https://ntrs.nasa.gov/search.jsp?R=19800002568 2020-03-21T20:19:06+00:00Z

N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE

0

NASA TECHNICAL MEMORANDUM NASA TM-75680

BEYSIK: LANGUAGE DESCRIPTION AND HANDBOOK FOR PROGRAMMERS (SYSTEM FOR THE COLLECTIVE USE OF THE INSTITUTE OF SPACE RESEARCH, ACADEMY OF SCIENCES USSR)

I. G. Orlov

Translation of "BEYSIK. Opisaniye Yazika i Rukovodstvo dlya Programmista (Sistema kollektivnogo pol'zovaniya IKI AN SSSR)," Academy of Sciences USSR, Institute of Space Research, Moscow, Report Pr-476, 1979, pp. 1-42.

(NASA-TM-75680) BEYSIK: LANGUAGE DESCRIPTION AND HANDBOOK FOR PROGRAMMERS (SYSTEM FOR THE COLLECTIVE USE OF THE INSTITUTE OF SPACE RESEARCH, ACADEMY OF SCIENCES (National Aeronautics and Space **N80-10813**

Hc A03/MF A01

Unclas

G3/61 45890

NOV 1979
RECEIVED
NASA STI FACILITY
ACCESS DEPT.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546 OCTOBER 1979

TABLE OF CONTENTS

/41

DING PAGE BLANK NOT FILMED

[illegible]DING PAGE BLANK NOT FILMED

BASIC

Description of Language and Programmer Guide

I. G. Orlov

The BASIC algorithmic language is described and a guide is presented for the programmer using the language interpreter. BASIC is a component of the display systems developed by personnel of the Systems Programming Laboratory of the Institute of Space Studies of the AS USSR.

1. Introduction /3*

The high-level algorithm language BASIC is a problem-oriented programming language intended for the solution of computational and engineering problems.

1.1 Brief Description of the Language

A fundamental feature of BASIC is operation in the dialog regime, i.e., the programmer can correct and debug the program directly from the console.

A program written in BASIC consists of statements, each of which occupies one line. The line length does not exceed 64 symbols. The statements are divided into BASIC commands and BASIC operators.

The commands are used to establish the program execution regimes, to print the program text, and to alter the translation and interpretation regimes.

*Numbers in margin indicate pagination of original foreign text.

The operators form the executable part of the program. They can be introduced both in the direct regime and in the sequential interpretation regime.

The operators introduced in the direct regime do not have line numbers. They are performed immediately after input. A list of the operators which can be executed in the direct regime is presented in the corresponding section.

The operators introduced in the sequential interpretation regime have numbers from 1 to 9999 and are executed in increasing line number order. Any operator can be introduced in this regime and the entry order need not be strictly sequential. After entry, the operators are sorted in increasing number order (for simplicity and convenience of addition, it is recommended that the operators be introduced with step 10). For replacement of an operator, the user must introduce a new operator with the same number. For removal of an operator introduced in the program regime, we need only introduce its number separately. /4

Examples

```
1Ø    A = 1                        Operator 1Ø   A==Ø replaces
2Ø    0 = 2                        Operator 1Ø   A=I
1Ø    A = Ø
3Ø    B = Ø
4Ø    D = B + 1                    Operator 30   B=Ø will be
3Ø                                                   removed
```

After entry in the sequential interpretation regime, the operators are checked for syntactic correctness and are converted to an intermediate form in which they are stored in the computer operative memory. The basic form of such an operator is stored in the direct-access working file and can be printed out at any moment with the aid of the corresponding command. All (or part) of the operators introduced in the sequential interpretation regime can be catalogued in the library of basic modules for later use.

When entering a program in BASIC, the user must remember that all the blanks in the text (other than the blanks enclosed in inverted commas or quotes) are ignored.

1.2 <u>BASIC Symbols</u> /5

The following symbols are used when writing programs in the BASIC algorithmic language:

a) 26 Latin letters: A, B, C, ..., X, Y, Z;

b) 10 Arabic numerals: Ø,1,2,3,4,5,6,7,8,9;

c) special signs:

	space	≠	not equal to
=	equal to	'	inverted comma
+	plus	;	semicolon
-	minus		
*	asterisk		
/	slash		
(	open paren		
)	close paren		
,	comma		
.	period		
>	greater than		
<	less than		
¬	negation		

In addition, if the input and output devices use the signs:

] right bracket
[left bracket
" quotation marks

then the following signs can also be used as symbols:

) right paren
(left paren
' inverted comma

respectively.

The remaining symbols of the alphanumeric set of any specific input or output device may appear between paired inverted commas (quotes) or in the language operator REM. /6

1.3 Characteristic Features of BASIC Writing

The modified Backus form is used in writing BASIC languate.

The syntactic elements in the definitions of the commands and operators are enclosed in angle brackets: "<" and ">". Optional elements are enclosed in square brackets "[" and "]".

Example:

LET< defined variable >=[< defined variable >]=expression

The second element [< defined variable >] is optional.

In case of repetition of one or more of the syntactic elements in the definitions, ellipses . . . can be used. In case of selection from several possibilities, braces "{" and "}" can be used.

Example:

```
CLEAR {[<line number>], [<line number>]}
```

In this command, we can use either the optional operand V or operands in the form

```
[<line number>], [<line number>]
```

The square brackets indicate the optional nature of both the first and second operands.

