Data General

Extended BASIC User's Manual

093-000065-08

Extended BASIC User's Manual

093-000065-08

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000065
©Data General Corporation, 1971, 1972, 1973, 1974, 1975, 1977, 1978
All Rights Reserved
Printed in the United States of America
Revision 08, November 1978
Licensed Material - Property of Data General Corporation

Extended
BASIC
User's Manual
093-000065

Revision History:

Original Release	-	November 1971
First Revision	-	May 1972
Second Revision	-	September 1972
Third Revision	-	March 1973
Fourth Revision	-	September 1973
Fifth Revision	-	October 1974
Sixth Revision	-	February 1975
Seventh Revision	-	April 1977
Eighth Revision	-	November 1978

This document has been extensively revised from revision 07; therefore, change indicators have not been used.

Preface

Data General's Extended BASIC is a powerful, straightforward language. Its interactive qualities enable beginning programmers to develop their skills; its advanced features allow experienced programmers to handle complex and diverse applications. These attributes have made BASIC very popular, and in recent years it has become a model language for multiuser systems.

This manual presumes that you have had some experience with the BASIC language. If you have not, the introductory manual, *basic BASIC*, will give you the background you need.

This manual begins with a brief description of Extended BASIC, explains terminology and symbols, and outlines the steps you will follow to log on to the BASIC system, create and run a program, and log off. Chapter 2 explains BASIC arithmetic and strings; it includes numbers, variables, arrays, and arithmetic operations.

In Chapter 3, all interactive BASIC commands are listed alphabetically; each is explained in detail, and illustrated by example. Chapter 4 presents the BASIC functions, and Chapter 5 gives you matrix statements and commands. File input and output is covered in Chapter 6.

Appendix A lists all BASIC error messages; Appendix B explains the steps you follow when calling an assembly language subroutine, and Appendix C outlines BASIC programming on mark-sense cards. Other appendixes include Hollerith and ASCII character sets and a BASIC keyword summary.

See the following manuals for related information:

093-000088	*basic BASIC*
093-000119	*Extended BASIC System Manager's Guide*
069-000002	*Introduction to the Real-Time Disk Operating System*
093-000075	*Real-Time Disk Operating System Reference Manual*
093-000093	*Introduction to the Real-Time Operating System (RTOS)*
093-000056	*Real-Time Operating System Reference Manual*
093-000201	*Diskette Operating System Reference Manual (DOS)*
093-000087	*BATCH User's Manual*
069-000016	*Introduction to the Advanced Operating System (AOS)*
093-000122	*AOS Command Line Interpreter User's Manual*
069-000018	*Learning to Use Your Advanced Operating System*

Keyword Descriptions

In this manual, we describe each BASIC keyword in the following format:

AOS	
RDOS	
DOS	

S	
C	
F	

BASIC Keyword and Its Purpose

Format

We show the BASIC word in its generalized format. The parentheses must be inserted as shown (whereas the brackets indicate optional arguments).

Arguments

We explain how you should evaluate particular arguments that appear in each BASIC format.

Remarks

This heading includes rules, cautions, and other pertinent comments to clarify the use of the BASIC word.

Examples

We provide typical uses to help you understand the BASIC word and its format.

AOS	√
RDOS	
DOS	

A checked box next to an operating system indicates that the keyword is available for use on that operating system's version of Extended BASIC. The keyword may be available on one, two, or all three systems. Certain keywords (e.g., MSG) are available in more than one version of Extended BASIC, but differ significantly in description. In these cases we have documented the keyword for each version separately.

S	√
C	
F	

A checked box here indicates whether the BASIC word is used as a statement (S), a command (C), or a function (F). Some BASIC words may be used both as statements and commands.

Note that reserved filenames under DOS and RDOS Extended BASIC use the $ symbol as a prefix, whereas under AOS Extended BASIC the prefix is an @ symbol. We executed most of the examples shown in this manual on an RDOS system, so if you are an AOS user, please note the difference. See Chapter 6 for detailed information on reserved filenames.

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\begin{Bmatrix}\text{required}_1\\ \text{required}_2\end{Bmatrix}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the NEW-LINE or RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, in examples of system interactions, we use:

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR SYSTEM RESPONSES AND QUERIES

End of Preface

Contents

Chapter 3 - BASIC Statements and Commands

Chapter 3 - BASIC Statements and Commands (continued)

Illustrations

Tables

Chapter 1
Introduction

Extended BASIC is an interactive programming language that operates under Data General's Advanced Operating System (AOS), Real-time Disk Operating System (RDOS), Real-Time Operating System (RTOS) as a feature of RDOS, and Diskette Operating System (DOS).

RDOS supports certain commands and features which DOS does not. Generally, if you try to use an RDOS-only command or feature in a DOS system, it will be nonoperational, and you'll receive an error message.

DOS does not support cassette use or BATCH mode. Diskettes created under DOS are entirely compatible with RDOS. Diskettes created under RDOS may not be compatible with DOS when certain RDOS features not supported under DOS are used. See the *Diskette Operating System Reference Manual,* Appendix H, for more information.

General Information

Data General's Extended BASIC is an implementation of the BASIC language as developed at Dartmouth College, and includes the following features:

- String Manipulation
- Matrix Operations
- Program and Keyboard Modes
- Fixed- and Variable-Length File Manipulation
- Format Control
- Assembly Language Subroutines

Abbreviations

We use the following abbreviations in the descriptions of BASIC keywords.

Abbreviation	Term
var	numeric variable
expr	numeric expression
rel-expr	relational expression
str lit	string literal
val	numeric value
line no.	line number
col	column
control var	control variable
svar	string variable
mvar	matrix variable
filename	a disk filename or a device

Terminology

The BASIC language includes words, sometimes called keywords or instructions. When written in an appropriate format, they can act as statements and functions in a program. Many keywords can also be used as console commands (keyboard mode).

Some BASIC words can be used alone to perform an operation. Others require one or more arguments to be properly executed. For example:

```
INPUT A,B
```

In this case, A and B are arguments to the INPUT instruction.

A BASIC program consists of several BASIC statements. Each statement includes a properly formatted BASIC word preceded by a line number in the range 1 to 9999. The line number you give to a BASIC statement determines the order in which it is executed. Program execution proceeds from the lowest numbered line to the next higher numbered line unless directed elsewhere by statements such as GOTO or GOSUB.

Each program statement is written on a separate line. You terminate each line with a carriage return ()).

```
*05  PRINT "SAMPLE PROGRAM")
*10  LET A=5)
*15  LET B=2)
*20  PRINT A*B)
*25  END)
```

The asterisk (*) prompt at the beginning of each line indicates that BASIC is ready for another instruction and you may enter a command or a program statement. Your system manager can create any prompt string of up to 10 characters, or use the default asterisk prompt. We use the asterisk prompt in our examples.

BASIC console commands do not include line numbers and are executed immediately after you terminate the command with a carriage return ()).

```
*LIST)
0005  PRINT "SAMPLE PROGRAM"
0010 PRINT 5*2
  .
  .
  .
0025  END
*RUN)
SAMPLE PROGRAM
10
END AT 0025
*
```

BASIC executes the program starting from the lowest numbered line.

Using Extended BASIC

Logging onto RDOS/DOS Extended BASIC

You can log onto the system as soon as you see the BASIC prompt message on your terminal:

```
BASIC XX.XX
READY
```

You begin the log on procedure by pressing either the ESCape or RUBOUT key. Use the ESCape key on hardcopy terminals for which you want backarrow (←) as a RUBOUT echo; use the RUBOUT key on CRT terminals for which you want to backspace the cursor and erase a character as a RUBOUT echo. The system will request your account name. Respond by typing the 4-character account name assigned to you by the System Manager and a carriage return. If you have been previously assigned a password, the system will ask for it next. Type the password and a carriage return. To protect the confidentiality of your password, it will not be echoed at your terminal.

If your identification (account name and password) are valid, the system will output your account name, the date, time, and terminal number assigned, followed by a prompt.

The format of the log on procedure is as follows:

```
BASIC XX.XX
READY
```

ESC	(You press ESC or RUBOUT.)
ACCOUNT NAME: xxxx)	(4-character account name.)
PASSWORD: yyyy)	(0-20 character password not echoed.)

$$\underbrace{xxxx}_{\text{(account name)}}\square\underbrace{MM/DD/YY}_{\text{(date)}}\square\underbrace{HH{:}MM}_{\text{(time)}}\square SIGN\ ON,\square\underbrace{ZZ}_{\text{(terminal no.)}}$$

. (prompt)

Logging onto AOS Extended BASIC

You log onto an AOS system according to the instructions provided in the *AOS Command Line Interpreter User's Manual.*

Once you're logged on to AOS, respond to the CLI prompt with the command to execute BASIC. BASIC will respond with a message and a prompt.

```
)EXECUTE BASIC)
AOS BASIC REVISION XX.XX 14-SEP-77 1-OCT-77 09:14:22
*
```

Creating a New Program

Having successfully logged onto the system, you may enter a new program, or modify and run an old program. It is generally good practice to type the NEW command before entering a new program. The NEW command clears your work area in memory and thereby prevents the interspersion of lines from an old program into your new program. You can find NEW and all other interactive BASIC commands described in Chapter 3.

When typing a new program, you must begin each line with a line number of not more than four digits and end each line with a carriage return ()). If you make a typing error before pressing the RETURN key, you can correct it with the RUBOUT key. Press the RUBOUT key once for each character you want erased; then continue by typing the correct characters.

If you are working on a CRT terminal, each RUBOUT will erase the last character on the screen. Note that on some terminals a DELETE key replaces a RUBOUT key.

