE$(X)$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
</table>
3.17.8 MKI$ Function

**[Explanation]**
This function converts the integer value, as 16 bit of the integer value, to 2 byte of a string. The lower byte of the integer value is stored to the first byte while the upper byte of the integer value is stored to the second byte.

**[Format]**
MKI$(X)$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Integer value to convert to string</td>
</tr>
</tbody>
</table>

**[E.g.]**
>LIST
1000 A$="A"+SPACE$(10)+"B"
1010 PRINT A$
Ready
>RUN
A          B
Ready

3.17.9 DATE$ Function

**[Explanation]**
This function returns the date the clock of the printer shows as the following format of the strings:
YYYYMMDD
YYYY: Year
MM: Month
DD: Date

However, when the printer is not equipped with the clock, the DATE$ function does not return the accurate year, month and date.

**[Format]**
DATE$

**[E.g.]**
>PRINT DATE$
20000101
3.17.10 TIMES$ Function

[Explanation]
This function returns the time the clock of the printer shows as the following format of the string:
HH:MM:SS
HH: Hour
MM: Minute
SS: Second

However, when the printer is not equipped with the clock, the TIMES$ function does not return the accurate time.

[Format]
TIMES$

[E.g.]
Ready
>PRINT TIMES$
00:00:00
Ready

3.17.11 EXTRACT$ Function

[Explanation]
This function extracts the string located between two designated strings to return it as a string. It returns the string from the Text$ string immediately after the one corresponding to Begin$ to the string immediately before the one corresponding to End$.

[Format]
EXTRACT$(Text$,Begin$,End$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>Intended string to extract string</td>
</tr>
<tr>
<td>Begin$</td>
<td>String type</td>
<td>String immediately before extracted string</td>
</tr>
<tr>
<td>End$</td>
<td>String type</td>
<td>String immediately after extracted string</td>
</tr>
</tbody>
</table>

[E.g.]
1000 A$="The quick brown fox jumps over the lazy dog"
1010 B$=EXTRACT$(A$,"The "," jumps")
1020 PRINT B$
Ready
>RUN
quick brown fox
Ready

3.17.12 LCASE$ Function

[Explanation]
The LCASE$ function converts the ASCII string to small letters. It also converts all strings passed in Text$ to small letters to return. The only characters convertible are 7 bit of ASCII characters from “A” to “Z”.

[Format]
LCASE$(Text$)
## 3.17.13 UCASE$ Function

**Explanation**
The UCASE$ function converts the ASCII string to capital letters. It also converts all strings passed in Text$ to capital letters to return. The only characters convertible are 7 bit of ASCII characters from “A” to “Z”.

**Format**

```
UCASE$(Text$)
```

### Example

```
>LIST
1000 A$="The quick brown fox jumps over the lazy dog"
1010 B$=UCASE$(A$)
1020 PRINT B$
Ready
>RUN
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ready
```

## 3.17.14 LTRIM$ Function

**Explanation**
The LTRIM$ function removes the leading space and control characters from the string.

**Format**

```
LTRIM$(Text$)
```

### Example

```
>LIST
1000 A$="   Hello world"
1010 B$=LTRIM$(A$)
1020 PRINT B$
Ready
>RUN
HELLO WORLD
Ready
```
3.17.15 RTRIM$ Function

【Explanation】
The RTRIM$ function removes the trailing space and control characters from the string.

【Format】
RTRIM$(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to remove the trailing space</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="Hello world   
1010 B$=RTRIM$(A$)
1020 PRINT B$;"*"
Ready
>RUN
Hello world*
Ready

3.17.16 HEX$ Function

【Explanation】
The HEX$ function returns hexadecimal expression of an integer with a string.

【Format】
HEX$(X)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Numeric value to convert to the hexadecimal string</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A=&H1234
1010 PRINT HEX$(A)
Ready
>RUN
1234
Ready

3.17.17 OCT$ Function

【Explanation】
The OCT$ function returns octal expression of an integer with a string.

【Format】
OCT$(X)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
</table>
3.17.18 REPEAT$ Function

【Explanation】
The REPEAT$ function returns the string made by repeating a string designated number of times. It also returns the string made by repeating a string designated by Text$ Count times. An example is: S$=REPEAT$("A", 10); in this case, S$ is set to the string AAAAAAAAAA.

