PART II: BASIC REFERENCE GUIDE

REFERENCE MANUAL

This brief description of PolyMorphic Systems BASIC is intended to provide an easy-to-use daily reference for the BASIC programmer. The commands, etc. described here are discussed at length in the manual System 88 Disk BASIC: A Manual.

IN ALL CASES, SEE THE BASIC MANUAL FOR DETAILS.

See also the System 88 User's Manual.

This reference applies to PolyMorphic Systems Disk BASIC version C00 and C00L.

Previous versions were (tape) 8V27, 9V27, A00, P00, P01; (disk) A01, B08, B08A, B08C.

Section 1

ENTERING BASIC; LOADING, RUNNING, AND SAVING; LEAVING BASIC

1.1 ENTERING BASIC

When BASIC is part of the System Disk, it is always instantly available. To use BASIC, start the system, then when you see the Exec prompt $, type BASIC. You will then see the BASIC version number and the BASIC prompt >. This BASIC prompt indicates that you are communicating with BASIC (i.e. that the BASIC interpreter is loaded and awaiting your keyboard input).

1.2 LOADING AND RUNNING PROGRAMS

You can run a BASIC program from the Exec prompt $ or $$ without typing BASIC. Just type the file specifier (the file name, preceded by the drive number if necesary); the system will note the .BS suffix and automatically bring in BASIC. (This does not happen if you have tagged the file with some suffix other than the .BS suffix tagged to BASIC files by default.)

To load a disk file while in BASIC, type LOAD and the file specifier. If the file name uses some suffix other than .BS, you must give the suffix.

From Exec, you can load BASIC and run a program with a single statement: filename. The filename must have a suffix of .BS.

You can interrupt the running of a BASIC program from the keyboard by typing CTRL-Y (hold down the CTRL key and type y or Y).

An interruption in the running of a BASIC program will cause the system to stop and display a line number and a double BASIC prompt >>. This double prompt tells you that a BASIC program has been interrupted while executing the line with the number displayed. It also indicates that you are communicating with BASIC.

To load POLY 88 tape BASIC programs, see FILMS in the User's Manual.

1.3 SAVING PROGRAMS (SAVE, SAVEF,SAVEP)

Programs are saved (into disk files) by using the SAVE, SAVEF, or SAVEP commands, plus a comma (or semi-colon).

Sample format:

```
SAVE,<2>Program-Name
```

SAVE,filename or SAVE;filename

saves the program in text form, so that it can be edited using the system editor and printed using PRINT and TYPE.

Program files created with SAVE do not auto-execute when loaded unless the SAVE command is followed with a semi-colon instead of a comma.

SAVEF,filename

saves the program in "internal" or "token" format. The program will load faster when saved in this format. However, it cannot be edited using the System 88 Editor nor listed using PRINT or TYPE.

SAVEP,filename

Like SAVEF above, but saves the program in encrypted form.

Once a program in SAVEP format has been loaded into BASIC, BASIC will only execute the commands RUN, SCR, or BYE. Thus, SAVEP is intended as a simple means for application developers to protect the logic of BASIC programs.

Only programs saved with SAVEF or SAVEP can be CHAINed or LINKed to. Programs saved with SAVEF and SAVEP always auto-execute when loaded.

1.4 LEAVING BASIC

To leave BASIC, type EXEC or BYE. You will be returned to the operating system.

EXEC

leaves the BASIC interpreter in memory, the BASIC program loaded, the current state of all variables intact, all data files open, and allows you to continue running your BASIC program by typing CONTINUE in Exec and then CONTINUE again in BASIC.

BYE

does the same thing as a SCRATCH, which erases program and variables and closes data files. Always use BYE when finished with a program that involves data files, if the program does not close then itself.

NOTE: EXEC requires a certain amount of 8080 stack space and should not be used casually. Always finish work in Exec after EXEC and get back to BASIC with CONTINUE. Don't run another program (EDIT, etc.) until BASIC has been left via BYE. You may also use Exec RESET to throw away the ability to CONTINUE and reclaim the 8080 stack space.)

Section 2

OPERATORS AND OPERANDS

2.1 Mathematical Operators

The following operators are recognized:

-	Unary minus
^	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

2.1.1 Priority of Mathematical Operations

Unary minus is done first, then exponentiation, then multiplication and division, then addition and subtraction. Unary minus is allowed before any variable or parenthetical expression. Multiple unary minuses are considered equivalent to a single unary minus. Chains of operations of equal precedence (Ex: A+B+C+D-E+2) are done left to right. Parentheses are allowed and may be nested to any depth.