If you are using a hardcopy terminal, a backarrow (←) will echo at your terminal each time you press RUBOUT. For example:

```
*10 PRINT "CONV←←RRECTION BY RUBOUTS")
```

In line number 10, two characters, (N and V), are rubbed out and then the line is completed. A LIST command would output the corrected line.

```
*LIST 10)
0010 PRINT "CORRECTION BY RUBOUTS"
*
```

In addition, you can delete an entire current statement or command line by typing the backslash character on your terminal. BASIC echoes with a backslash and a carriage return. An ESC will have the same effect as the backslash key.

```
*10 PRINT "CONR\ (Backslash deletes line.)
10 PRINT "CORRECTION BY BACKSLASH")
(Line is typed over.)
```

Running a Program

After you have finished typing a program, you execute it by giving the command:

```
RUN)
```

BASIC will run the program starting from the lowest numbered statement, assuming there are no runtime errors (see Appendix A) and will output all results that you requested in PRINT statements.

Programs which were previously written and SAVEd can be run by typing:

```
*LOAD "filename")
*RUN) (or RUN "filename"))
```

where:

filename is the name of your program.

Correcting the Program

After running a program, you may need to change it because of error messages or incorrect results. You can correct the program by using any of the following procedures:

a. You can substitute a new statement for a statement containing errors by retyping the entire line (including line number and carriage return).

b. You can eliminate a statement from the program by typing its line number followed by a carriage return or by using the ERASE command.

```
*125)
```
Deletes line 125.

```
*ERASE 200,300)
```
Deletes all lines between 200 and 300 inclusive.

c. You can insert new statements between existing statements in the program simply by typing the new statements with intermediate line numbers. If the number of new statements exceeds the number of line numbers available between the existing statements, you can use the RENUMBER command (described in Chapter 3) to change the increment between line numbers. When you write a program, it is generally good practice to number your lines by increments of 10 to allow for program expansion and correction.

Interrupting Program Execution

To stop a running program, the listing of a program, or any other task which is being performed by BASIC, press the ESCape key. BASIC wil then output a prompt to signal that you can enter a new command.

```
*RUN)
 .
 .
 .
ESC

STOP AT 0110
*
```

The line number that is output is the last line that completed execution. Certain BASIC statements or commands (INPUT, LREAD, ENTER, ERASE, MAT PRINT, DELAY, MAT INPUT, LIST, FILE, and LIBRARY) may require a long time to execute. They may be aborted, but further execution dependent upon their completion will be affected.

The statement MAT INV and all file I/O statements may require considerable execution time, but can't be interrupted by ESC. If ESC is enabled on the console and you get no response to ESC, you must check to see if you're executing one of these noninterruptable statements or if there is a system failure.

Logging off RDOS/DOS Extended BASIC

After you have finished working with BASIC at the terminal, log off by typing the command BYE. The BASIC system will then output a summary of usage information and put the terminal into an idle state.

```
*BYE)
xxxx MM/DD/YY HH:MM SIGN OFF, ZZ
xxxx MM/DD/YY HH:MM CPU USED, QQ
xxxx MM/DD/YY HH:MM I/O USED, RR, SS

BASIC xx.xx
READY
```

Where:

xxxx is your ACCOUNT NAME.

MM/DD/YY is today's date.

HH:MM is the current time.

ZZ is the terminal port number.

QQ is the number of CPU seconds you used during the terminal session (calculated to the nearest tenth of a second).

RR is the number of file input and output statements you executed (OPEN, CLOSE, READ, WRITE, etc.).

SS is the number of BASIC I/O statements you executed (LIST, LOAD, ENTER, etc.).

Logging off AOS Extended BASIC

Type the BYE command to exit from BASIC and return to the CLI.

```
*BYE)
)
```

Telephone Line Interruption

If your terminal is attached to the computer by means of a Bell 103 modem (or compatible hardware), and line transmission fails for any reason, BASIC will save your current program and data in a core image file and will then do an implicit BYE command. You can retrieve the file, named AAAA$.CI ("AAAA" is your account name), by using the LOAD command. For an example, see Figure 1-1.

```
ACCOUNT NAME: MARY)                      (User signs on)
PASSWORD:

MARY 10/26/77 06:35 SIGN ON, 12

*20  FOR I = 0 TO 19)                    (User enters and runs program)
*30  ;I, SYS(I))
*40  NEXT I)
*LIST)
0020 FOR I = 0 TO 19
0030     PRINT I, SYS(I)
0040 NEXT I

*RUN)
0          23746
1          26
2          10
3          1977
4          2
5          .2
6          6
7          0
8                                        (Disconnect occurs here)

ACCOUNT NAME: MARY)                      (User repeats sign-on)
PASSWORD:

MARY 10/26/77 06:47 SIGN ON, 03

*WHATS "MARY$.CI")                       (Identifies disconnected program)

MARY$.CI  DW  379 10/26/77  06:38  (10/26/77)  00

*LOAD "MARY$.CI")                        (Retrieves it)

*LIST)                                   (Examines it)
0020 FOR I = 0 TO 19
0030     PRINT I, SYS(I)
0040 NEXT I
*...                                     (Session continues)
```

Figure 1-1. Telephone Line Interruption

BASIC Program Example

The following example shows an entire BASIC session: logging in, communicating with the system operator, running the program, and logging off.

```
BASIC xx.xx
READY
(ESC)                    (Press ESC or RUBOUT key)
ACCOUNT-NAME: KAST)
PASSWORD: JERR)   (Password JERR not echoed)
KAST 1/23/78 10:32 SIGN ON, 2

*MSG OPER PLS MOUNT TAPE #1255 (NO RING))
FROM OPER: DONE-TAPE ON MT12
*MSG OPER THANX)
*LOAD "PRODUCTION")

*LIST)
0010 DIM A$(10)
0020 INPUT "TAPE MOUNTED ON",A$
0030 A$=A$,":0"
0040 OPEN FILE (0,3),A$
0050 READ FILE (0),A,B,C$
0060 IF EOF (0)=1 GOTO 200
0070 PRINT A,B,C$
0100 GOTO 50
0200 CLOSE FILE (0)
0210 PRINT "END OF JOB"
0220 STOP

*RUN)
TAPE MOUNTED ON MT12)
.
.
.
END OF JOB

STOP AT 0220
*MSG OPER PLS RELEASE MT12)
*FROM OPER: TAPE REMOVED FROM MT12
*BYE)
KAST 1/23/78 10:40 SIGN OFF, 2
KAST 1/23/78 10:40 CPU USED, .3
KAST 1/23/78 10:40 I/O USED, 4,2

BASIC xx.xx
READY
```

Notice that this program was written to provide device independence. That is, the assignment of the magnetic tape drive number was deferred until program execution time, thereby allowing the system operator to assign any available unit.

Commands Derived From Statements

You can use most BASIC statements as console commands. However, certain statements have meaning only within the context of a program and cannot be used as commands. These statements are DATA, DEF, END, FOR, GOSUB, GOTO, NEXT, ON, REM, RETURN, RETRY, and STOP. All other BASIC statements can act as commands to help you:

- Perform file I/O
- Perform desk calculations
- Dynamically debug programs

Perform File I/O

Using BASIC, you can open and close files; and you can input or output programs and data from files and devices by commands derived from the file I/O statements described in Chapter 6.

```
*OPEN FILE (1,3), "$PTR")
*READ FILE (1), A, B, C, D, E, F, G (5))
```

Desk Calculator

With the PRINT command, you can obtain immediate results of arithmetic computations (a semi-colon (;) can be used for the word "PRINT"):

```
*;EXP(SIN(3.4/8)))
1.5103188
*LET A=EXP(SIN(3.4/8)))
*PRINT USING "+####.####↑↑↑↑",A)
+1510.3188E-03
*
```

Desk Calculator Using Program Values

You can interrupt a running BASIC program and use the assigned values of program variables to make calculations.

```
*LIST)
0010 FOR J = 1 TO 1000000
0020 X = X + 1
0030 NEXT J

*RUN)

(Press ESC key.)

STOP AT 0010
*PRINT J, 10 * X)
100   990
*
```

Dynamic Program Debugging

You can interrupt a running program (using ESC or programmed STOP statements) at a number of different program points. You can check the current values of the variables at those points and make corrections to statements or variables in the program, as necessary. You can then use either RUN line no. or CON to restart the interrupted program without losing the values of the variables at the point of interruption or the newly inserted values and statements.

Examples

1.
```
*RUN)
.
.
.
(ESC)
```
(Press ESC key.)
STOP AT 1100
```
*IF A<>B THEN PRINT B,A
```
(Command conditionally provides for examination of A and B.)
.025 .5
```
*
```

2. This program performs a series of calculations and then prints results.
```
*RUN)
.
.
.
```
2.33333
5.41234
8.99999
```
(ESC)
```
(Press ESC key.)
STOP AT 0570
```
*READ X1, X2, X3)
```
(Space over the next 3 values in the data block. Resume program execution at next statement.)
```
*CON)
```
3.16524
1.65318
```
.
.
.
```

3.
```
*RUN)
.
.
.
(ESC)
```
(Press ESC key.)
STOP AT 1100
```
*;A)
```
(Print value of variable A.)
0
```
*A = -1)
*C$ = "% OF LOSS"
```
(Change the value of arithmetic variable A and string variable C$.
```
*RUN 505)
.
.
.
```
(Resume running at statement 505.)

4.
```
*DIM A (14,4))
*RUN)
.
.
.
(ESC)
```
(Press ESC key.)
STOP AT 0500
```
*DIM A (3,5))
```
(Redimension array A.)

End of Chapter

Chapter 2
Arithmetic and Strings

Numbers

An Extended BASIC number may range from + or - 5.4 * 10^{-79} to + or - 7.2 * 10^{75}. Numbers may be expressed in integer, floating-point, or in exponential form (E-type notation).

BASIC provides either all single-precision or all double-precision calculations.

The format of converted numeric data (for example, as converted by a PRINT statement) depends upon the BASIC system you generated. The least significant digit of any printed number is always rounded.

Single-precision Print Representation

BASIC does not use exponential format for any floating-point or integer number of six digits, or less. A floating-point or integer number that requires more than six digits is printed in the following E-type notation.