1.4 Objects Used by the BASIC Algorithmic Language /7

The following objects can be used in the algorithmic language program:

a) numerical constants,
b) symbolic constants,
c) one-dimensional and two-dimensional arrays,
d) variables,
e) standard functions,
f) user functions.

By (numerical) constant is meant any decimal number, written with or without sign, with or without decimal point, with or without exponent. If a number is followed by the letter E, possibly followed by a sign and one or two decimal numerals, this means that the number is to be multiplied by the corresponding power of 10.

Examples of numerical constants:

- Ø.12E + 8	= $-0.12 \cdot 10^{8}$	.ØØ3	= 0,003
1	= 1,0	1.	= 1.0
2.87E3	= $2,87 \cdot 10^{3}$	3E-1	= $3,0 \cdot 10^{-1}$

Any number specified explicitly in the BASIC program is a constant. Any set of symbols enclosed in inverted commas or quotes which in the given case is not a part of a constant is termed a symbolic constant. If the user wishes to make an inverted comma a part of a symbolic constant, he must repeat the inverted comma.

Examples of symbolic constants:

```
'A␣B␣CDEF'
'A " B'
```

A variable in BASIC is a quantity which can alter its value in the computational process. The name of the variable is denoted by a Latin letter or by a letter and numeral. /8

Examples of names of variables:

```
A,Z.U1;DØ:E9
```

The first operator in which the variable is used must assign it some value. A variable whose value has not been defined cannot be used. In this case, an error message is generated.

In BASIC an ensemble of like quantities combined under a single name is termed an array. One-dimensional and two-dimensional arrays are permitted. Since the array name is denoted by a Latin letter (and there are 26), no more than 26 arrays can be used in BASIC. The array elements are termed indexed variables. Arrays are identified either by the operator DIM or by implication. The index is written in parentheses after the array name A(7,6), B(2). The following rules must be followed then using indexed variables:

1. Array indexing always begins with zero, thus the first

element of the one-dimensional array A will be A(Ø), while that of the two-dimensional array B will be B(Ø,Ø).

2. The maximal value of the index cannot exceed 3210241, but the array dimension may be limited by the available computer memory volume.

3. The maximal value of the indexes for the arrays is defined by the operator DIM, for arrays defined by implication this value is equal to 1Ø (for one-dimensional arrays) or (1Ø,1Ø) for two-dimensional arrays.

4. Use in the program of the same names for indexed and nonindexed variables is permitted. However, one-dimensional and two-dimensional arrays cannot have the same names. /9

5. An expression can be used as an array element index. The result of calculation of the expression is rounded to the closest integer.

Initially, all the array elements contain the maximal in modulus negative number. Therefore, use of an undefined indexed variable leads to an error.

1.5 Expressions of the BASIC Algorithmic Language

The BASIC algorithmic language admits arithmetic expressions, which are used for the calculation of some value. The expression is a complete entry indicating which quantities are to be taken and what operations are to be performed on them in order to calculate this value. The value of the arithmetic expression is a real number. The simplest arithmetic expression consists of an elementary expression, which is:

a) a constant,
b) a simple variable,
c) an indexed variable,
d) referral to a function,
e) an expression enclosed in parentheses.

More complex expressions can be formed from the elementary expression by use of arithmetic operations. The following arithmetic operations are admissible in BASIC:

a) addition (+)
b) subtraction (-)
c) multiplication (*)
d) division (/) /10
e) exponentiation (**)

The sequence of performance of the mathematical operations coincides with the sequence used in mathematics. The use of functions is permitted in BASIC. Reference to a function has the form:

<function name>(<argument>, <argument>,)

The function name consists of three letters. We differentiate two classes of functions: user functions and standard functions. The user function is defined in the operator DEF and its name has the form:

FN<letter>

1.6 BASIC Standard Functions

Ten standard functions are used in the BASIC algorithmic language: SIN, COS, TAN, ATN, LOG, EXP, INT, ABS, SQR, RND;

only a single argument is used for all the standard functions. The arguments of the trigonometric functions SIN(x), COS(x), TAN(x) are specified in radians; the result of calculation of the function ATN(x) is the principal value of the arctangent in radians; the function LOG(x) is used to calculate the natural logarithm; and the function (EXP(x) is used to calculate the exponential function. The function INT(x) is used to calculate the whole part of the argument, i.e., INT(3.7)=3; INT(2.7)= -3; INT(Ø)=Ø. The function RND(x) is used to generate pseudo-random numbers in the limits from Ø to 1, and the function ABS(x) is used to calculate the absolute magnitude of the argument. It should be noted that in BASIC there is no difference between whole and real numbers. In the computer memory, all numbers are numbers with floating decimal point.

2. BASIC Commands /11

2.1 RUN Command

The program located in the operative memory begins to be executed on the RUN command -- only the operators introduced in the sequential interpretation regime are executed. The command has the format:

```
RUN<line number>.
```

The memory distributed during the preceding execution of the program (arrays defined by implication and simple variables) is cleared by the RUN command without a line number, and an indefinite value is assigned to all the array elements defined explicitly. Execution of the program begins with the operator having the smallest number. On the command RUN with a line number, the program is executed, beginning with the selected line. The variables and the array elements retain the values obtained during the last execution of the program. After termination of execution of the program, the basic text and values of all the variables are retained in the memory.