2.2 Relational Operators

Symbol	Operation
=	Equals
<	Is less than
>	Is greater than
<>	Does not equal
>= or =>	Is greater than or equals
<= or =<	Is less than or equals

BASIC will evaluate relational operations and return a 1 if true or 0 if false. Strings or numerics may be compared, but not arrays. String comparisons do alphabetic comparison using the collating sequence of ASCII. This means lower case letters are greater than upper case.

2.3 Logical Operators: AND, OR, NOT

Logical operators have this order of execution:

NOT	Logical complement bitwise
arithmetic operations	(As above)
relational operations	(As above)
AND	Logical conjunction bitwise
OR	Logical disjunction bitwise

The logical operations are done separately on the corresponding bits in the 16 bit integer representation of their arguments (bitwise.)

This should mean that:

```
100 IF NOT 1 THEN X
```

will not do X. However, the IF statement recognizes NOT 1 as false, so X is in fact executed. NOT expression, where expression yields 1, is equivalent to NOT 1.

2.4 ARITHMETIC PRECISION

BASIC "rounds" numbers to a precision of 8 decimal digits.

This precision may modified to any precision from 6 to 26 decimal digits by the use of a DIGITS statement. Once a DIGITS statement is executed, the precision will remain as specified until anotner DIGITS statement is executed or until BASIC is re-loaded. The DIGITS statement takes the following form:

```
100 DIGITS 12
```

When DIGITS is used, all variables are CLEARed. Therefore, the DIGITS statement should normally be the first statement in a program.

BASIC will automatically discover a North Star floating point board if one has been installed. When the floating point board is present in the system, BASIC can only compute 6 to 14 digits of precision.

2.4.1 NUMERIC REPRESENTATION

Whole numbers longer than the current precision are represented in scientific notation, thus:

```
 3.76E+02  means  +3.76 x 10^02  or  3.76 x 100  or  376
-3.76E+02  means  -3.76 x 10^02  or -3.76 x 100  or -376
 3.76E-02  means  +3.76 x 10^-02 or  3.76 x .01  or  0.0376
-3.76E-02  means  -3.76 x 10^-02 or -3.76 x .01  or -0.0376
```

2.4.2 STRINGS

A string constant is represented as a sequence of characters between double quotation marks: " string ". Blanks and ASCII TAB may be included. ASCII CR may not.

2.5. VARIABLES

Variables may be either numeric or string. Any variable

may be an array of unlimited dimensions, except function variables or function parameters.

Numeric variable names consist of one or two characters: a single upper-case letter, optionally followed by a single digit.

String variable names are identical to numeric, but are followed by a dollar sign $.

2.6. ARRAYS

Both numeric and string array names are the same as numeric and string scalars except for a subscript list immediately following the name. (A scalar is a non-array, or a one-dimensional array of one element.) This has the form of a list of expressions separated by commas and surrounded by parenthesis: A(1,3) or Z8$(I+3).

In a MAT statement, an array does not require a subscript list.

A given upper case letter or upper case letter and digit may be used in the name of a numeric scalar, numeric array, numeric function, string scalar, string array, and string function all in the same program. Each use will designate a different variable in each context.

The first element of an array is given an index of 1, unless a DIM0 statement precedes the DIM: then the first element of an array is number 0.

2.7 SPECIAL VARIABLES

The following variables are recognized anywhere a normal BASIC variable would be recognized, except that their values may not be changed by the programmer.

PI

Always has the value 3.14159265358979323846264432. It is truncated at the right two digits at a time when current precision is less than 26.

ERR

The value is the error code of the last error that occurred.

LINE

Returns the number of the line in which the last error occurred.

#

This special variable is set to the index number of an element by certain array functions. (See MAT) It may be changed in an assignment statement.

Section 3

INPUTTING A PROGRAM

To add or change a line, type a line number and the new line. An old line having that number will be deleted automatically.

To delete an existing line without replacing it, type the line number and hit RETURN.

To delete two or more lines in sequence, use DEL (described below).

BASIC will accept a maximum of 80 characters per line. Blanks count as characters. Blanks are never required in the interpretation of a statement (except, of course, inside string constants). Removal of blanks reduces the amount of memory space a program requires.

Program lines must begin with a line number from 0 through 65535. Blanks or tabs BEFORE line numbers are ignored.

Multiple statements may be included on one line by using one line number and separating statements with a back-slash \.

A GOSUB or function call in the midst of a multiple statement line will return to the proper statement even though it is not the first statement in the line.

An IF statement will not execute ANY of the remaining statements on a line if the condition is false and there is no ELSE clause.

If there is an ELSE clause, the statement works as one might expect: a true condition will only execute the statement before the ELSE clause, and a false condition will only execute the part of the statement between the ELSE clause and the next \.

Multiple IF statements may be used on one line.

Section 4

COMMANDS

LIST

displays a BASIC program on the screen.