(sign)n.nnnnnE(sign)XX

Where:

n.nnnnn is an unsigned number carried to five decimal places with trailing zeros suppressed.

E means "times 10 to the power of."

XX represents an unsigned exponential value.

Number	Single-precision Output Format
2,000,000	2E+06
108.999	108.999
.0000256789	2.56789E-05
24E10	2.4E+11

Double-precision Print Representation

BASIC does not use exponential form for any floating-point or integer number of eight digits, or less. A floating-point or integer number that requires more than eight digits is printed in the following E-type notation.

(sign)n.nnnnnnnE(sign)XX

Where:

n.nnnnnnn is an unsigned number carried to 7 decimal places with trailing zeros suppressed.

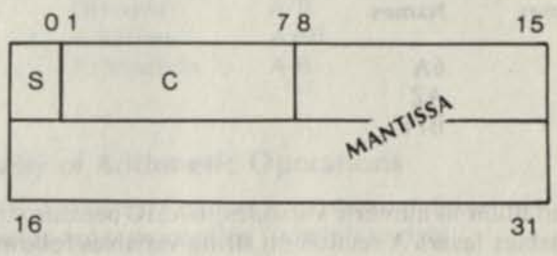
E means "times 10 to the power of."

XX represents an unsigned exponential value.

Number	Double-precision Output Format
.666666666	.66666667
108.999868	108.99987
111111111.99	1.1111111E+08

Internal Number Representation

Internally, BASIC stores numbers in a format compatible with other Data General Corporation software such as FORTRAN IV and the relocatable assemblers. Single-precision floating point numbers are stored in two consecutive 16-bit words of the form:



SD-1093

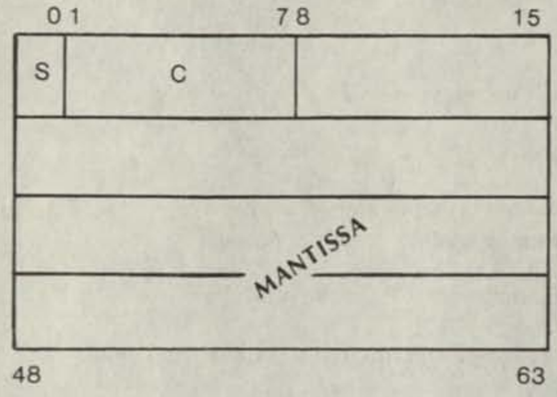
Where:

S is the sign of the mantissa (0 is positive, 1 is negative).

C is the characteristic and is an integer expressed in excess-64 code.

The mantissa is a normalized six-digit hexadecimal fraction.

Double-precision floating-point numbers add two words of precision to the mantissa, which can be represented as:



SD-1094

The internal floating-point precision is 7 decimal digits for single-precision and 16 decimal digits for double-precision. PRINT USING can be used to override the PRINT format and display the extra digits.

Variables

You must express the names of numeric variables (shown in program statements as var) as either a single letter or a single letter followed by a digit. For example:

Acceptable Variable Names	**Unacceptable Variable Names**
A	6A
A3	AZ
Z	B14
Z6	

In addition to numeric variables, BASIC permits string variables (svar). A section on string variables follows in this chapter.

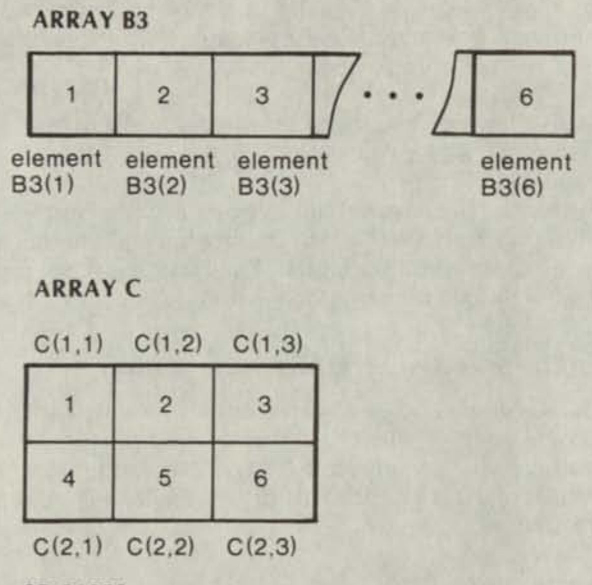
Arrays

An array represents an ordered set of values. Each member of the set is called an *array element.* An array can have either one or two dimensions. An array name may be a single letter or a single letter followed by a digit.

Array Elements

Each of the elements of an array is identified by the name of the array followed by a parenthesized subscript. See Array B3 in Figure 2-1.

For a two-dimensional array, the first number gives the number of the row and the second gives the number of the column for each element. The elements of Array C would be as shown in Figure 2-1.



SD-01058

Figure 2-1. Array Elements

A reference to element zero or a negative reference is an error.

Declaring an Array

Most arrays are declared in a DIM statement, which gives the name of the array and its dimensions. You can also dimension an array in a MAT INPUT, MAT READ, MAT READ FILE, or MAT assignment statement (a two-dimensional array is also called a *matrix).*

The lower bound of a dimension is always 1; the upper bound is given in DIM statements as in this example:

```
*10  DIM A(15), B1(2,3))
```

The upper bound of array A is 15, the upper bounds of matrix B1 are 2 and 3. These statements will also declare arrays:

```
*10  MAT A = CON (4,5))
*20  MAT READ A (4,5))
*30  MAT A = B*C)
```

The upper bound or bounds of an array are determined by the expressions in MAT or by the subscripts of the array variable as in MAT READ.

If an array is not declared, then a default value of 10 is assigned to each dimension of the array. That is, if C has not yet been dimensioned, then

```
*10  C(5) = 1
```

will create array C with 10 elements, with the fifth element set to 1. If you try to create an array or a matrix with dimensions bigger than 10, and you haven't declared the array, as in the statement

```
*10  C(9,11) = 273
```

then C will become a matrix of 100 elements, even though the error message--ERROR 31-SUBSCRIPT--will occur and the value 273 will not be stored.

There is no limit on the number of elements in a given array other than restrictions of available memory. Storage for an array is permanently allocated when it is declared. You can redimension an array, but you will only be rearranging the initially allocated space. If you try to redimension an array to a larger size, you will get the error message, ERROR 28-DIM OVFL.

Redimensioning an array using a DIM statement will not disturb the data in the array, as shown in this example:

```
*LIST)
0010 DIM A(12)
0020 FOR I=1 TO 12
0030    LET A(I)=I
0040 NEXT I
0050 DIM A(3,4)
0060 MAT PRINT A
0070 END
*RUN)
1   2   3   4
5   6   7   8
9  10  11  12

END AT 0070
*
```

A numeric scalar variable and a numeric array variable may share the same name, and BASIC will treat them as two distinct variables, provided you declare the array in a DIM statement before referring to the scalar variable. For example:

```
*10  DIM A(3,3))
*20  A=5067)
*30  MAT A=CON)
*40  PRINT A)
*50  MAT PRINT A;)
*RUN)
5067
1  1  1
1  1  1
1  1  1

END AT 0050
*
```

Line 20 refers to the numeric scalar variable A. Line 50 refers to the numeric array A.

Arithmetic Operations

A numeric expression is any combination of numbers, numeric variables, and array variables and functions, linked together by arithmetic operations. Numeric expressions appear in program statement formats as expr. The operators used in writing numeric expressions are:

Operator	Meaning	Example
+	Unary plus	A+(+B)
-	Unary minus	A+(-B)
↑	Exponentiation	A↑B (A to the B power)
*	Multiplication	A*B
/	Division	A/B
+	Addition	A+B
-	Subtraction	A-B

Priority of Arithmetic Operations

BASIC evaluates a numeric expression (expr) in the following order proceeding from left to right:

1. Any expr within parentheses are evaluated before any unparenthesized expr. When parenthesized exprs are nested, the innermost expr is always evaluated first.

2. Unary plus and minus.

3. Exponentiation.

4. Multiplication and division (equal priority).

5. Addition and subtraction (equal priority).

6. When two operators have equal priority (* and /), evaluation proceeds from left to right.

7. In a series of exponentiations, evaluation proceeds from left to right. That is, A↑B↑C = (A↑B)↑C.

For example:

Z+(-A)+B*C↑D

Step 1. A is negated.

Step 2. C↑D is evaluated.

Step 3. B is multiplied by the result of Step 2.

Step 4. Z is added to the result of Step 1.

Step 5. The result of Step 4 is added to the result of Step 3.

Parentheses

Since parenthesized exprs are evaluated first, you can use parentheses to change the order of evaluation for an expr. Using the same variables as the previous example:

Z - ((A + B) * C) ↑ D

Step 1. A + B is evaluated.

Step 2. The value from Step 1 is multiplied by C.

Step 3. The value from Step 2 is raised to the D power.

Step 4. The value from Step 3 is subtracted from Z.

With parentheses, you can clarify the order of evaluation and legibility of an expr. For example, the following exprs are equivalent:

A * B ↑ 3/4 + B/C + D ↑ 3

((A*B↑3)/4) + ((B/C) + D ↑ 3)

Relational Operators and Expressions

Relational operators are used to compare two exprs in a relational-expression (rel-expr). A relational expression has the form:

expr1 relational operator expr2

The relational operators used in BASIC are:

Symbol	Meaning	Example
=	Equal	A = B
<	Less than	A < B
<=	Less than or equal	A <= B
>	Greater than	A > B
>=	Greater than or equal	A >= B
<>	Not equal	A <> B

You may use a string (see *String Conventions* following) instead of an expr in relational expressions.