2.2 SELECT PRINT Command

The SELECT PRINT command is used for printout of the text and results of execution of the program. The command format is: SELECT PRINT.

As a result of performance of this command, all the lines output to the terminal are stored in the working file of the system. Upon completion of operation with BASIC, the stored lines are printed out. Operation of the SELECT PRINT command is terminated upon performance of the STOP operator, and also /12 upon performance of any operator in the direct regime. However, the information previously stored in the working file is retained. Upon entry of a new SELECT PRINT command, the new information supplement that information already stored in the working file.

If a printout device is not available to the system, the first BASIC operator introduced after the SELECT PRINT command calls up an error message and the SELECT PRINT regime will be terminated.

If the user wishes to print out the text of his program, it is recommended that the LIST operator be introduced after the SELECT PRINT operator. In this case the program text output to the terminal will be stored in the system working file and printed out later.

2.3 CLEAR Command

This command is used to erase the program and the values of the variables and the arrays from the operative memory.

The command has the format:

$$\text{CLEAR}\left\{\overset{(V)}{\text{<line number>, <line number>}}\right\}$$

Use of the CLEAR command without parameters erases from the operative memory the entire program and the values of the variables and the arrays. CLEAR V assigns to all the variables and the array elements indefinite values; the memory assigned to arrays defined by implication is cleared. The CLEAR command with indication of the line numbers removes the part of the program text located between these numbers (including the operators with the given numbers).

Examples: /13

CLEAR	clear memory of program and variables
CLEAR 1,9999	removes all the program text while retaining the values of the variables
CLEAR 7,81	removes from the memory operators with numbers from 7 to 81
CLEAR V	clears the memory assigned to the arrays defined by implication and assigns an indeterminate value to all elements of the explicitly defined arrays and variables

2.4 CONTINUE Command

This command is used to renew operation of a program interrupted by the user or as a result of occurrence of an error in the program execution stage. The command has the format:

CONTINUE

2.5 LIST Command

This command makes it possible to output the program text located in the memory in the line number sequence. The command has the following possible formats:

$$\text{LIST}\left\{\begin{matrix}\$\\ \langle\text{line number}\rangle[,\langle\text{line number}\rangle]\end{matrix}\right\}$$

When using the LISTS format, the next 15 program lines are output, after which the program stops. When using the LIST command with indication of a single line number, only /14
this line is read out. When using two line numbers, the lines located between these numbers (including the specified numbers) are output.

Examples:

LISTS	The first 15 operators are printed out; the second LISTS command prints out the next group of 15 operators, etc.
LIST 1Ø,18	The text of all operators having numbers from 10 to 18 is printed out.

2.6 LOAD Command

The LOAD command is used to load the program or part of the program into the operative memory from the basic text library. The command format is:

LOAD[P] '<Name>'

With the presence in the LOAD command of the parameter P, the program text and the values of the variables and the arrays located in the operative memory remain unchanged.

In the absence of the parameter P, the operation of the CLEAR command without parameters is modeled before loading the program text from the library, i.e., the program text located in the memory is removed and the memory assigned to the arrays is cleared; the variables take indeterminate values. A name enclosed in inverted commas can contain no more than eight symbols. The program with such a name is sought initially in the individual library and then in the systems library of basic texts in the sublibrary B.

Example: /15

LOAD 'TESTMAT'	Clearing of the memory and loading of the book B.TESTMAT are accomplished

2.7 SAVE Command

This command is used to catalog (enter) in the basic module library the program text located in the operative memory. The command has the format:

```
SAVE 'name' [[,<line number>],<line number>]
```

< Name > -- a line of no more than eight symbols, defining the book name in the library of basic modules.

The first line number indicates which line of the program is to be cataloged in the basic module library. If both line numbers are omitted, the entire program text is cataloged.

Cataloging of the program is possible only into the individual basic text library -- into the sublibrary B. The use of special symbols in the name is not recommended, since books with such names cannot always be processed by the systems program LIBRARIAN. The program cataloged using the SAVE command can later be used with the aid of the LOAD command. The user must remember that only operators introduced which can be in the sequential interpretation regime can be cataloged.

Examples:

SAVE 'TESTMAT'	The entire program text is cataloged under the name B.TESTMAT;
SAVE'TESTINV1',17Ø	The operators beginning with number 17Ø are cataloged under the name B.TESTINV1.

/16

2.8 TRACE Command

The TRACE command is used to establish or cancel the operator tracing regime. The command format is:

$$\text{TRACE} \begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix}$$

The TRACE command with the parameter ON establishes the tracing regime; when executed the TRACE message nnnn is printed for each operator,

where nnnn -- number of the executed line.

The TRACE command with the parameter OFF cancels the tracing regime.

2.9 RENUMBER Command

This command is used to renumber the program operators located in the memory. The command format is:

RENUMBER [<line number>] [,< integer >]

The first parameter defines the new number being assigned to the first program operator. The second parameter (integer) indicates the renumbering step. If the renumbering step is omitted, it is taken equal to 10. If the first parameter is omitted, it is taken equal to the step. The line numbers in the operators ON, GOTO, GOSUB are renumbered automatically.

Example: /17

RENUMBER 25,5	Renumber the program with step 5, the new number of the first operator is 25.

2.10 END Command

On this command the BASIC interpreter terminates its operation.