To display a portion of a program, type the first and last line numbers of the block of lines you wish to see, without spaces:

```
>LIST number,number
```

To display the program from a particular line to the end of the program, type the number of the first line you wish to see followed by a comma:

```
>LIST number,
```

To display a single line, type its number (without a comma):

```
>LIST number
```

REN

Re-numbers all program lines, numbering them 10, 20, etc.

To re-number all program lines starting with a number other than 10, type REN and the desired first line number:

```
>REN 100
```

To use an increment other than 10, type REN and a first line number, then the desired increment:

```
>REN 5,5
```

REN automatically changes all references to line numbers within the program to correctly reflect the new line numbers.

Programs using the special variable LINE described in paragraph 2.7 should not be renumbered.

RUN

begins execution of a program from the first line and does a CLEAR.

To begin execution at some other line, type RUN and the line number. The existing run-time environment is preserved (no CLEAR is done).

CTRL-Y

interrupts the running of a BASIC program (hold down CTRL key and hit Y key).

CONTINUE

resumes execution of a BASIC program after an interruption. Use CON after the >> prompt only.

SCR orSCRATCH

"scratches" (erases) everything in user memory and closes all open files.

DEL

deletes blocks of lines. Give the first and last line numbers of the block to be deleted: DEL number,number. If only one number is given, only that line will be deleted.

XREF

cross-references program variables with the numbers of the lines in which they appear.

WALK

allows you to single-step through a program. The program RUNs but each statement executed is displayed on the screen, and each statement waits for a single character command to be typed by the operator. An X will execute one statement. A D will execute one statement and DUMP the scalar variables. A G will continue full speed RUN. In WALK mode, CTRL-Y is disabled.

DUMP

will cause the current values of all scalar variables and array dimensions to be printed on the screen.

Section 5

PROGRAM STATEMENTS

5.1 GENERAL STATEMENTS

REM

is for comments. BASIC ignores the statement REM and all characters to its right on the same line. REM lines in a SAVE file without line numbers will not be loaded.

STOP

stops execution. To resume, type CON.

ASSIGNMENT STATEMENTS:

LET may precede assignment statements but need not be used.

The usual form of an assignment statement is variable=expression.

Values may be numeric or string:

```
100 A=1
110 A$="Total:"
```

To assign several variables the same value at once, use a multiple assignment:

```
100 A,B=0
```

Order of assignment is right to left.

Both numeric and string variables cannot be on the left side of a multiple assignment statement.

CLEAR

erases all variables and reclaims their memory space.

DIGITS

sets the precision of numeric calculations. BASIC defaults to eight digits of precision, but will operate at from six to twenty-six digits. To change precision, type DIGITS n. North Star Hardware math boards will be automatically discovered by BASIC if plugged in, and will restrict DIGITS to from 8 to 14.

5.2 INPUTTING DATA

5.2.1 FROM KEYBOARD

INPUT

prompts for data input from the keyboard. INPUT is followed by a numeric or string variable, which is set equal to the data entered from the keyboard.

If immediately followed by a variable, INPUT displays a question mark as a prompt.

INPUT can be immediately followed by a prompt string, e.g

```
100 INPUT "Enter date:",X$
```

in which case no question mark appears. More than one variable may follow INPUT, and each will be prompted for in succession. The first prompt is a question mark or the prompt string given. Further prompts are question marks. One string value may be input per line keyed in. Many numeric values may be input per line by separating them by commas, blanks, or ASCII TABs.

INPUT1

is the same as INPUT except that it does not print a carriage return when the operator keys RETURN-- thus the display cursor remains after the data. The next INPUT will thus appear on the same line as the INPUT1.

5.2.2 Inputting Data from the BASIC Program

DATA

statements contain data used by the program. A comma-separated list of numeric or string constants must follow each DATA statement:

```
100 DATA 10,20,30,455
110 DATA "Jan","Feb","Mar"
```

READ

statements read the data in the DATA statements starting at the first data statement in the program or the data statement at the RESTOREd line number. READ statements contain lists of numeric or string variables to be read corresponding to the data in DATA statements. Every time a variable in a READ statement causes a data item in a DATA statement to be read, a pointer is advanced to the next READ variable and to the next DATA statement item. Data

items in excess of READ variables go unread. READ numeric variables must correspond to DATA numeric data and READ string variables to DATA string data.

RESTORE

sets the data pointer to the first data item in the first DATA statement in the program.

A line number may be given:

RESTORE linenumber.

The next data item read will be the first data item in line n (or after line n if line n contains no DATA statement).

If no number is given, the data pointer is set to the first data item in the first DATA statement in the program.

INP(n)

is a function which tests for characters in the input buffer and accepts input of characters from the keyboard.

INP(Ø) returns Ø if there is nothing in the input buffer.

INP(1) returns the integer value corresponding to the ASCII code of the next character waiting in the input buffer.