【Format】
REPEAT$(Text$, Count)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to repeat</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Number of times to repeat</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$=REPEAT$("Hello", 3)
1010 PRINT A$
Ready
>RUN
HelloHelloHello
Ready

3.18 Numeric Value Function

3.18.1 MOD Function

【Explanation】
This function sets up residues. It returns the residue of Y for X. The return value is XmodY.

【Format】
MOD(X, Y)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Dividend</td>
</tr>
<tr>
<td>Y</td>
<td>Integer type</td>
<td>Divisor</td>
</tr>
</tbody>
</table>

【E.g.】
Ready
>PRINT MOD(99, 5)
4
Ready
Citizen BASIC Interpreter

3.18.2 ORD Function

【Explanation】
This function digitalizes and returns the leading 1 byte of the character code of the string.

【Format】ORD(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to digitalize</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="ABC"
1010 B=ORD(A$)
1020 PRINT B
Ready
>RUN
65
Ready

3.18.3 LEN Function

【Explanation】
This function returns the length of the string.

【Format】LEN(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to calculate length</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 LINPUT A$
1010 PRINT "Length=";LEN(A$)
Ready
>RUN
?1234567890
Length=10
Ready

3.18.4 VAL Function

【Explanation】
The VAL function converts the string standing for an integer to the numeric value.

【Format】VAL(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to digitalize</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="12345"
1010  B=VAL(A$)
1020  PRINT B
Ready
>RUN
12345
Ready

3.18.5 ASC Function

【Explanation】
This function digitalizes and returns the leading 1 byte of the character code of the string.

【Format】
ASC(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to digitalize</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000  A$="ABC"
1010  B=ASC(A$)
1020  PRINT B
Ready
>RUN
65
Ready
3.18.6 INSTR Function

【Explanation】
The INSTR function returns the index value of the first character of the specific partial string found in the designated string. It seeks the partial string Substr$ from the string Text$. Substr$ and Text$ are the formulae of the stringer-type. Seeking Substr$ from Text$, the INSTR function returns the integer value standing for the index of the first character of Substr$. In searching the string by this function, capital letters and small letters are distinguished. If Substr$ is not found, the INSTR function returns 0.

【Format】
INSTR(Text$,Substr$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>Intended string to search string</td>
</tr>
<tr>
<td>Substr$</td>
<td>String type</td>
<td>String to search</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A$="The quick brown fox jumps over the lazy dog"
1010 B=INSTR(A$,"quick")
1020 PRINT B
Ready
>RUN
5
Ready

3.18.7 MKI Function

【Explanation】
The MKI function converts the leading 2 byte of the string to 16 bit of the integer value. The first byte of the string shall be the lower byte of 16 bit of the integer value while the second byte of the string is the upper byte of 16 bit of the integer value.

【Format】
MKI(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>String to convert to integer value</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A=1234
1010 B$=MKI$(A)
1020 C=MKI(B$)
1030 PRINT C
Ready
>RUN
1234
Ready

3.18.8 DATE Function

【Explanation】
This function returns the date the clock of the printer shows as the following format of the integer value:
YYYYDDD
YYYY: Year
DDD: Lapsed days from 1 January

However, when the printer is not equipped with the clock, the DATE function does not return the accurate year, month and date.

**Format**
DATE

**E.g.**
>PRINT DATE
2000001
Ready

### 3.18.9 TIME Function

**Explanation**
This function returns the lapsed time from 0 o’clock a.m. the clock of the printer shows by seconds as the numeric value. However, when the printer is not equipped with the clock, the DATE function does not return the accurate year, month and date.

**Format**
TIME

**E.g.**
>PRINT TIME
0
Ready

### 3.18.10 MAX Function

**Explanation**
This function returns the greater figure of the two.

**Format**
MAX(X,Y)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Comparative figure 1</td>
</tr>
<tr>
<td>Y</td>
<td>Integer type</td>
<td>Comparative figure 2</td>
</tr>
</tbody>
</table>

**E.g.**
>LIST
1000 A=123
1010 B=456
1020 PRINT MAX(A, B)
Ready
>RUN
456
Ready

### 3.18.11 MIN Function

**Explanation**
This function returns the smaller figure of the two.

**Format**
MIN(X,Y)
### Parameter Data type Meaning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Integer type</td>
<td>Comparative figure 1</td>
</tr>
<tr>
<td>Y</td>
<td>Integer type</td>
<td>Comparative figure 2</td>
</tr>
</tbody>
</table>

【E.g.】

> LIST
1000 A=123
1010 B=456
1020 PRINT MIN(A, B)
Ready
> RUN
123
Ready

### 3.18.12 MAXLEN Function

【Explanation】
This function returns the maximum number of the stored characters of a string variable. The value is fixed at 255.