3. BASIC Operators

The BASIC operators are divided into executable and non-executable. The executable operators indicate the sequence of operations to be performed by the interpreter. The nonexecutable operators introduce information which the system requires for operation or information which makes the program more easily visualized. These operators reserve memory for the arrays (DIM operator), define the use functions (DEF operator), define the data which during execution of the program can be assigned to the simple and indexed variables (DATA operator). The commentary operator REM is used only to make the program more convenient for reading. The nonexecutable operators are processed by the system in the compilation stage; while in the interpretation stage they are dummy operators. The position of the nonexecutable operators in the program is immaterial; however, the user must remember that in case of redefinition of an array or function the redefinition operator must have the same number as the first-definition operator.

Example: /18

```
10 DIM A(3,4)
20 DIM A(5,6)
BAS024 ERROR IN LINE 20
10 DIM A(5,6)
```

Array A of dimension (Ø:3; Ø4) is defined; improper attempt to redefine the array and transfer it to the large memory; redefinition of the array reserves for the array A of dimension (Ø:5;Ø:6) a memory of (5+1)(6+1) machine words.

3.1 Assignment Operator LET

This operator is used to assign to variables the values of an arithmetic expression or a constant. The command has the format:

```
      variable being  variable being
[LET]< defined      >=[< defined      >=]...<expression>
```

The LET operator calculates the value of the arithmetic expression located to the right of the rightmost equality sign in the line and assigns the result of calculation of the expression to all the indexed variables.

The user must remember that the indexes of the variables being defined are calculated before executing the first assignment.

Example:

```
1ØØ A(K,L)=K=L+1Ø. *SIN(3)
```

The indexes of the array A(K,L) are calculated before obtaining the new value of K.

The operator LET can be executed both in the direct regime and in the sequential interpretation regime.

Example: /19

```
4Ø LET A = 1 + SIN (3.14/LOG(X)) * EXP(4);
7Ø B(7,K+3) = E(Ø) = T = 1+A+FNA (EXP(Z)).
```

3.2 Transfer Operator GOTO

The GOTO operator is used to change the normal program execution sequence. The operator has the format:

```
GOTO <line number>
```

The GOTO operator accomplishes unconditional transfer of control to the line, the number of which is specified in the operator. The GOTO operator cannot be executed in the direct regime.

Example:

```
1Ø A = 1
2Ø GOTO 5Ø
3Ø X= SIN(A)
4Ø B = A/2
5Ø R1 = A** 2
```

In this example a program fragment is executed not in the increasing number order but rather in the order: 1Ø; 2Ø; 5Ø:

3.3 Conditional Transfer Operator IF

The IF operator is used to alter the order of execution of the operators in the program as a function of the results of comparison. The operator has the format:

```
IF <comparison> THEN [GOTO] <line number>,
```

where.

```
<comparison> ::= <arithmetic expression><relational operator><arithmetic expression>
<relational operator> :: = >|<|>=|=>|<=|=<|<>|><|¬=|=
```

The relational operators denote: /20

>	greater than,
<	less than,
>=, =>	greater than or equal to,
<=, =<	less than or equal to,
<>, ><	not equal to.

If the comparison is true transfer takes place to execution of the line with the number following THEN or GOTO. The word GOTO is an optional element and does not influence execution of the operator. The IF operator cannot be executed in the direct regime.

3.4 Cycle Organization Operators FOR and NEXT

The FOR and NEXT operators are used to organize program cycles. FOR defines cycle initiation, NEXT defines the end. The operators have the formats:

```
FOR<control variable>=expression1>TO<expression 2>
                              [STEP<expression 3>]
NEXT<control variable>
```

The control variable is a nonindexed variable with values varying from expression 1 to expression 2 with step equal to expression 3 .

The operators located between the operators FOR and n NEXT are executed as many times as the FOR operator indicates. If the phrase STEP expression 3 in the FOR operator is omitted, the step is taken equal to 1.

Cycle examples: /21

```
10 FOR I=1 TO 10 STEP 3
20 A(I)=3.14*I/2
30 NEXT I
```

The cycle consisting of a single operator with number 2Ø is repeated four times for 1 = 1, 4, 7, 1Ø the cycle is performed four times.

```
10 FOR I=1 TO 10 STEP 3
20 IF I >5  THEN 50
30 A(I) = SIN(3.14*1/8)
40 NEXT I
45 GOTO 80
50 A(I) = -SIN (3.14*1/8)
60 NEXT I
80 REM
```

For 1 = 1, 1 = 4 the cycle of operators 2Ø, 3Ø, 4Ø is performed.
For 1 = 7, 1 = 10 the cycle of operators 2Ø, 5Ø, 6Ø is performed.

The last example shows that NEXT must logically follow the operator FOR.

It should be noted that the cycle may be terminated by an operator FOR in which the name of the control variable coincides with the name of the control variable of the uncompleted cycle.

Example:

```
10 FOR I = 1TO3
20A(I)=COS(3.14*I/8)
30FORI=7TO9
4 0 T (I) = T(I)+I/2
5 0 N E X T I
```

The operators are executed in the following order: 1Ø, 2Ø, 3Ø, 4Ø, 5Ø, 4Ø, 5Ø, 4Ø, 5Ø.

Nesting of the FOR cycles is permitted. The maximal cycle nesting level depends on the available memory size but must not exceed 12.