NOTE: See Section 11 of this reference for INPUTTING FROM DISK FILES

5.3 OUTPUTTING FROM THE BASIC PROGRAM

PRINT printlist

outputs text corresponding to the values of a list of numeric and string expressions following PRINT. The expressions are separated by commas.

Numeric values are converted to decimal and printed.

String literals are denoted by double quote marks ("). Quotation marks are, of course, not printed.

PRINT followed by nothing produces a blank line.

If there is a comma after the last element in the list, no carriage return will be printed after the last element. Otherwise, a carriage return (ASCII CR) is output.

The default PRINT format is left-justified, with the cursor displayed, all numeric elements are separated by blanks, and a carriage return automatically generated after a line is output. This default format may be altered by the programmer with TABs and "format strings."

TAB

moves the cursor (i.e. the next output character position) to the print position specified:

```
100 PRINT TAB(20),A$
```

will cause the value of A$ to be printed at position 20 on the screen.

"Format Stings"

The printed format of numbers can be altered by using "format strings" in PRINT statements. A format string begins with a per cent sign (%), followed by another character or characters indicating how the format is to be changed. These other characters are:

C puts commas into numbers as required. (e.g. 1,000,000)

$ puts a dollar sign immediately preceding numbers.

Z eliminates trailing zeroes.

If a format string contains a pound sign (#) as the character immediately following the % sign, the format as defined in that format string will thereafter be the default format.

The null format string %# returns the default format to the standard default format when used as illustrated:

```
200 PRINT %#,
```

Format strings must be seperated from elements of the print list with commas.

BASIC includes some standard formats for numeric data other than the default format. The characters below are used in combination with the % character to use these other standard formats. In all cases below, digits in excess of m are rounded. If there are digits in the element to be formatted in excess of n, asterisks are printed in the field. No error is generated by the asterisk substitution.

F Right-justified in a field n characters wide with m digits to the right of the decimal point: %nFm.

I Integers only, right-justified in a field n characters wide: %nI.

E Right-justified in a field n characters wide in scientific notation (regardless of length) with m digits to the right of the decimal point: %nEm.

NOTE: See Sections 12 and 13 for OUTPUTTING TO DISK AND/OR TO PRINTEER

INP(port)

returns an 8 bit value which is the result of doing an 8080 INP instruction from the specified port. The port number may be from 0 to 256.

OUT port,expression

is a statement (not a function) which outputs the expression to the specified port. System 88 uses 8080 ports 0 through 32. The rest are free for user I/O devices.

5.4 LOOPS

FOR and NEXT

FOR and NEXT provide for program loops. FOR-NEXT loops may be nested (Also see EXIT.)

5.5 BRANCHING STATEMENTS

GOTO

continues execution at a specified program line:

GOTO n

where n is the line number of the next line to be executed.

ON expr GOTO line#,line#,line#...

moves the program counter to one of several specified program lines. The evaluated variable or expression following ON indicates which of the line numbers following GOTO is to be selected; 1 selects the first number following GOTO, 2 the second, and so forth.

ON expr GOSUB line#,line#,line#...

moves the program counter to the line whose number follows GOSUB, which is assumed to be the first line of a subroutine; after execution of the subroutine, execution returns to the line immediately following the ON...GOSUB statement. Otherwise, this statement resembles ON...GOTO.

IF expr THEN statement

conditionally executes a statement following THEN in the same line. IF is followed by an expression which evaluates to true: 1, or false: Ø or NOT 1.

THEN can be followed by a GOTO statement or just the line number to GOTO (GOTO is implied), GOSUB line number, RETURN, PRINT print list, ON variable or expression GOTO line number, ON variable or expression GOSUB line number, any kind of assignment statement, or another IF statement.

If the IF condition is not met (expression evaluates to false), execution passes to the next line (except see ELSE below).

ELSE

When the condition in an IF...THEN statement is not met, execution can continue with another statement preceded by ELSE.

ELSE must be the next statement after THEN and must be on the same line. ELSE can introduce another IF...THEN.

If ELSE is to be followed by GOTO, the GOTO statement is assumed and need not appear; only the line number need be included.

EXIT linenumber

branches execution out of a FOR-NEXT loop. EXIT terminates ALL active FOR-NEXT loops, reclaims the associated stack memory, and passes execution to the line whose number follows EXIT.

EXIT leaves ALL current loops.

CHAIN string-expression

brings in the next program and runs it automatically as in CHAIN "Program-name" or CHAIN A$. The CHAINed to program must have been saved in token (SAVEF) or encrypted (SAVEP) format.

If the CHAINed to program is on a drive other than the

System Drive (i.e. the default drive), give the drive number (or <?>, or <#> if defined).