String Conventions

String Literals

A *string* is a sequence of characters which may include letters, digits, spaces, and special characters. A *string literal* (constant) is a string enclosed within quotation marks. String literals are often used in PRINT and INPUT statements as described in Chapter 3.

```
*050  REM THE NEXT STATEMENT PRINTS A STRING)
*100  PRINT "THIS IS A STRING LITERAL")
*150  REM STATEMENT 200 INCLUDES A)
*160  REM STRING PROMPT)
*200  INPUT "X=",X)
```

The enclosing quotation marks are not printed when the string is output to a terminal. You can include special and nonprinting ASCII characters in string literals by enclosing the decimal equivalent of the character in angle brackets (< >). See Appendix E for the decimal equivalents of ASCII character codes.

```
*LIST)
0010 PRINT "USE DECIMAL 34 TO PRINT < 34 > "
*RUN)
USE DECIMAL 34 TO PRINT "

END AT 0010
*
```

String Variables

Extended BASIC allows *string variables* as well as string literals. A *string variable* name consists of a letter or a letter and a digit, followed by a dollar sign ($).

Legal String Variables	Illegal String Variables
A$	A14$
A2$	AA$
D6$	2$
	2C$
	A1

String values are assigned to string variables by the use of LET, INPUT, and READ statements.

Dimensioning String Variables

Unless you declare a string variable in a DIM statement, BASIC assumes a maximum string length of 10 characters. Therefore, undimensioned string variables longer than 10 characters are truncated to 10 characters. Good programming practice suggests that you dimension all string variables, regardless of size. The length of a string must be in the range:

0 < string length < = 32767 characters

In the following DIM statement, the string A$ has a maximum length of 25 and B3$ has a maximum length of 200.

```
*10 DIM A$ (25), B3$ (200))
```

In the following example, note that BASIC has truncated the string to its assigned dimension: 15 characters (including spaces).

```
*LIST)
0010 DIM A2$(15)
0020 LET A2$="PRINT A2$ IS TOO LONG"
0030 PRINT A2$
*RUN)
PRINT A2$ IS TO

END AT 0030
*
```

Substrings

You can select portions of strings (substrings) in program statements and functions by using subscripts. Subscripted string variables have the form:

$$\text{svar}\left[\begin{Bmatrix} x \\ y,z \end{Bmatrix}\right]$$

svar is a string variable name.

x is the xth through last character of svar.

y,z is the yth through zth characters inclusive of svar.

For example:

A$	Refers to the entire string.
A$(2)	Refers to the second character through the last character in the string inclusive.
A$(I)	Refers to position I through the last character in the string inclusive (where I evaluates to a character position in the string).
A$(3,7)	Refers to character occupying positions 3 through 7 inclusive.
A$(I,J)	Refers to characters occupying positions I through J inclusive, where I< = J
A$(1,1)	Refers to only the first character in the string.

When referring to substrings, x, y, and z must not be negative and y must not be greater than z. If x or y is 0, it defaults to 1. If z is 0, it defaults to the current length of the string for substring extractions and to the dimensioned length of the string for substring assignments. For extractions, x, y, and z must not be greater than the current length. For assignments, x, y, and z must not be greater than the dimensioned length.

When an assignment provides too many characters, the extra characters are truncated. When an assignment provides too few characters, blanks are assigned to the remaining character positions. When x or y is more than 1 beyond the current length in an assignment, blanks are assigned to the character positions between the current length and the character position x or y.

Examples

```
*LIST)
0005 DIM A$(20)
0010 LET A$(1,3)="SUB"
0020 LET A$(4,10)="STRING "
0030 LET A$(11,17)="EXAMPLE"
0040 PRINT A$
*RUN)
SUBSTRING EXAMPLE

END AT 0040
*
```

You can change the value of a string variable during a program. For example:

```
*LIST)
0010 LET A$="ABCDEF"
0020 PRINT A$
0030 LET B$="1"
0040 LET A$(3,3)=B$
0050 PRINT A$
0060 LET A$(4)=B$
0070 PRINT A$
*RUN)
ABCDEF
AB1DEF
AB11

END AT 0070
*
```

Assigning Values to String Variables

READ and DATA statements can assign string values to string variables. When you include string data in a DATA list, always enclose the string elements in quotation marks.

```
*LIST)
0005 DIM A1$(20),B$(10),D$(5)
0010 READ A,A1$,B$,C,D$
0015 PRINT A,C,D$
0020 DATA 5,"ABCD","EFGH",10,"IJKL"
*RUN)
5  10 IJKL

END AT 0020
*
```

As this example shows, string data and numeric data may be intermixed in a DATA list. However, each variable in the READ statement must be of the same type (numeric or string) as its corresponding element in the DATA list or else an error message will result.

You may also use INPUT statements to input string data to a program. When you respond to the INPUT statement question mark (?), quotation marks to enclose the string are optional. If data for more than one string variable is requested by the INPUT statement, the data entered for each string must be separated by a comma or a carriage return.

You may include commas in a string by enclosing the entire string in quotation marks. To include quotation marks, enclose the decimal value 34 in angle brackets. Caution must be exercised when NULL <0>, FORM FEED <12> or CR <13> for RDOS/DOS systems, or NL <10> for AOS systems are included since these characters are string delimiters.

```
*10  INPUT A$, B$, C, D, E$)
  .
  .
  .
RUN)
?ABCD, "EF,GH", 2, 4, "SIX")
```

If you want to assign exactly what you typed to a string, use the LREAD statement. The LREAD statement does not strip leading or trailing blanks, does not use commas for delimiters, and does not process angle brackets.

Strings in IF - THEN Statements

Strings may also be used in the relational expression of an IF - THEN statement. In this case, BASIC compares the strings character by character on the basis of the ASCII character value (see Appendix E) until a difference is found. If a character in a given position in one string has a higher ASCII code than the character in that position in the other string, the first string is greater. If the characters in the same positions are identical, but one string has more characters than the other, the longer string is greater.

```
*200  LET A$="ABCDEF")
*300  LET B$="25  ABCDEFG")
  .
  .
  .
*310  IF A$>B$ GOTO 500)
(True: transfer occurs)
*320  IF A$>B$(4) GOTO 500)
(False: no transfer)
*330  IF A$(1,4)=B$(4,7) GOTO 500)
(True: transfer occurs)
```

String Concatenation

You may concatenate string variables and string literals on the right-hand side of LET statements, using a comma (,) as the concatenation operator. For example:

```
*10  DIM A$ (50),B$(50))
*15  LET A$="@$2.50, PROFIT MARGIN IS 15%")
*20  LET B$=A$(1,4),"25",A$(7,26),"1%")
*30  PRINT B$)
*RUN)
@ $2.25, PROFIT MARGIN IS 11%
*
```

String concatenation, therefore, allows the following statement:

```
*10  A$=A$,B$)
```

where A$ is concatenated by B$ to yield a new value of A$.

However, when concatenated strings are assigned to a string variable, the result of the concatenation is not calculated before the assignment is made. The variable being assigned is constructed piece by piece, therefore, care must be taken when you use the same string variable on both sides of an assignment. For example,

```
*10  A$=B$, A$)
```

does not use a temporary string to do the concatenation, so the value of A$ on the right side of the equal sign is not the original value, but rather the value after B$ has been assigned.

```
*LIST)
0003 A$ = "12345"
0006 B$ = "A"
0010 A$ = B$,A$
0020 PRINT A$
*RUN)
AA
END AT 0020
*
```

String Arithmetic

You can perform arithmetic on string variables and string literals. The arithmetic operation will be executed provided the strings (or substrings which begin at the first character of the strings) have legal numeric values. Any alphanumerics which follow the numeric substring are ignored. If the substring is not a legal number, an error condition will occur.

Valid String	**Invalid String**
"123"	"FRED"
"123."	"123.E+FRED"
"-123."	"-+123"
"-123.E5"	"FRED123"
"-123.E-5FRED"	

Notice that decimal points, signs, and exponential format are permitted in the substring as long as they conform to the numeric representation described at the beginning of this chapter.

You may use the operators +, -, *, and / to link strings and create an expression to be evaluated numerically. The concatenation character (,) may not be used in a string arithmetic expression.

```
*LIST)
0010 LET A$="1234 GEARS"
0020 LET B$="5678 GEARS"
0030 PRINT A$+B$+"10"
*RUN)
6922.

END AT 0030
*
```

BASIC returns 18 digits of precision when string arithmetic calculations are made. If any precision is lost, an error message is output. For example:

```
*10 PRINT "123E27" + "5.793E-4")
```

This statement would cause an error message since the decimal point location for the two strings causes the number of significant digits to exceed 18.

End of Chapter

Chapter 3
BASIC Statements and Commands

Introduction

This chapter describes the most common Extended BASIC statements and commands. The keyword descriptions are arranged in alphabetical order. Keyword descriptions for functions, matrix arithmetic, and file I/O are described in Chapters 4, 5, and 6, respectively.

AOS	✓		S	✓
RDOS			C	✓
DOS			F	

ACL

prints a report of, or changes the Access Control List for a file in your directory.

Format

ACL "filename" *[, "UserID", "attributes"]...*

Arguments

filename is a string literal or string variable that evaluates to a filename in your directory.

UserID is a string literal or string variable that evaluates to your identification.

attributes are file attributes as described under *Remarks.*

Remarks

1. The file attributes which you may use in an ACL command are:

   R Read Access. Permits UserID to examine the data in filename.

   W Write Access. Permits UserID to modify data in filename.

   O Owner Access. Permits UserID to change the Access Control List for filename, delete the file, or rename the file.

   E Execute Access. Permits UserID to execute the file.

   A Append Access. This attribute has no meaning for nondirectory files. For directory files, Append Access permits you to make entries in the directory.

2. When you create a file in your own directory, the file will automatically have all five attributes, OWARE, for your UserID.

3. The ACL command allows you to change the Access Control List to allow for others to have full or partial access to your file, or to change your own access privileges.

4. You must string the file attributes together in the attributes argument without any delimiting spaces or punctuation; you may express them as either string literals or string variables.