【Format】
MAXLEN(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$:</td>
<td>String variable</td>
<td>String variable to calculate the maximum number of stored characters</td>
</tr>
</tbody>
</table>

【E.g.】

Ready
> PRINT MAXLEN(A$)
255
Ready

### 3.18.13 MAXNUM Function

【Explanation】
This function returns the maximum value of an integer type variable. The value is fixed at 2147483647.

【E.g.】

> PRINT MAXNUM
2147483647
Ready

### 3.19 Input-Output Function

#### 3.19.1 INPUT$ Function

【Explanation】
This function inputs the data of the byte length designated by the designated device to return it as the string.

【Format】
INPUT$(Length[,Device])
### Parameter Data type | Explanation
---|---
Length | Integer | Designate the byte length to input by bytes.
Device | Channel number | Designate in \#n format the channel number of the device to input. When omitted, input from the console.

#### 3.19.2 GET$ Function

**Explanation**
This function inputs 1 byte of data from the designated device to return it as the string.

**Format**
GET$(Device)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Channel number</td>
<td>Designate in #n format the channel number of the device. Designation of 0 inputs data from the console.</td>
</tr>
</tbody>
</table>

**E.g.**
>LIST
1000 OPEN #1:NAME "SER"
1010 DO
1020 A$=INPUT$(1,#1)
1030 IF A$=CHR$(27) THEN EXIT
1040 PRINT A$;
1050 LOOP 1
1070 CLOSE #1
Ready
>RUN
(input 1234567890 ESC)
1234567890
Ready

#### 3.19.3 EOF Function

**Explanation**
This function judges the end of a file by file input. However, since the BASIC interpreter does not support file input-output, this function is unable to be used.

**Format**
3.19.4 INKEY$ Function

【Explanation】
This function inputs 1 byte from the console.

【Format】
INKEY$

【E.g.】
>LIST
1000 DO
1010 A$=INKEY$
1020 IF A$=CHR$(27) THEN EXIT
1021 PRINT A$;
1030 LOOP WHILE 1
Ready
>RUN
(input 1234567890 ESC)
1234567890
Ready

3.19.5 SEARCHTO$ Function

【Explanation】
This function inputs data from the designated device to search the string. The detected string recovers from the function to return such string as a result.

【Format】
① SEARCHTO$(SourceDevice, SearchText$)
② SEARCHTO$(SourceDevice, SearchText$, DestDevice)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceDevice</td>
<td>Channel number</td>
<td>Channel number of the device for data input</td>
</tr>
<tr>
<td>SearchText$</td>
<td>String-type array</td>
<td>Designate the string to search. For several strings to search, designate string array.</td>
</tr>
<tr>
<td>DestDevice</td>
<td>Channel number</td>
<td>Channel number of the device for data output</td>
</tr>
</tbody>
</table>

① inputs data from SourceDevice to seek the string corresponding to SearchText$.
② inputs data from SourceDevice to seek the string corresponding to SearchText$.
Further, the input data not corresponding to the search character are output to DestDevice.
When there are several strings to search, designate the string array to SearchText$.

【E.g.】
>LIST
1000 OPEN #1:NAME "SER"
1010 A$=SEARCHTO$(1,"ABC",1)
1020 PRINT #1,
1030 PRINT #1,A$
1040 CLOSE #1
Ready
>RUN
3.19.6 ISERROR Function

【Explanation】
This function returns the error condition of the printer.

【Format】
ISERROR

【Return value】
0: Normal
1: Error condition

【E.g.】
>SETERR
Ready
>PRINT ISERROR
1
Ready
>CLRERR
Ready
>PRINT ISERROR
0
Ready

3.19.7 ISWARNING Function

【Explanation】
This function returns the warning error condition of the printer.