Example:

```
10 FOR I=0 TO 3
20 FOR J=0 TO 1+1
30 C(I,J)=I+SIN(J*12)
40 N E X T J
50 NEXTI
```

If necessary, the NEXT operator of the outer cycle may terminate the inner cycle. /22

Example:

```
10 FOR I=1 TO 3
20 FOR J=2 TO 4
30 IF I> =J THEN 50
35 C(I,J)=0.
40 NEXT J
50 NEXT I
```

Operator 35 is performed for the following indexes

I	J
I	2
I	3
I	4
2	3
2	4
3	4

The FOR and NEXT operators cannot be used in the direct regime.

3.5 Array Memory Distribution Operator DIM

The DIM operator distributes the memory for one-dimensional and two-dimensional arrays. The operator has the format:

```
DIM<array name>(<dimension>[,<dimension>])
    [,<array name>(<dimension>[,<dimension>])...
```

The dimensions indicated in the DIM operator after the array name determine the maximal value of the index. The minimal index value in BASIC is equal to Ø. The array dimensions must be integers. The use of expressions is not permitted.

Example: /23

	These arrays are defined
10 DIM A (10,7),B(16),D(40)	A - dimension (Ø:1Ø; Ø:7);
	A - dimension (Ø:16);
	D - dimension (Ø:4Ø).

If the user utilizes an array without defining it in the operator DIM, the one-dimensional array has the maximal index 1Ø, while the two-dimensional array has the maximal index 1Ø * 1Ø.

When using the operator DIM, the user must remember that:
a) the DIM operator may be found in any program location;
b) the memory can be distributed to several arrays by a single DIM operator; the number of arrays in a single DIM operator

is limited only by the BASIC line length;

c) redistribution of the memory to the arrays by a new DIM operator is forbidden; if it is necessary to redistribute the memory to an array, this can be done by introducing a DIM operator with the same number (however, in this case the memory reserved for the other arrays in the first DIM operator is cleared).

Example:

`10 DIM A(7),B(3),D(10,7)`	Eight words are reserved for array A, 4 words for B, 88 words for D.
`10 DIM A(15)`	16 words are reserved for array A, arrays B and D become undefined.

The user must remember that the memory cannot be distributed by implication to arrays used in matrix operations. The DIM operator cannot be introduced in the direct regime and is a nonexecutable operator.

3.6 DATA Operator

This operator is the set of values which in the course of program execution are assigned to the indexed and nonindexed variables with the aid of the READ operator. The operator has the format:

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

The constants in the DATA operator can have any form admissible in BASIC and are separated by commas. If there are several DATA operators in the program, they form the overall ensemble of values in accordance with the operator numbers. The DATA operator is nonexecutable and cannot be introduced in the direct regime.

Example:

```
1Ø DATA 1Ø, 11, 12
2Ø REM  example
3Ø DATA 13,17E-7,-1E-3
```

A data block of six numbers is formed:
1Ø, 11, 12, 13, 17E-7, -1E-3.

3.7 READ operator

The operator is used to assign the variable values from the block of data introduced with the aid of the DATA operators. The operator has the format:

READ< variable >[,< variable >]...

The variables in the READ operator list may be indexed or nonindexed. Each variable from the READ operator list takes the values of the next constant from the data block. This continues until a value is assigned to all the variables of the READ operator list or until the data block is exhausted. In the latter case the next variable from the READ operator list takes the value of the first data block element. /25

Example:

```
2Ø READ A,B,C,D,E,
3Ø DATA 1,2,3,4
3Ø DATA 5,6,7
5Ø READ F,G,H,I
```

After execution of the operator 2Ø the values of the variables are: A = 1, B = 2; C = 3, D = 4, E = 5;
After execution of the operator 5Ø the values of the variables are: F = 6, G = 7, H = 1, I = 2.

The operator is executable and can be introduced in the direct regime.

3.8 RESTORE Operator

This operator is used to set the read indicator to a definite location in the data block. The operator has the format:

RESTORE [‹element number›]

When using the RESTORE operator without a parameter, the first data block element is assigned to the first element of the variable list of the next READ operator. When using the RESTORE operator with an element number, the READ indicator is set to the data block element with the indicated number, i.e., the first element of the data list of the next READ operator takes the values of the data block element with the number indicated in the RESTORE operator. The data block elements are numbered beginning with one. The RESTORE operator is executable and can be introduced in the direct regime.

Example: /26

```
1Ø DATA 1,2,3,4,5,6,7
2Ø DATA 8,9,1Ø
3Ø READ A,B,C
4Ø RESTORE
5Ø READ D,E
6Ø RESTORE 9
7Ø READ F,G
```

As a result of operation of the program fragment the variables take the values: A=1, B=2, C=3, D=1, E=2, F=9, G=1Ø.

3.9 Direct Input From Terminal Operator INPUT

This operator is used for operative input from the terminal of the values of the indexed and nonindexed variables. The operator has the format:

```
INPUT<element>[,<element>]
```

where <element> -- an indexed or nonindexed variable or signed constant.

During execution of the INPUT operator, all its elements are processed sequentially. If the element being processed is a signed constant, the latter is output to the terminal in the form of a line of signs. If the element being processed is an indexed or nonindexed variable, execution of the user's program is halted until a numerical constant is introduced from the terminal. After entry of the numerical constant, its value is assigned to the element being processed -- the variable.