5. The ACL command, followed only by filename (no UserID and attributes arguments) prints the current Access Control List for filename.

6. For RDOS/DOS systems the CHATR statement provides a similar facility.

Examples

```
*ACL  "PAGE2.2","JOE","REWAO","MARK","RE")
*ACL  "PAGE2.2")
```
JOE,OWARE MARK,RE

In this example, JOE has all access privileges to PAGE 2.2, and MARK is limited to Read and Execute access privileges.

AOS	✓
RDOS	
DOS	

S	✓
C	✓
F	

AUDIT

copies console input and output to a file named by argument.

Format

AUDIT *["filename"]*

Arguments

filename is an optional string literal or string variable to represent the audit file that would contain console input and output.

Remarks

1. AUDIT is valid as a statement or as a command. Using AUDIT as a command allows copying of both console input and output.
2. Only one audit file can be in effect at a time. When an audit file is opened, an error message occurs if the file already exists.
3. AUDIT without *filename* will shut off the AUDIT operation.

Examples

```
*LIST)
0010 DELETE "COPY.DT" ! DELETE OLD FILE
0020 LET A$ = "COPY.DT" ! FORM AUDIT ARG
0030 AUDIT A$ ! OPEN "COPY.DT" AS AUDIT FILE
0040 LET A = 3 ! ASSIGN A
0050 PRINT -A ! OUTPUT -3 TO CONSOLE
0060 AUDIT ! CLOSE AUDIT FILE
0070 END
```

When this program is RUN, the file *"COPY.DT"* will contain the three ASCII bytes "-3 <012>" that are output to the console.

AOS	
RDOS	✓
DOS	✓

S	✓
C	✓
F	

BYE

signs off from the BASIC system and makes the terminal available to others.

Format

BYE

Remarks

1. You can use BYE as a console command or as a program statement to log off the system.
2. BASIC displays accounting information after you enter the BYE command.
3. Telephone connections are severed.
4. The ESC key will not be recognized until the messages which follow BYE have been output.
5. Do not disconnect your terminal until the BYE sequence is complete.

Examples

```
*BYE)
xxxx 01/02/76 10:06 SIGN OFF, 04
xxxx 01/02/76 10:06 CPU USED, 206
xxxx 01/02/76 10:06 I/O USED, 11, 137

BASIC xx.xx
READY
```

In the first line, 04 is the terminal number. The second line indicates that 206 seconds of CPU time were used. The third line shows that 11 file I/O calls were made and 137 BASIC I/O calls were made.

AOS	✓
RDOS	
DOS	

S	✓
C	✓
F	

BYE

signs off from BASIC and returns one level, or returns to the CLI, or logs off, depending on how your system is configured.

Format

BYE

Remarks

You can use BYE as a console command or as a program statement to exit from BASIC and return to the CLI.

Examples

```
*BYE)
)
```

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	✓
F	

CALL

calls a subroutine written in assembly language from an Extended BASIC program.

Format

CALL subr *[,expr]...*

Arguments

subr is a positive integer representing an assembly language subroutine number.

expr is as many as eight optional arguments to be passed to the subroutine. Arguments may be arithmetic or string variables, or expressions.

Remarks

1. All variable arguments passed to an assembly language subroutine must be initialized before using them in a CALL statement, or else an error message will occur.

2. Arrays cannot be passed as arguments to a CALL subroutine. Therefore, dimensioned variables used as arguments to a CALL statement must be considered as expressions and must include subscripts indicating the one element to be passed to the CALL.

3. Appendix B describes creating assembly language subroutines which may be CALLed from Extended BASIC programs.

Examples

```
0005 LET A = 12
0010 LET B = A * 2
0015 CALL 33,A,B
  .
  .
  .
```

Statement *15* CALLs subroutine *33*, with the values of *A* and *B* as arguments to the subroutine.

AOS	
RDOS	✓
DOS	✓

S	✓
C	✓
F	

CARDS

transfers and merges BASIC statement lines in DGC mark sense card format from the card reader, other device or disk file named by filename into your current program storage area.

Format

CARDS "filename"

Arguments

filename is a device or disk file, expressed as a string literal or variable.

Remarks

1. If filename is a disk file, BASIC searches for the filename in your directory first. If not found, BASIC searches the library directory for the filename.
2. When a statement line from filename has the same line number as a line in the current program, the current statement is replaced.
3. CARDS provides a convenient method of entering statements from Extended BASIC mark sense cards. (Use ENTER for statements not in mark sense card format.)
4. CARDS provides a way to use both 80 column punch cards and 37 column DGC mark sense cards in the same BASIC system, on machines with two card readers, or one reader equipped to read both types of cards. However, 37 column and 80 column cards cannot be mixed in the same deck. You may enter mixed programs in two steps, as shown in the example.

Examples

```
*NEW)
*ENTER "$CDR")
*CARDS "$CDR")
```

BASIC statements on 80 column cards are merged with statements on special DGC mark sense cards.

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	✓
F	

CHAIN

runs a separate program when the CHAIN statement is encountered in your program.

Format

CHAIN "filename" *[THEN GOTO line no.]*

Arguments

filename is a string variable or string literal that evaluates to a disk filename or a device.

line no. is a line number in program filename.

Remarks

1. When BASIC encounters a CHAIN statement in a program, it stops execution of that program, retrieves the program named in the CHAIN statement from the specified device or file, and begins execution of the CHAINed program.
2. If the program is on disk, the system searches your directory for filename; if not found, the system will search the library disk directory.
3. If filename is found, BASIC clears your current program from memory and loads filename into memory. If filename is not found, your current program remains in memory, and an error message occurs.
4. CHAIN does not change the status of files. Open files remain open, and current file position pointers are maintained.
5. A program must be in core image format before it can be CHAINed, and may have been partially executed before it was SAVEd.
6. By default, all variables are cleared from the new program, and it is run from the lowest numbered statement. If CHAIN THEN GOTO *line no.* is used, variables in the main program maintain the values they had when the program was SAVEd, and the program is run from *line no.* If *line no.* doesn't exist in the new program, the new program is loaded but an error message occurs.

Examples

```
0010 READ A
0020 IF A > 5 THEN GOTO 0060
0030 IF A = 5 THEN GOTO 0070
0040 DATA 4,1,6,3,5
0050 GOTO 0010
0060 CHAIN "SERVICE"
0070 CHAIN "SUBR" THEN GOTO 0050
```

AOS	√		S	√
RDOS			C	√
DOS			F	

CHAR

changes or prints a report of the current device characteristics.

Format

$$\text{CHAR}\left[\left\{\begin{array}{l}\textit{"ON"}\\ \textit{"OFF"}\\ \textit{"characteristics"}\\ \textit{"LPP", svar}\\ \textit{"CPL", svar}\\ \textit{"device"}\end{array}\right\}\left[,\left\{\begin{array}{l}\textit{"ON"}\\ \textit{"OFF"}\\ \textit{"characteristics"}\\ \textit{"LPP", svar}\\ \textit{"CPL", svar}\\ \textit{"device"}\end{array}\right\}\right]\ldots\right]$$

Arguments

svar is a string variable or string literal which represents a numeric value.

device is the name of a terminal expressed as a string variable or string literal.

characteristics is a device characteristic, expressed as a string variable or string literal. See *Remarks* for a list of device characteristics.

Remarks

1. If you type the CHAR command without a list of arguments, BASIC prints a report listing the current device characteristics on your terminal.

2. The *ON* and *OFF* arguments apply only to *characteristics* arguments and do not apply to *device*, *LPP*, or *CPL* arguments.

3. When you use the CHAR command, the keyword *ON* is in effect until BASIC encounters the keyword *OFF*. *OFF* then remains in effect until BASIC encounters *ON*.

4. Available device names include the following:

4010I (Infoton)	4010A
6012	6040
6052	6053

5. The CHAR command only changes the device characteristics specified in the command. It does not replace the existing device characteristics.

6. *LPP* sets lines per page; *CPL* sets characters per line.

7. The MOD characteristic (device on a modem line) cannot be set on or off with the CHAR command; it is displayed for your information only.

8. The device characteristics which you can turn *ON* or *OFF* are listed below. Not all characteristics apply to all devices.

ST	simulate tab settings (eight columns).
SFF	simulate form feed.
EPI	require even parity on input.
WRP	device wraps around when line too long.
SPO	set even parity on output.
RAF	send 21 rubouts after each form feed.
RAT	send 10 rubouts after each tab.
RAC	send 10 rubouts after each carriage return or new-line.
NAS	device is non-ANSI standard.
OTT	old TTY: convert 175 and 176 to 33 octal.
EOL	do not execute automatic carriage return - line feed if CPL is exceeded.
UCO	convert lowercase output to uppercase.
LT	output 55 nulls upon open and close.
FF	output a form feed upon open.
EB0	echo all characters.
EB1	echo all characters except control characters.
ULC	input both upper- and lowercase.
PM	device is in page mode.
NRM	disable message reception via ?SEND.
TO	timeouts enabled on reads and writes.
TSP	no trailing blank suppression (card reader).
PBN	packed format on binary read (card reader).
ESC	ESC character produces an interrupt.

Examples

```
*CHAR)
CRT2  LPP 24 CPL 80
ON    ST EPI NAS EB0 ESC
OFF   SFF SPO RAF RAT RAC OTT EOL UCO LT FF EB1 ULC PM NRM MOD TU TSP PBN WRP
*CHAR "OFF","ST")
*CHAR)
CRT2  LPP 24 CPL 80
ON    EPI NAS EB0 ESC
OFF   ST SFF SPO RAF RAT RAC OTT EOL UCO LT FF EB1 ULC PM NRM TO MOD TSP PBN WRP
```

AOS	
RDOS	√
DOS	√

S	√
C	√
F	

CHAR

changes or prints a report of the current device characteristics.