The "run-time environment" is preserved (no CLEAR is implied).

LINK string-expression

operates in the same manner as CHAIN except that user memory is CLEARED and SCRATCHed.

DUMP

outputs scalar numeric program variables to the output device in human readable tabular form.

WAIT

halts program execution, prints "Waiting..." on the screen, and awaits any keystrike before resuming execution.

PAUSE n

stops execution for n ticks (one tick is 1/60 second). n must be between 0 and 65535 inclusive. Expressions for n are evaluated.

ON ERROR linenumber or ON ERROR THEN statement

provides a routine or statement to be executed whenever there is an error. Any error that occurs after an ON ERROR statement has been executed causes the ON ERROR statement to be executed.

ON ERROR THEN linenumber

does an implied GOSUB to the line number; when the routine at the linenumber executes a RETURN, program execution will resume after the point of error.

Without the line number, THEN is ignored, and statement is executed.

ON ESCAPE linenumber

Upon typing CTRL-Y, BASIC begins execution of the statement number following ON ESCAPE. Otherwise identical to ON ERROR.

> NOTE: indiscriminate use of ON ESCAPE can result in programs which are impossible to abort except by pressing the Load button!

RESET

inactivates all previously executed and active ON ERROR or ON ESCAPE statements. Subsequent errors are handled exactly the same as if no ON ERROR or ON ESCAPE had been executed.

Section 6

FUNCTIONS AND SUBROUTINES

6.1 INTRINSIC FUNCTIONS

6.1.1 Standard Intrinsic Functions

Functions are called thus: function-name(argument-list). The argument-list is separated by commas.

SQRT

returns the square root of the argument.

EXP

returns the value of e (2.71828...) raised to the power specified by the argument.

LOG

returns the natural logarithm (base e) of the argument.

LOGT

returns the logarithm to the base 10 of the argument.

COS

returns the cosine of the argument (presumed to be in radians).

SIN

returns the sine of the argument.

TAN

returns the tangent of the argument.

ABS

returns the absolute value of the argument.

INT

returns the nearest integer less than the argument.

SGN

returns 1 if the argument is negative, 0 if it is zero, and -1 if it is negative.

RND

returns a real random number greater than 0 and less than 1. The argument gives the "seed value," which must be greater than 0 and less than 1: RND(n). If the seed is an integer number greater than zero, RND returns an integer number from 1 through the number given. RND(1) always returns 1. Using the same seed in the same program always produces the same sequence of pseudo-random numbers.

RANDOMIZE

is a statement (not a function) which sets the random number generator seed according to the current value of the low order 16 bits of the real time clock. This insures that each run of a program will produce a new, randomly selected pseudo-random sequence.

TIME

returns the 16 low-order bits of the real-time clock. TIME (0) returns the 16 bits; TIME with a non-zero argument returns the 16 bits and sets the timer to 0.

COSH

returns the hyperbolic cosine of the argument.

SINH

returns the hyperbolic sine of the argument.

TANH

returns the hyperbolic tangent of the argument.

ATAN

returns the arctangent of the argument, from +PI/2 to -PI/2 radians.

ASIN

returns the arcsine of the argument, from +PI/2 to -PI/2 radians.

FREE(Ø)

prints the number of unused bytes available for a user's BASIC program in RAM memory.

MEM(variablename)

returns the address (in decimal) in memory of the stated variable. Argument may not be an expression.

6.1.2 Intrinsic Functions Directly Accessing Memory and the Processor.

Numbers used in these functions must be integers.

POKE memory-address,expression

is a statement (not a function) which stores the value of the expression in the 8080 address given by memory-address.

> NOTE: POKE is very dangerous. Careless use of POKE may derange the operating system, the BASIC interpreter, the BASIC program, and many types of I/O devices in utterly unpredictable ways.

PEEK(memory-address)

returns the contents of the specified 8080 address.

6.1.3 Intrinsic String Functions

LEN(string variable)

returns the number of characters of the string currently stored in the specified string variable.

VAL(string variable)

returns a numeric value given by regarding the string argument as a number in scientific notation or integer form.

STR$(expression)

returns the scientific notation or integer representation of the value of the expression.

VAL(STR$(expr) does nothing. STR$ may include the % format declarators of the PRINT statement before the numeric argument.

ASC(string variable)

returns an integer corresponding to the ASCII code for the first character of the string specified.

CHR$(expression)

returns a one-character string which is the ASCII character corresponding to the value of the integer argument.

LEFT$(string-variable,n)

returns the left-most n characters of the string. n may be an expression. If n<0, a null string is returned; if n>LEN(string-variable), the entire string is returned.

RIGHT$(string-variable,n)

like the above, but returns the right-most n characters.