Execution of the INPUT operator terminates after all the elements indicated in this operator have been processed. /27

Example:

```
INPUT 'A=?',A,'ARRAY',B(A),B(A+1),B(A+2)
A=?                         -- output to terminal
1.2E1                       -- user introduces from terminal
ARRAY                       -- output to terminal
1                              ┌ user inputs
-7.Ø                        -- ┤ from
-2.E-3                         └ the terminal
```

After execution of the INPUT operator, the variable A is equal to 12 and the elements of array B: B(12), B(13), B(14) are equal to 1; -7, -Ø.ØØ2, respectively.

3.10 Output Operator PRINT

This operator is used to output information in zonal or compact format. The operator has the format:

PRINT<element>[<punctuation sign><element>]...

<punctuation sign>	--	comma for the zonal format and a semicolon for the compact format.
<element>	--	expression or line of symbols

This operator outputs all the list elements to the terminal. In the zonal format case, each line is broken down into four zones of 16 symbols each. A comma standing before an element which is to be output to an external unit means that the element will be placed at the beginning of the next zone, and if the last zone in the line is filled the element will be placed at the beginning of the first zone of the new line. /28

Example:

```
1Ø PRINT  SIN(3.141592/6),2+3.5,1Ø*1E4
```

Print

```
Ø.5␣␣␣␣␣␣␣␣␣␣␣␣␣5.5␣␣␣␣␣␣␣␣␣␣␣␣␣1ØØØØØE␣Ø6
```

In the compact format, a semicolon means that the following element (subject to output) must be placed directly after the preceding element if this preceding element is a line of symbols. If the preceding element is the result of calculation of an expression, the element being output is separated from the preceding element by a blank space.

Example:

```
1Ø PRINT 1;2;'A␣';3
```

As a result of execution of the operator 1Ø, the line 1␣2␣A␣3 will be output to the external device.

If in the PRINT operator a comma or semicolon follows the last element, the action of the punctuation sign extends to the first element of the next PRINT operator. If the last element of the preceding PRINT operator was not followed by a comma or a semicolon, the elements of the next PRINT operator are output from a new line.

Numerical values in the PRINT operator are printed in the following format:

a) for values from the interval [0,I; $I0^6$], the format without an exponent is used; up to six significant digits are output;

b) for values outside the interval [0,I; $I0^6$], the format with an exponent is used; up to six significant digits are output; position 1 is set aside for the sign and the number occupies up to 12 positions.

Example: /29

```
10 PRINT 1,2;
20 PRINT 3E6,4;5;
30 PRINT 'FIVE';5
```

Print:

```
␣1␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣2␣␣Ø.30000E␣Ø6␣␣␣␣4␣5␣FIVE5
```

The PRINT operator is executable and can be introduced in the direct regime.

3.11 Transfer Operator GOSUB

This operator is used to transfer to a subprogram located in the text of the primary program. The operator has the format:

GOSUB <operator number>

As a result of execution of the GOSUB operator, control is transferred to the operator with the number indicated in the GOSUB operator. The GOSUB operator remembers the number of the operator following it for return from the subprogram. Exit from the subprogram takes place on the basis of the RETURN operator and control is transferred to the operator following GOSUB.

Example:

```
10 GOSUB 250
60 STOP
   ...
250 A = 1
300 RETURN
```

The following operators are executed: 5Ø, 25Ø, ..., 3ØØ, 5Ø .

The GOSUB operator is executable and cannot be introduced in the direct regime.

3.12 RETURN Operator /30

The RETURN operator transfers control to the operator following the last executed GOSUB operator. The operator has the format:

RETURN

The RETURN operator is executable but cannot be introduced in the direct regime.

3.13 Transfer Operator ON

The ON operator is a conditional transfer operator and has the format:

```
ON<expression>GOTO<line number>[,<line number>]...
```

The operator is used to create branching transfers in the program.

When the operator is executed, the value of the arithmetic expression standing after the key word ON is calculated. The result of calculation of the expression is rounded to the nearest integer. If the result of rounding is equal to 1, transfer takes place to the line whose number follows directly the key word GOTO in the ON operator. If the result of rounding is equal to 2, transfer takes place to the line whose number is second in the line number list. If the result of rounding is larger than the number of numbers in the ON operator list transfer takes place to the operator following the ON operator. The operator is executable but cannot be introduced in the direct regime.

3.14 STOP Operator

/31

The STOP operator terminates execution of the user program and has the format:

```
STOP
```

Upon execution of the STOP operator, the following text is output:

```
***   STOP AT nnnn  ***
```

where nnnn is the number of the STOP operator. After output of the text, operation of the program terminates. The STOP operator is executable but cannot be introduced in the direct regime.

3.15 User Function Definition Operator DEF

This operator is used to define the user function operators and makes it possible to define functions of both one and several variables with the names FNA, FNB, FNC, ... FNZ.

The operator has the format:

```
DEF FN<letter>(<parameter name>[,<parameter name>]...)=<expression>
```

<parameter name> -- name of the variable used in the expression

Upon referral in any program expression to the function /32

```
FN<letter>(<argument>[,<argument>]...)
```

the values of the arguments are calculated and the expression indicated in the DEF operator for the corresponding function is calculated. When calculating this expression, the values of the arguments calculated during referral to the function are substituted in place of the values of the variables whose names coincide with the names of the parameters. The expression in the DEF operator may contain (in addition to parameters) constants, indexed and nonindexed variables, and referrals to the functions. The values of all the variables in the expression must be determined before calculating the function. The function cannot refer to itself directly or indirectly. During referral to the function, the arguments must be coordinated with the parameters with regard to number and sequence. Any expression admissible in BASIC can be an argument.