Format

$$\text{CHAR}\left[\begin{Bmatrix}\textit{"ON"}\\ \textit{"OFF"}\\ \textit{"characteristics"}\end{Bmatrix}\left[,\begin{Bmatrix}\textit{"ON"}\\ \textit{"OFF"}\\ \textit{"characteristics"}\end{Bmatrix}\right]\ldots\right]$$

Arguments

characteristics is a device characteristic, expressed as a string variable or string literal. See *Remarks* for a list of device characteristics.

Remarks

1. If you type the CHAR command without a list of arguments, BASIC prints a report which lists its current characteristics on your terminal.
2. When you use the CHAR command, the keyword *ON* is in effect until BASIC encounters the keyword *OFF*. *OFF* remains in effect until *ON* is encountered.
3. The CHAR command only changes the device characteristics specified in the command. It does not replace the existing device characteristics.
4. The device characteristics which you can turn *ON* or *OFF* are listed below.

NCR	no carriage return echo.
DSP	disable spooling.
DLC	disable line feed after carriage return.
XON	XON/XOF protocall for $TTR (only on multiplexor terminals).
DNF	disable 20 nulls after form feed.
NOE	no echo of input.
BSP	backspace for rubout.
DTS	disable tab simulation.
ESC	escape character produces interrupt.
NRM	disable message reception from other users.

5. The MOD characteristic (device on a modem line) cannot be set on or off with the CHAR command; it is displayed for your information only.
6. For the CLI system console, the NCR, DLC, DNF, and DTS characteristics are valid only in input mode.

Examples

```
*CHAR)
ON XON ESC
OFF NCR DSP DLC DNF NOE BSP MOD DTS NRM
*
```

```
*CHAR "ON","BSP")
*CHAR)
ON XON BSP ESC
OFF NCR DSP DLC DNF NOE MOD DTS NRM
*
```

AOS		S	√
RDOS	√	C	√
DOS	√	F	

CHATR

changes, adds, or removes the resolution file attributes assigned to a file which already exists in your directory.

Format

CHATR "filename", attributes

Arguments

filename	is a disk file in your directory, expressed as a string literal or string variable.
attributes	are file attributes described under *Remarks.*

Remarks

1. Attributes which can be set by the BASIC CHATR command are compatible with those of the RDOS CHATR command.

2. File attributes may be strung together in the attributes argument without delimiting spaces or punctuation, and may be expressed as a string literal or string variable.

3. The attributes given in the BASIC CHATR command replace existing attributes, unless you specify otherwise.

4. Any attempt to CHATR an open file will generate an error condition.

5. The attributes which may be added or removed by the BASIC CHATR command are:

P	Permanent file. The file filename cannot be deleted or renamed after you assign this attribute.
R	Read-protected. The file filename cannot be accessed for reading.
W	Write-protected. The file filename cannot be altered.
H	Sharable. The file filename may be accessed by other users if they know the directory and filename. The file is permanent (P) and write-protected (W).
O	Sharable. The file filename may be accessed by other users if they know the directory and filename. The file is not permanent (P) or write-protected (W) and, therefore, may be deleted, written into, or renamed by other users.
E	Execute only. Other users may execute the BASIC program contained in filename, but they cannot examine the program source statements. Commands such as the LIST or SAVE commands will result in an error message.
0	Zero. Removes current file attributes. When 0 is listed with other attributes in a CHATR command, the attributes listed replace existing attributes.
*	Asterisk. Preserve current file attributes and add those specified. The asterisk may only be used in conjunction with other attributes in the argument.
+	Plus. Preserve current file attributes and add those following the plus sign.
-	Minus. Only those attributes following the minus sign are removed.

Examples

```
*WHATS "TESTFILE")
TESTFILE.   D        260
*CHATR "TESTFILE","WP")
*WHATS "TESTFILE")
TESTFILE.   WPD      260
*
```

AOS	√	S	√
RDOS		C	√
DOS		F	

CLI

provides access to the CLI without terminating the BASIC process.

Format

CLI *[command]*

Arguments

command is any valid CLI command.

Remarks

1. If you execute a CLI command, the BASIC process is temporarily suspended until the CLI command is completed.
2. The CLI command, without a command argument, allows you to remain in the CLI. To return to BASIC, type: BYE)

Examples

```
*CLI)
)TIME)
13:25:12
)DUMP @MT0:0 -.SR)
)BYE)
*CLI TIME)
13:26:25
*
```

AOS	√	S	
RDOS	√	C	√
DOS	√	F	

CON

continues the execution of a program after a STOP statement in the program has been executed, the ESCape key has been pressed, or an error has occurred.

Format

CON

Remarks

1. The CON command causes the continuation of the program from the point where it stopped.
2. If a runtime error is encountered within the program, you may correct the error and issue the CON command to begin execution from the statement following the one in which the error occurred.

Examples

```
*LIST)
0010 PRINT "PRINCIPAL  INT(%) ";
0020 PRINT "TERM(YRS)  TOTAL"
0030 READ P,I,T
0035 IF T=0 THEN GOTO 0080
0040 LET A=P*(1+I/100) ↑ T
0050 PRINT P: TAB(12):I;
0055 PRINT TAB(21);T: TAB(32);A
0060 GOTO 0030
0070 DATA 1000,5,10,0,0,0
0080 PRINT
0090 PRINT "CHANGE DATA AT LINE 70"
0100 STOP
0110 GOTO 0010
*RUN)
PRINCIPAL   INT(%)   TERM(YRS)   TOTAL
1000        5        10          1628.8946

CHANGE DATA AT LINE 70

STOP AT 0100
*70 DATA 2500,3,10,1459,6,12,0,0,0)
*CON)
PRINCIPAL   INT(%)   TERM(YRS)   TOTAL
2500        3        10          3359.7909
1459        6        12          2935.7947

CHANGE DATA AT LINE 70

STOP AT 0100
*
```

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	
F	

DATA

provides values for variables specified in a READ statement.

Format

$$\text{DATA} \begin{Bmatrix}\text{val}\\ \text{"str lit"}\end{Bmatrix} \left[\begin{Bmatrix}, \textit{val}\\ , \textit{"str lit"}\end{Bmatrix}\right] \ldots$$

Arguments

val and str lit are elements that can form a list of numeric values and string literals.

Remarks

1. You can use more than one DATA statement in a program.
2. The DATA statement is a nonexecutable statement. The values appearing in a DATA statement or statements form a single list. The first element of this list is the first item in the lowest numbered DATA statement. The last item in this list is the last item in the highest numbered DATA statement.
3. Both numbers and string literals may appear in a DATA statement and each value in the DATA statement list must be separated from the next value by a comma.
4. String literals must be enclosed in quotes.

Examples

0100 DATA 1, 17, "AB,CD", -1.3E-13

(See the READ and MAT READ statements for usage and additional examples.)

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	✓
F	

DELAY

delays program execution for a specified amount of time.

Format

DELAY = expr

Arguments

expr is a numeric expression which represents time in seconds to the nearest tenth of a second.

Remarks

1. You can use DELAY to postpone program execution on an error condition before attempting a RETRY.
2. In AOS, the maximum number for expr is approximately 4,000,000. In RDOS and DOS, the maximum is approximately 65,000.

Examples

0005 ON ERR THEN 100
0010 OPEN FILE (0,2), "THISFILE"
0020 LET I=0

.
.
.

0100 IF SYS (7) <> -48 THEN 200
(Is anyone else using this file?)
0105 I = I + 1
0110 IF I > 10 THEN GOTO 200
(Allows 10 RETRY attempts.)
0120 DELAY=10
(One second DELAY before RETRY.)
0125 RETRY
(Returns to statement which caused error.)
0200 STOP

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

DELETE

removes a file from your directory.

Format

DELETE "filename"

Arguments

filename is a file in your directory expressed as a string literal or string variable that is not protected (see CHATR).

Remarks

1. After the DELETE command, BASIC searches your directory and deletes the directory entry for filename.

2. An error message is returned if the file cannot be found, is delete-protected, or is not in your directory.

Examples

```
*DELETE "TEST.SR")
*
```

BASIC removes file TEST.SR from your directory and frees the disk blocks which it occupied.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

DIM

defines the size of one or more numeric variable arrays.

Format

$$\text{DIM}\left\{\begin{array}{l}\text{svar (n)}\\ \text{array (m)}\\ \text{array (row,col)}\end{array}\right\}\left[\left\{\begin{array}{l}\textit{,svar (n)}\\ \textit{,array (m)}\\ \textit{,array (row,col)}\end{array}\right\}\right]\cdots$$

Arguments

array is a BASIC numeric variable name.

m is an expression for the number of elements in a one-dimensional array.

row is an expression for the number of rows in the array.

col is an expression for the number of columns in the array.

n is the maximum string length.

Remarks

1. **Array Elements**

   The concept of arrays is described in Chapter 2. The DIM statement declares the size of an array to be a number of elements other than the default number (10) for each dimension. For example:

```
*10 DIM A(13),B(7,7),C(20,5))
```

   The initial value of all elements in an array is zero until your program assigns other values.

   Any variable or expression that you use for a subscript must have a value between 1 and the value given in the DIM statement. For example:

```
*01  DIM A (5,5))
*05  X=2)
*10  PRINT A(1,X↑2))
```

   If the variable or expression subscript does not evaluate to an integer, BASIC will convert it to an integer using the INT function (see Chapter 4).

   If a subscript evaluates to a larger integer than the upper bound of the dimension for the array or smaller than 1, a subscript error condition occurs.