MID$(string-variable,n,m)

returns the nth through the mth characters of the specified string. If n=m, one character is returned. If n>LEN(string-variable) or m<1 or n>m, the null string is returned.

6.2 USER-DEFINED FUNCTIONS

Use FN to define your own functions.

Functions may be one-line or multi-line. The format for one line functions is:

```
DEF FNvariable-name(argument-list)=function
```

Example:

```
100 DEF FNA1(A,B)=A+B
```

For multi-line functions, the format is:

```
DEF FNvariable-name(argument-list)
```

followed by lines defining the function. The last line in the function definition must be FNEND.

Example:

```
100 DEF FNA(X)
110 ...your statement...
120 ...your statement....
130 FNEND
```

Notice the DEF cannot have any additional statements on the same line.
Arguments in function definitions are "dummies" or "formal parameters" and are replaced by the "actual" arguments given in each function call. If variables with the same names as the names of the arguments in the argument list exist elsewhere in the program, their values are not changed (the formal parameters are "local" to the function definition). The number of arguments in the function definition must equal the number of arguments in the function call.

6.3 SUBROUTINES

Execution of a subroutine begins with GOSUB line-number from outside a subroutine, and ends with RETURN in the subroutine.

RETURN returns execution to the statement following the most recent GOSUB: they may be nested.

The number of GOSUBs must equal the number of RETURNs.

Section 7

STRINGS, ARRAYS, AND MATRIXES

7.1 ARRAYS

Elements within arrays are specified by a subscript to the array variable giving the position of the element within the array. For instance, the nth item of the array X is cited thus:

```
X(n)
```

An array may have more than one dimension. An element is then referenced by its position in each dimension, in the order that the dimensions were given in the DIM statement, e.g. X(5,10).

DIM dimensions the array, thus:

DIM array-variable(number-of-elements).

Example:

```
100 DIM X(500).
```

Multi-dimensional arrays are dimensioned DIM X(n,m...).

Example:

```
100 DIM X(5,100)
```

Arrays must be dimensioned before any element can be referenced.

An array can be re-dimensioned in a program after a CLEAR statement has been executed.

> NOTE:CLEAR resets ALL variables to zero or null and destroys the "run time environment".

DIM0

The first element in an array (the array base) is number 1 by default. To number elements starting with 0, use DIM0 before using DIM. This is particularly useful when converting programs written in another version of BASIC where the array base defaults to 0.

DIM1

re-establishes 1 as the array base.

7.2 STRINGS and STRING ARRAYS

Scalar strings are automatically created and dimensioned to a maximum length of 10 characters when first used in a program.

However, strings may be dimensioned to any size:

```
100 DIM A$(1:100)
```

dimensions a scalar string of 100 characters.

Strings arrays may be created with unlimited number of dimensions:

```
100 DIM A$(5,5:15)
```

indicates that each item within the 5x5 array A$ is a string consisting of a maximum of 15 characters.

7.3 MATRIX OPERATIONS

Given one dimensional arrays A, B, and C, of equal size, the statement:

```
100 MAT A=B+C
```

sets A(1) equal to B(1)+C(1), A(2) equal to B(2)+C(2), etc. Although A above must be an array, B, C... can be expressions to be evaluated, etc.

```
100 MAT A=SQRT(B^2+C^2)
```

sets A(1) equal to the square root of (B(1)^2 + C(1)^2), A(2) equal to the square root of (B(2)^2 + C((2)^2), etc.

MAT can be combined with these other statements: LET, PRINT, READ, INPUT, PLOT, IF-THEN-ELSE. For instance, the instruction

```
100 MAT IF A=0 THEN STOP
```

results in a stop if ANY item in array A is 0.

```
100 MAT PRINT A,
```

prints out all the elements in the array A, in order. (Note the comma at the end of the line above. This will cause the printing of each element sequentially across the screen. Without the comma, each element will be printed on a seperate line on the screen.)

In general, MAT takes the size of the first array it finds to the right of MAT as the number of repetitions to make. In IF statements, fewer than this number of repetitions may be made because a true condition will stop the MAT statement's repetition. A multi-dimensional array is considered uni-dimensional, with the elements taken in row-major order. Thus the effective dimension of any array is the product of all the sizes of its dimensions. If any effective dimension of an array in a MAT statement is less than the effective dimension of the first element, a dimension error results. Otherwise, only the required part of each array is used.

During the execution of a MAT statement, the special variable # is incremented from 1 through the effective dimension of the first array. Thus

```
100 MAT A=#
```

sets the array A to the identity sequence 1,2,3,4,5,6... Also, # is set to the index of the array element which caused termination of a MAT IF statement:

```
MAT IF A=1 THEN PRINT #
```

will print the index of the first element of the array A which is equal to 1.