【Format】
ISWARNING

【Return value】
0: Normal
1: Warning error condition

3.19.8 DATAREADY Function

【Explanation】
This function inspects whether data are ready in the device. When the data already exist in the designated device, the return value is 1. If not, the return value is 0.

【Format】
DATAREADY(Device)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Channel number</td>
<td>Designate the channel number of the device to inspect whether data are ready</td>
</tr>
</tbody>
</table>

【Return value.】
0: Data not ready in the device
1: Data ready in the device
3.20 Error Handling Function

3.20.1 ERL Function

【Explanation】
This function returns the line number where an error occurred. The ERL function is used in the error handling routine defined by the ONERRORGOTO statement.

【Format】
ERL

【E.g.】
>LIST
1000 ON ERROR GOTO 9000
1010 A=1000/0
1020 END
9000 PRINT "error on ";ERL
9010 END
Ready
>RUN
error on 1010
Ready

3.20.2 ERR Function

【Explanation】
When an error occurs, the ERR function obtains the error code. It is used in the error handling routine defined in the ONERRORGOTO statement.

【Format】
ERR

【E.g.】
>LIST
1000 ON ERROR GOTO 9000
Citizen BASIC Interpreter

1010 A=1000/0
1020 END
9000 PRINT "error on ";ERL
9001 PRINT "error code ";ERR
9010 END
Ready
>RUN
error on 1010
error code 25
Ready

3.20.3 ERMG$ Function

【Explanation】
This function obtains the content of the error message corresponding to the error code.

【Format】
ERMSG$(Errcode)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errcode</td>
<td>Integer type</td>
<td>Error code to obtain error message</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 ON ERROR GOTO 9000
1010 A=1000/0
1020 END
9000 PRINT "error on ";ERL
9001 PRINT "error code ";ERR
9002 PRINT "error message ";ERMSG$(ERR)
9010 END
Ready
>RUN
error on 1010
error code 25
error message Zero divisor error
Ready

3.21 Mathematical Function

Since the mathematical function handles the floating point, use it in the state of ZBICOMPATI OFF.

3.21.1 SGN Function

【Explanation】
This function obtains the sign of the figure X.

【Format】
SGN(X)

【Return value】
-1   X is negative
 0   X = 0
 1   X is positive
Citizen BASIC Interpreter

3.21.2 ABS Function

【Explanation】
This function returns the absolute value of the figure X.

【Format】
ABS(X)

3.21.3 SQR Function

【Explanation】
This function returns the square value of X.

【Format】
SQR(X)

3.21.4 LOG Function

【Explanation】
This function calculates the natural logarithm of X.

【Format】
log(X)

3.21.5 EXP Function

【Explanation】
This function calculates Xth power of the exponential function e.

【Format】
exp(X)

3.21.6 EVAL Function

【Explanation】
This function evaluates the formula given by Text$.

【Format】
EVAL(Text$)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text$</td>
<td>String type</td>
<td>Give evaluation formula by string</td>
</tr>
</tbody>
</table>

【E.g.】
>LIST
1000 A=12
1010 B=23
1020 PRINT EVAL("A+B")
Ready
>RUN
35
Ready

3.21.7 INT Function

【Explanation】
This function takes out the integral part of the figure X.
Citizen BASIC Interpreter

3.21.8 SIN Function

[Explanation]
The SIN function calculates the sine and that of the input value. Its angle is designated by radian unit.

[Format]
SIN(X)

3.21.9 COS Function

[Explanation]
The COS function calculates the cosine and that of the input value. Its angle is designated by radian unit.

[Format]
COS(X)

3.21.10 ATAN Function

[Explanation]
The ATAN function calculates the arctangent and that of the input value.

[Format]
ATAN(X)

3.21.11 RND Function

[Explanation] The RND function generates the random value, and returns pseudo random numbers between 0 and X every time it is called.

[Format]
RND(X)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Floating point</td>
<td>Positive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative</td>
</tr>
</tbody>
</table>

Return pseudo random numbers between 0 and X
Return equivalence returned by last call
Initialize random number generator with the absolute value of X to return the pseudo random number between 0 and X

3.21.12 PI Function

[Explanation]
This function returns the value of pi (π).

[Format]
PI