2. **Redimensioning Arrays**

   You can redimension a previously defined array during execution of a program by declaring the array in another DIM statement. The total number of elements of the newly dimensioned array must not exceed the original total number of elements.

```
*100  DIM A(3,3))
  .
  .
  .
*200  DIM A(2,3))
  .
  .
  .
*300  DIM A(2,2))
```

   BASIC reassigns the values of elements in array A(3,3) to elements in array A(2,3) and then to elements in array A(2,2) as follows:

$$\begin{vmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{vmatrix}\qquad\begin{vmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{vmatrix}\qquad\begin{vmatrix}1 & 2\\ 3 & 4\end{vmatrix}$$

```
A(1,1) = 1  A(1,1) = 1  A(1,1) = 1
A(1,2) = 2  A(1,2) = 2  A(1,2) = 2
A(1,3) = 3  A(1,3) = 3  A(2,1) = 3
A(2,1) = 4  A(2,1) = 4  A(2,2) = 4
A(2,2) = 5  A(2,2) = 5
A(2,3) = 6  A(2,3) = 6
A(3,1) = 7
A(3,2) = 8
A(3,3) = 9
```

3. For a discussion on the dimensioning of strings, please refer to Chapter 2.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

DISK

determines the number of 256-word blocks still available in the partition in which your directory resides.

Format

DISK

Examples

```
DISK)

USED: 332
LEFT: 193
```

This message indicates that 193 out of 525 blocks are still available for use.

AOS	√
RDOS	√
DOS	√

S	√
C	
F	

END

terminates execution of the program and returns to interactive mode.

Format

END

Remarks

1. Data General's Extended BASIC does not require an END statement to declare the physical end of a program. If a control passes through the last executable statement of the program and if that statement does not change the flow of control (as with a GOTO or GOSUB statement), then the program will transfer control to interactive mode. We have included the END statement for compatibility with BASIC programs written for other systems.

2. Multiple END statements may appear in the same program. When an END statement is encountered it will terminate execution of the program followed by a prompt at your terminal.

3. If an END statement is executed before a FOR/NEXT terminating condition is reached, an error message will be printed.

Examples

```
*20  PRINT "PROGRAM DONE")
*30  GOTO 60)
 .
 .
 .
*60  END)
*RUN)
PROGRAM DONE

END AT 0060
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

ENTER

transfers and merges BASIC statement lines from filename into your current program storage area.

Format

```
ENTER {"filename"}
      {svar      }
```

Arguments

filename is a device or disk file, expressed as a string literal.

svar is a string variable representing filename.

Remarks

1. If filename is a disk file, BASIC searches for filename in your directory first. If not found, BASIC searches the library directory for filename.
2. When a statement line from an ENTERed filename has the same statement number as a line in the current program, the ENTERed statement replaces the current program statement.
3. ENTER does not restore data statments even though some may exist in the ENTERed program. If a RESTORE is desired, it must be explicitly coded.

Examples

```
*NEW)
*ENTER "TEST1.SR")
*ENTER "TEST2.SR")
*LIST "FINAL.SR")
```

Your storage area is cleared and source programs TEST1.SR and TEST2.SR are ENTERed and merged. The resultant program is LISTed to your directory as FINAL.SR.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

ERASE

removes statements from a program.

Format

ERASE n1, n2

Arguments

n1 and n2 are line numbers in a program.

Remarks

1. This command removes n1 through n2, inclusively, from your program. ERASE simplifies editing by allowing you to delete more than one line at a time.
2. Typically, you use the ERASE statement in a program to clear a range of statements for replacement with the ENTER command, or to remove initialization code which is not needed during execution.
3. If no lines exist in your program in the range n1 to n2, BASIC outputs an error message to your terminal. If n1 and/or n2 don't exist but there are lines in your program between n1 and n2, those lines are erased.
4. Both line number arguments are necessary.
5. The RENUMBER command does not renumber the arguments to the ERASE statement, since they most often refer to fixed statement numbers in an external program file. If you RENUMBER a program, make certain you change any ERASE statements in the program to agree with the new line numbers.

Examples

```
*10 ERASE 1500, 1900)
```
(Delete lines 1500 through 1900.)

```
*20 ERASE 200, 200)
```
(Delete line 200.)

AOS	√		S	√
RDOS			C	√
DOS			F	

FILE

prints all the filenames in your directory that match the template.

Format

FILE *[''template'']*

Arguments

template is any combination of up to 15 valid characters, including asterisk (*), dash (-), and plus sign (+), in accordance with AOS template rules. You must express the template as a string literal or string variable.

Remarks

1. If you omit the template argument, then BASIC prints a list of all files in the directory.
2. The following information follows each filename printed:
   - the type of file
   - the date last modified
   - the time last modified
   - the size of the file (in bytes)

Examples

```
*FILE "-.-")
PAGE2.2  UDF  11-MAR-77  11:17:48  78
*FILE "-.SR)
*FILE "-.2")
PAGE2.2  UDF  11-MAR-77  11:17:48  78
*A$="-.2")
*FILE A$)
PAGE2.2  UDF  11-MAR-77  11:17:48  78
*
```

AOS			S	√
RDOS	√		C	√
DOS	√		F	

FILE

prints all filenames in your directory.

Format

FILE

Remarks

BASIC prints one filename per print zone.

Examples

```
*FILE)
157.       134.          GOSUB1.SR
STOP.SR    121.          NEW.
ON.ES      READ.SR       116.
TIME.      FOR1.SR       FOR2.SR
MORSE.     110.SR        FOR4.SR
113.       TAB.SR        132A.
COM.CM     CON.          110.
TAB.       SUBSTRINGS.   CONCAT.
115.       GOTO.         111A.
PAGE.      HELLO.SV      PRINT1.SR
PRINT2.SR  109B.         PRINT3.SR
PRINT4.SR  92.           INPUT2.SR
107.       117.          IF3.
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	
F	

FOR and NEXT

execute a block of statements a specified number of times.

Format

```
FOR control var=expr1 TO expr2 [STEP expr3]
.
.
(Block of statements)
.
.
NEXT control var
```

Arguments

control var   is a nonsubscripted numeric variable.

expr1   is a numeric expression which defines the initial (first) value of control var.

expr2   is a numeric expression which defines the limiting value of control var.

expr3   is a numeric expression which defines the increment added to control var each time the loop is executed.

(Block of statements)   are any statements which may also contain FOR - NEXT loops.

Remarks

General

1. A program loop begins with a FOR statement which provides the specifications for repetition, a block of statements which BASIC executes during each repetition of the program loop, and a NEXT statement which denotes the end of the loop.

   FOR statement
   (Block of statements)
   NEXT statement

2. The initial, limiting, and incremental values for control var determine the number of times the statements contained in a FOR - NEXT loop are to be executed. The loop is repeated until the value of the control var meets the termination condition.

Rules

1. Every FOR or NEXT statement must have a matching NEXT or FOR statement or an error message is printed. If a NEXT statement is executed without the corresponding FOR statement, and vice-versa, an error will occur.

2. control var must not be subscripted.

3. Expressions expr1, expr2, and *expr3* may have positive or negative values and *expr3* must not be zero.

   If you omit STEP *expr3* from the FOR statement, then *expr3* is assumed to be 1.

4. The termination condition for a FOR - NEXT loop depends on the values of expr1 and *expr3*. The loop terminates if: *expr3* is positive and the next value of control var is greater than expr2; *expr3* is negative and the next value of control var is less than expr2.

   If the value of expr1 (the initial value) meets the termination condition, then the loop is not performed even once. (See example 3.)

5. When the termination condition is met, the loop is exited; control var equals the first value not used in the loop.

6. Branching in and out of a FOR/NEXT loop is possible, but if you enter a loop at any point other than a FOR statement, upon encountering the corresponding NEXT statement an error will occur.

Program Loop Operation

1. The expressions expr1, expr2, and *expr3* are evaluated. If you omit *expr3*, it is assumed to be 1.

2. The control var is set to the value of expr1.

3. If *expr3* is positive and control var is greater than expr2, then the termination condition is satisfied and control passes to the statement following the corresponding NEXT statement. The value of control var then equals the first value not used in the loop; i.e., control var + *expr3*.

   If *expr3* is negative and control var is less than expr2, then the termination condition is satisfied and control passes to the statement following the corresponding NEXT statement. The value of control var then equals the first value not used in the loop; i.e., control var + *expr3*.

FOR and NEXT (continued)

Otherwise, the system performs the following steps:

4. BASIC executes the statements in the FOR - NEXT block.
5. When the corresponding NEXT statement is executed, control var is set to the value of control var + *expr3*.
6. Repeat step 3 (control passes to FOR statement).

Nesting Loops

You can nest FOR - NEXT loops to a depth specified by your system manager. The FOR statement and its terminating NEXT statement must be completely contained within the loop in which it is nested.

Legal Nesting

```
┌─FOR X = ...
│┌─FOR Y = ...
││┌─FOR Z = ...
││└─NEXT Z
│└─NEXT Y
└─NEXT X
```

Illegal Nesting

```
┌FOR X = ...
│┌FOR Y = ...
└┼NEXT X
 └NEXT Y
```

Examples

1.
```
*LIST)
 0010 FOR I=1 TO 9
 0020 NEXT I
 0030 PRINT I
*RUN)
 10
```
I (control var) equals first value not used in the loop.
```
END AT 0030
*
```

2.
```
*LIST)
 0040 FOR J=1 TO 9 STEP 3
 0050 NEXT J
 0060 PRINT J
*RUN)

END AT 0060
*10
```
Final value of J when terminating value (expr2) was exceeded.

3.
```
*LIST)
0010 FOR I=1 TO 3 STEP -1
0020 PRINT "SHOULD NOT ENTER HERE"
0030 NEXT I
0040 PRINT I
*RUN)
1
END AT 0040
*
```

AOS	
RDOS	✓
DOS	✓

S	✓
C	✓
F	

GDIR

prints the name of your directory.

Format

GDIR

Remarks

1. The knowledge of your directory name will be useful to other programmers who wish to create a link to any of your files which are CHATRed sharable (O).
2. You can write files to your own directory, read files from the library directory BASIC.DR, and read files from other directories which have the sharable attribute (O).

Examples

```
*GDIR)
JOE
```
All of your file references are to directory *JOE*.