7.4 SPECIAL ARRAY FUNCTIONS

All the functions below take the following form: function-name(array-variable) as illustrated:

```
100 A=SUM(B)
```

> NOTE: arrays created with DIM0 in effect have zeroeth elements which may not be evident to the programmer, but which, nonetheless, are included in the computation of these functions.

SUM

returns the sum of all the elements in an array.

PROD

returns the product of all the elements in an array.

MIN and MAX

return the largest or smallest element in an array; # is set to the number of that element.

MEAN

returns the mean of the elements in an array.

STD

returns the standard deviation of the elements in an array.

Section 9

DEBUGGING FEATURES

RUN(line number)

runs the program starting with the line indicated.

DUMP

prints a list of and the current value of all the scalar numeric variables currently making up the run-time environment.

XREF

prints a listing of variables with the numbers of the lines in which they occur. Command only.

> NOTE: Both DUMP and XREF may have their output routed to the printer (if one is "attached" as described in Section 11).

WALK

lets you run the program one line at a time.

> After beginning to WALK, to run the next line, type X.
>
> To execute a single statement and DUMP, type D.
>
> To RUN from the current line, type G.
>
> WALK may be followed by a line number.

ON ERROR and ON ESCAPE

See ON ERROR and ON ESCAPE statements

ERR

is set to the error code of the most recent error. For instance, ON ERROR PRINT ERR displays the error code of each error when it occurs, then continues after the ON ERROR statement.

LINE

returns the line number of the line in which the most recent error occurred.

RESET

clears previous ON ERROR and ON ESCAPE statements; ends WALK.

Section 10

DATA FILES

10.1 FILE CHANNELS

BASIC provides eight file channels, numbered 0 through 7.

0	for inputting data from the keyboard.
1	for outputting to the screen.
2, 3	for outputting to a printer or special device.

Channels 4, 5, 6, and 7 may be used for inputting from or outputting to disk files, to a printer, or special devices.

A channel number may be given as an expression evaluating to a number 0-7. In the following discussions, "n" refers to a channel number.

10.2 DATA FILE FORMATS

There are two formats of BASIC data files:

1. Fixed record length
2. Variable record length.

When a data file is created, by writing the records of it to a channel opened in the OUT mode, BASIC keeps track of the length of each record. If all lengths are exactly equal, when the file is closed it is given a fixed record length which is one greater than the measured length. The increment is for a carriage return at the end of every record.

Fixed length record files are directly accessed in about 1/3 of a second for long files, faster for extremely short files.

Variable length record files may be directly accessed but it is not generally recommended, as it may be extremely time-consuming.

When the records of a file are re-written using INOUT mode (modes are described below), a shorter record may be written in place of any record, but not a longer one. A shorter record is padded with zero bytes between the last data character and the terminating ASCII CR, which is not moved.

If READ: and WRITE: (described below) are used in creating and maintaining the file, the records will have a CR between every data element, or field, in the record plus a CR terminating the record.

The READ: and WRITE: statements allow more than one string variable to be stored in a single record. There is no necessity to "pad" strings to fill out the record. Nor, does the programmer have to read a record as a string the size of the record and then, by programming, "break" the record into its logical data elements.

10.3 DATA FILE MODES OF OPERATION

There are three modes of operation of BASIC data files:

1. OUT files
2. INPUT files
3. INOUT files

OUT files

are created by a BASIC program. Only one OUT file may be open on a Disk Drive at one time. OUT files may not be read from.

INPUT files

provide data which may be read sequentially by a BASIC program. INPUT files may not be written to.

INOUT files

are files which may be read from or written to by a BASIC program. Records may be directly accessed, read, and rewritten in place.

INOUT files may not have new records appended to them. It is necessary to create the file, as an OUT file, with space for the maximum number of records expected to be required for the file.

10.4 FILE STATEMENTS

FILE

is a statement which operates on a channel number specified as an expression separated from FILE by a colon, with an effect determined by the function keyword, which follows the channel number, separated by a comma: FILE:n,keyword,...

Example:

```
100 FILE:6,OPEN,"<2>Real-estate",INPUT
```

Following are the allowable keywords:

OPEN

opens a data file

Example:

```
100 FILE:4,OPEN,filename,mode
```

"filename" is any string expression evaluating to a valid System 88 file name.

CLOSE

closes a channel.

Example:

```
100 FILE:4,CLOSE
```

No filename or mode is required. The channel may be the LIST channel, a data file, or a user defined channel.

POS

("position") sets the read and write pointers to a particular data record. POS is preceded by a channel number and followed by a record number expression, which evaluates to an integer from 1 to 65535. Records are automatically numbered starting with 1 in the order written originally to the file.

Example:

```
100 FILE:4,POS,A
```

would position the "read pointer" to the relative record number as expressed by the value in the variable A.

REW
("rewind") positions to record 1.