Example:

```
1Ø DEFFNA (x,y,U)= U*A+3*X + SIN(Y)
2ØA=U=Ø
3ØC=FNA(A,U,SIN (COS(3.1415/5))
```

In this example, the function FNA is defined in the operator 1Ø with three parameters, X, Y, U. In the operator 3Ø, referral to the function FNA takes place. Upon referral, the

parameter X takes the value of the argument A=Ø, the parameter Y takes the value of the argument U=Ø, and the parameter U takes the value of the argument SIN(COS(3.1415/5)).
The operator 3Ø is equivalent to the operator:

```
3Ø C=SIN (COS(3.1415/5))*A+3*A+SIN(U).
```

Application of user functions can be recommended in the case of multiple repetition of the same expressions; however, the programmer must remember that in this case the computation speed decreases. /33

3.16 Matrix Operation Operator MAT

This operator is used to execute operations on matrices (two-dimensional arrays). The operator has one of the following formats:

```
MAT <array name>=<array name><operation><array name>
                               sign
                    or
      MAT <array name>=INV(<array name>)
      < operation > - these "+", "-", "*"
          sign         are
```

Multiplication, addition, subtraction and matrix inversion can be performed in the MAT operator. In the left side of the MAT operator there is written the name of the array taking the value, and in the right side there are written the names of the operand matrices. Multiplication, addition, subtraction and matrix inversion are performed in accordance with the rules of matrix algebra, and the corresponding signs " + ", " - ", " * ", INV are used to denote these operations. Inversion of matrices of dimension larger than 3Ø * 3Ø is forbidden.

Examples:

```
1Ø MAT A=A+B
2Ø MAT B=A*B
3Ø MAT Z=INV(X)
4Ø MAT X=INV(X)
```

3.17 Program Segment Dynamic Loading Operator FETCH

/34

The FETCH operator is used to load (input) the program parts (program segments) from the basic module library and to transfer control to an operator with indicated number.

The operator has the format:

```
FETCH [< no. 1>,< no. 2>]'< segment name >'[< no. 3>]
```

The following operations are performed using the FETCH operator:

a) the operators from number 1 through number 2 located in the operatative memory are removed (in the absence in the operator of the parameters <number 1> , <number 2> all the program operators located in the operative memory are removed);

b) the program segment with the name <segment name> , cataloged previously in the basic module library, is introduced;

c) control is transferred to the operator with the number <number 3> (in the absence of the parameter number control is transferred to the first operator of the segment introduced from the library). The FETCH operator is used to execute large programs, the text of which cannot be stored completely in the operative memory. After execution of the FETCH operator, the values of the variables and arrays remain in the operative memory without changes, except for the arrays defined by the DIM operators removed in the process of execution of the FETCH operator.

The user must remember that use of the FETCH operator slows markedly the program execution and it should not be used unless absolutely necessary. It is recommended that only those operators which are required in the segment being loaded be stored in the memory. It is recommended that from the very beginning the individual DIM operators be used to define the /35 memory for those arrays which are necessary only for the given segment and the arrays which are necessary in the subsequent segments.

The FETCH operator is executable and can be introduced in the direct regime. A check for the presence of the required segment in the library takes place only at the moment of execution of the FETCH operator.

Example:

The following segments are cataloged in the basic module library:

1) segment with the name SEGMENTØ

```
1Ø DIM A(4,4),B(7),C(4)
2Ø DEFFNA (X,Y) = (X+Y)*(X-Y)
3Ø DATA 1,2,3,4,5,6,7,8
4Ø FOR I=Ø TO 4
5Ø READ D(I),C(I)
6Ø NEXT I
7Ø DIM D(4)
8Ø FOR I=1 TO 4
9Ø FOR J=1 TO 4
1ØØ A(I,J)=FNA(C(I),D(J))
11Ø NEXT I
12Ø NEXT(J)
13Ø FETCH 4Ø,13Ø 'SEGMENT1',4Ø
```

2) segment with the name SEGMENT1

```
40 FOR I=0 TO 4+3
50 B(I)=A(I/2,(I+1)/3)+FNA(C(I/2),C(I/4))
60 NEXT I
70 FETCH 40,70 'SEGMENT2'
```

3) segment with the name SEGMENT2 /36

```
70 FOR I=Ø TO 4+3
80 PRINT 'B(',I,')','=';B(I),
90 NEXT I
100 STOP
```

If the user inputs from the terminal the directives

```
LOAD 'SEGMENTØ'
RUN
```

then initially there will be loaded into the memory the segment operators SEGMENTØ with numbers from 1Ø to 13Ø, then operators 4Ø to 12Ø will be performed, and the memory set aside for array D will be cleared. After this, the SEGMENT1 segment operators 4Ø to 7Ø will be introduced and the operators with numbers 4Ø to 6Ø will be performed. The FETCH operator with number 7Ø leads to removal of the entire SEGMENT1 segment and loading of the operators 7Ø to 1ØØ. The FOR operator with number 7Ø takes control. Execution of the entire program is terminated by the STOP operator with number 1ØØ.