Example:

```
100 FILE:4,REW
```

10.5 FILE INPUT/OUTPUT STATEMENTS

WRITE:n,printlist
writes data to a file. Each element in the printlist is delimited by a carriage return when it is written, for compatability with READ:n.

Example:

```
100 FILE:4,POS,N
110 WRITE:4,A,A$,B,C
```

The WRITE: statement MUST be immediately preceded by a POS statement as illustrated above.

> NOTE: It is good programming practice to immediately follow the WRITE: or PRINT: (described below) statements with a REWind statement. This forces the data, which may have been written only to the internal buffer, to be written to the disk.

READ:n,readlist
inputs data from a file. READ:n operates identically to READ but obtains data from file n rather than from data statements. The variables are separated by carriage returns (ASCII CR) in each record.

Example:

```
100 READ:4,A,A$,B,C
```

PRINT:n,printlist

writes data records to the file from the printlist. Records only, not elements from the printlist, are delimited with CRs. The format of the record may be controlled identically to the manner the format of a PRINT to the screen is controlled.

Example:

```
100 PRINT:4,%5I,A,%8F2,B,B$
```

INPUT:c,readlist

Reads data into the variables specified in the readlist.

Example:

```
100 INPUT:4,A,B,B$
```

- - -CAUTION- - -

Files written using WRITE: should be read using READ: and files written using PRINT: should be read using INPUT:.

INP(n)

transfers one byte from the data file when used in the following manner:

```
100 A=INP(c)
```

Section 11

USING A PRINTER FROM BASIC

A BASIC program can use the facilities provided by the Universal Printer Driver of System 88.

First, the printer must be assigned a channel number. Thus:

```
100 FILE:2,LIST
```

"attacnes" channel 2 to the printer.

Thereafter, PRINT statements of the form:

```
100 PRINT:n,printlist
```

will cause the contents of the printlist to be printed on the printer which has been initialized by System 88.

To "attach" special devices, see the printer discussion in the BASIC manual.

Section 12

ERROR HANDLING

If BASIC detects an error in a program or command, it will normally stop execution of the program and display a message briefly describing the error. This is quite useful for a programmer testing and debugging a program.

However, it may be disastrous to a non-programmer who is executing an application program to have a program abort with an error message and a BASIC "prompt."

BASIC C00 incorporates several statements and functions to aid programmers in developing "bomb proof" application programs. Each of these has been previously described. Reviewing, they are:

ON ERROR statement
ON ERROR THEN statement
LINE special variable
ERR function
RESET statement

The following program illustrates the use of ON ERROR THEN processing to "trap" errors and execute error correction procedures in an application program.

```
REM
REM             Illustration of the use of RESET in
REM             restoring normal error processing,
REM             and the implicit GOSUB in ON ERROR.
REM             Also illustrates the LINE function.
REM
9000 ON ERROR THEN GOTO 10000 \ RETURN
9010 X=RND(10)-1
9020 PRINT 1/X,
9030 X=RND(10)-1
9040 PRINT 1/X
9050 GOTO 9010
10000 E=E+1
REM                Test for Division by Zero Error
10010 IF ERR=1036 THEN 10100
10020 RESET(ERR)\GOTO 9000
10100 PRINT "DIVISION BY ZERO IN LINE",LINE
10200 RETURN
```

The decimal values returned by the ERR special variable after an error occurs during execution of a BASIC program are listed below:

```
1024  Syntax error
1025  Syntax error
1026  Subscript error
1027  Bad argument error
1028  Dimension error
1029  Function definition error
1030  Out of bounds error
1031  Type error
1032  Format error
1033  I can't find that line
1034  FOR-NEXT error
1035  RETURN without GOSUB
1036  Division by zero
1037  Function definition error
1038  Missing matching NEXT
1039  READ error
1040  Oops...BASIC goofed!
1041  Oops...BASIC goofed!
1042  Input error
1043  Out of memory
1044  I can't do that directly
1045  Argument mismatch error
1046  Length error
1047  Overflow error
1050  Can't continue!
1051  That''s not a BASIC file!
1052  Nothing to save!
1053  That channel not open!
1054  That channel not open for INPUT
1055  That channel not open for OUTPUT
1056  End of file on that channel
1057  That program is for a different version of BASIC!
1058  CHAIN programs must be saved with SAVEF
1059  That record is past the end of the file
1060  I can only do that to a disk file
1061  End of file on that channel
1062  Type error on READ
1063  That's not a BASIC data file
1064  MAT subscript error
1065  I can't do that to a protected file!
1072  Too many digits for hardware!
1073  Renumbering error
1074  The minimum allowable precision is 6.
1075  The maximum allowable precision is 26.
1088  ....LOAD interrupted
1279  I can't do that to an OUT file
```