4. BASIC Error Messages in the User Program Entry and Interpretation Stage

An error message is generated in case of entry of syntactically incorrect BASIC operators and in case of onset of inadmissible conditions in the user interpretation stage. The error messages have the following format:

```
BAS nnnn ERROR IN LINE tttt
```

where nnnn -- four-place error code;
      tttt -- number of the operator in which the error is discovered.

In case of errors in the operators in the direct regime /37
Ø or the symbols * * * * are printed in plaqe of the number.

The codes ØØØ1 - ØØFF are the syntactic error codes; the codes Ø1Ø1 - Ø1FF are the standard function execution error codes.

4.1 Error Message Codes

Code	Cause
0001	invalid operator number
0002	error in left side of LET operator
0003	ambiguous operator
0004	in the READ operator the expression is not used in the variable index
0005	error in syntax of expression or error in DIM operator constant list
0006	error in DEF operator parameter list
0007	add number of inverted commas in a line
0008	incorrect constant
0009	nonexistent standard function
000A	operator not completed
000B	error in DIM operator
000C	redefinition of user function, and the numbers of the DEF operators do not coincide
000D	forbidden separator used in the PRINT operator
000E	inverted comma in forbidden operator
000F	excessively long symbol constant in PRINT

0010	error in variable identifier in NEXT operation	/38
0011	comparison operation used in operator differing from IF	
0012	no comparison symbol in operator IF	
0013	operator being removed is not present.	
0014	excessively long program (large number of operators)	
0015	" " " (long overall operator text)	
0016	incorrect format of LIST command	
0017	more than 8 letters used in book name in SAVE or LOAD commands	
0018	book being loaded by LOAD command is not found in the library	
0019	no room in library for book being cataloged by SAVE command	
001A	invalid number in RUN command	
001B	inadmissible operator in direct regime	
001C	incorrect syntax of FETCH command	
001D	" " " RENUMBER operator	
001E	" " " CLEAR operator	
001F	" " " SELECT operator or inaccessible printer	
0021	right bracket missing in expression	
0022	left bracket missing in expression or excess comma is present	
0023	array of dimension over 2 is defined or used	
0024	conflicting distribution of memory by DIM operator, i.e., attempt to distribute memory by two different operators	/39
0025	excessively complex expression	
0026	memory inadequate for arrays	
0027	undetermined error in expression syntax	
0028	too many constants	
010C	over-filling of order	

010D	disappearance of order
010E	loss of significance (excessively small numbers obtained in calculation)
010F	division by zero
0111	variable being used is not defined
0112	too many simple variables
0113	inadequate memory for distribution of array by implication
0114	array index outside given bounds
0115	maximal cycle saturation exceeded
0116	too many parameters activated simultaneously
0117	undefined function is used
0118	number of argument when referring to user function not equal to number of parameters in DEF operator
0119	data block defined by the DATA operator is exhausted
011B	array used in matrix operator is not explicitly defined
011C	invalid dimensions for MAT operator
011D	incorrect information entry during operation of INPUT operator /40
0122	transfer to undefined operator
0123	depth of transfers using GOSUB is exceeded
0124	use of RETURN without GOSUB operator
0126	array element possibly did not take initial value
0201	attempt to calculate TAN(x) with $\|x\| > 3.14150202*10^{15}$.
0202	" " " TAN(x) with $\|x\| \approx 3.141592 + 3.141592 \cdot N$.
0203	" " " SIN(x) or COS(x) with $\|x\| > 3.141592020E15$
0204	" " " EXP(x) with $x > 174.673$
0206	" " " SQR(x) with $x < \emptyset$
0207	" " " X↑Y with $X \leq \emptyset,\ Y \neq \emptyset$

0208	attempt to calculate LOG(x) with x<0
0210	zero determinant of matrix being inverted
0211	attempt to invert matrix of dimension over 30*30
0220	forbidden input symbol
0224	forbidden output symbol

operators removed in the process of execution of the FETCH operator.

The user must remember that use of the FETCH operator slows markedly the program execution and it should not be used unless absolutely necessary. It is recommended that only those operators which are required in the segment being loaded be stored in the memory. It is recommended that from the very beginning the individual DIM operators be used to define the /35
memory for those arrays which are necessary only for the given segment and the arrays which are necessary in the subsequent segments.

The FETCH operator is executable and can be introduced in the direct regime. A check for the presence of the required segment in the library takes place only at the moment of execution of the FETCH operator.

Example:

The following segments are cataloged in the basic module library:

1) segment with the name SEGMENTØ

```
1Ø DIM A(4,4),B(7),C(4)
2Ø DEFFNA (X,Y) = (X+Y)*(X-Y)
3Ø DATA 1,2,3,4,5,6,7,8
4Ø FOR I=Ø TO 4
5Ø READ D(I),C(I)
6Ø NEXT I
7Ø DIM D(4)
8Ø FOR I=1 TO 4
9Ø FOR J=1 TO 4
1ØØ A(I,J)=FNA(C(I),D(J))
11Ø NEXT I
12Ø NEXT(J)
13Ø FETCH 4Ø,13Ø 'SEGMENT1',4Ø
```

0208	attempt to calculate LOG(x) with $x<0$
0210	zero determinant of matrix being inverted
0211	attempt to invert matrix of dimension over 30*30
0220	forbidden input symbol
0224	forbidden output symbol