AOS	√		S	√
RDOS	√		C	
DOS	√		F	

GOSUB and RETURN

GOSUB directs program control to the first statement of a subroutine. RETURN exits the subroutine and returns program control to the next statement following the GOSUB statement.

Format

GOSUB line no.

RETURN

Arguments

line no. is a program line number.

Remarks

1. A subroutine is a group of program statements which is entered via the GOSUB statement and exited via the RETURN statement. Instead of repeating the statements each time they are required, you write the statements into the program only once and access them by GOSUB statements. The RETURN statement returns control to the statement following the last executed GOSUB statement. In this manner, the program continues at the appropriate place after the subroutine has been executed.

2. A subroutine must always be entered by using a GOSUB statement. Otherwise, the RETURN-NO GOSUB error message is printed when the RETURN statement is executed.

3. You may use more than one RETURN statement in a subroutine if program logic requires the subroutine to terminate at one of a number of different places.

4. Although a subroutine may appear anywhere in a program, it is good practice to place the subroutine distinctly separate from the main program. To prevent inadvertant entry to the subroutine by other than a GOSUB statement, the subroutine should be preceded by a STOP statement or GOTO statement which directs control to a line number following the subroutine.

5. Subroutines may be nested to a depth specified by your system manager. Nesting occurs when a subroutine is called during the execution of a subroutine. Upon execution of the first RETURN statement, control passes to the statement immediately following the last executed GOSUB statement. The next RETURN statement causes control to pass to the next to last executed GOSUB statement, and so on.

Examples

1.
```
*LIST)
0010 LET A=6
0020 GOSUB 0100
0030 LET A=10
0040 GOSUB 0100
0050 STOP
0100 FOR I=1 TO A STEP 2
0110    PRINT I;
0120 NEXT I
0130 PRINT
0140 RETURN
*RUN)
1 3 5
1 3 5 7 9

STOP AT 0050
*
```

2.
```
*LIST)
0010 GOSUB 0040
0020 PRINT "EXAMPLE";
0030 STOP
0040 PRINT "NEST";
0050 GOSUB 0080
0060 PRINT "INE ";
0070 RETURN
0080 PRINT "ED ";
0090 GOSUB 0120
0100 PRINT "ROUT";
0110 RETURN
0120 PRINT "SUB";
0130 RETURN
*RUN)
NESTED SUBROUTINE EXAMPLE
STOP AT 0030
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

GOTO

unconditionally transfers control to the statement with the specified line number.

Format

GOTO line no.

Arguments

line no. is a program statement line number.

Remarks

1. If control passes to an executable statement, that statement and those following are executed.
2. If control passes to a nonexecutable statement (e.g., DATA), program execution continues at the first executable statement which follows the nonexecutable statement.
3. If line no. is not a line number in the program, an error will occur.

Examples

```
*LIST)
0010 READ X
0020 PRINT X
0030 GOTO 0010
0040 DATA 1,2,3,4,5
0050 DATA 20,21,23
0060 END
*RUN)
1
2
3
4
5
20
21
23
ERROR 15 AT 0010 - END OF DATA
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

HELP

displays information about each BASIC statement and command.

Format

HELP "verb"

Arguments

verb is the name of a statement, command, or function expressed as a string literal or string variable.

Examples

```
*HELP "DATA")

DATA TO PROVIDE VALUES FOR VARIABLES
        SPECIFIED IN A [MAT] READ STATEMENT.
```

Remarks

HELP "HELP" will display a list of all statements, commands, and functions that you can use for verb in a HELP statement or command.

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

IF -- THEN

executes a statement on the basis of whether an expression or a relational expression is true or false.

Format

$$\text{IF} \begin{Bmatrix}\text{rel-expr}\\ \text{expr}\end{Bmatrix} \begin{Bmatrix}\textit{[THEN]}\ \text{statement}\\ \text{THEN line no.}\end{Bmatrix}$$

Arguments

rel-expr is a relational expression as defined previously in Chapter 2.

expr is a numeric expression.

statement is any BASIC statement except FOR, NEXT, DEF, END, DATA, and REM.

Remarks

1. BASIC evaluates the relational expression, rel-expr. If it is true, then BASIC executes statement following the optional *THEN* or transfers control to line no. following the required THEN. If the expression is false, program execution continues at the next statement after the IF--THEN statement.

2. You can use a numeric expression (expr) instead of a relational expression (rel-expr). The numeric expression is considered false if it has a value of 0 and is true if it has a nonzero value.

NOTE: Since the internal representation of noninteger numbers, as described in Chapter 2, may not be exact (for example, .2 cannot be exactly represented), it is advisable to test for a range of values when testing for a noninteger. For example, if the result of a computation, A, was to be 1.0, then a reliable test for 1 is:

```
IF ABS (A-1.0)<1.0E-6 THEN...
```

If this test succeeds, then A is equal to 1 within 1 part in 10↑6. This is approximately the accuracy of single-precision floating point calculations.

```
*05  IF A=B THEN 100)
*10  IF A=B THEN GOTO 100)
*20  IF A=B GOTO 100)
*30  IF A-B < = 5 THEN C=0)
*40  IF A*B < 50 THEN GOSUB 300)
*50  IF A↑B > 100 GOSUB 400)
```

Lines 5, 10, and 20 are equivalent variations of the IF--THEN statement.

Examples

```
*LIST)
0005 REM-START
0010 LET N=10
0020 INPUT "X= ",X
0030 IF X THEN GOTO 0050
0040 GOTO 0100
0050 IF X > =N THEN GOTO 0080
0060 PRINT X, "X IS LESS THAN 10"
0070 GOTO 0020
0080 PRINT X, "X IS GREATER OR EQUAL TO 10"
0090 GOTO 0020
0100 PRINT X, "X=0"
0110 END
*RUN)
X= 5
5       X IS LESS THAN 10
X= 7
7       X IS LESS THAN 10
X= 12
12      X IS GREATER OR EQUAL TO 10
X= 10
10      X IS GREATER OR EQUAL TO 10
X= 0
0       X=0

END AT 0110
*
```

Note the nested IF statement in the following example.

```
*LIST)
0010 LET X=5
0030 IF X=5 THEN IF A$="C" THEN PRINT "C"
0040 END
*RUN)
C

END AT 0040
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

INPUT

requests data from your terminal and assigns the values you supply to a list of variables.

Format

INPUT *["str lit",]* $\left\{\begin{matrix}\text{svar}\\ \text{var}\end{matrix}\right\} \left[\left\{\begin{matrix}\textit{svar}\\ \textit{var}\end{matrix}\right\}\right]$ *... [;]*

Arguments

var and svar are numeric and string variables separated by commas.

str lit is a message or prompt.

Remarks

1. You can use the INPUT statement to enter numeric data, string data, or both to a program.
2. When an INPUT statement is executed, a question mark is output as an initial prompt unless the INPUT statement contains the *str lit* option. Then the *str lit* is output as an initial prompt.
3. You respond by typing a list of data, separating each datum from the next by a comma or a carriage return. The list is terminated with a carriage return.
4. If you terminate the data list with a carriage return before a value has been supplied for each variable in the INPUT statement, then a question mark will be output as a prompt, indicating that you must supply more data.
5. The data that you input in response to the prompt must be of the same type (numeric or string) as the variable in the INPUT statement list for which the data is being supplied. Variables in the INPUT statement list may be subscripted array elements, scalars, or strings.
6. If the data you input from the terminal does not match the type of a variable in the INPUT statement list, then a \? is output to the terminal for the data in error.
7. If you terminate the INPUT statement variable list with a semicolon, then the cursor is left following the last input data item. Otherwise, a carriage return-line feed is output. The optional semicolon terminator has no function in an AOS system, except to make the INPUT statement compatible in syntax with that of its counterpart in an RDOS system.
8. Numeric variables may include digits, plus and minus signs, decimal points, and the letter E (exponential notation).
9. If you use commas to delimit the data list and you supply more items than there are variables in the INPUT list, an error condition occurs. The values you supplied will be assigned to the variables in the list and the excess will be ignored.

Examples

1.
```
*LIST)
0005 INPUT A,B,C,D,E
0010 PRINT A+B,C+D,D+E
*RUN)
? 1,2,3,4,5)
3  7  9

END AT 0010
*
```

2.
```
*LIST)
0010 INPUT "A,B,C,D,E= ",A,B,C,D,E
0020 PRINT A+B,C+D,D+E

*RUN)
A,B,C,D,E= 1,2) ? 3,4,5)
3  7  9

END AT 0020
*
```

3.
```
*LIST)
0010 INPUT A,B,C;
0020 PRINT "NO RETURN"
*RUN)
? A)\? 1,2,3) NO RETURN

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

LET

evaluates expr and assigns the resultant value to var or svar.

Format

[LET] $\begin{Bmatrix}\text{var}\\\text{svar}\end{Bmatrix}$ = expr

Arguments

var and svar are numeric or string variable names.

expr is an arithmetic or string expression.

Remarks

1. The mnemonic *LET* is optional.
2. The variable var or svar may be subscripted.
3. String expressions may be assigned to string variables.
4. In AOS systems, more than one variable can be assigned a value by using multiple LET, valid as either a statement or a command. See line 40 in the example.

Examples

```
*10 LET A=A+1)
```
(Variable A is assigned a value one greater than it was before.)

```
*20 A(2,1) = B↑2+10)
```
(The element in row 2, column 1 of array A is assigned the value of expression B↑2+10.)

```
*30 A$=B$,C$)
```
(A$ is assigned the concatenated value of B$ and C$.)

```
*40 LET A=B=C=2)
```
(For AOS only.
Multiple LET assigns the value 2 to A, B and C.)

```
*50 LET A$=B$+C$)
```
(String arithmetic.)

AOS	√	S	√
RDOS		C	√
DOS		F	

LIBRARY

prints the filenames in the directory specified that match the template.

Format

LIBRARY [{ *"directory", "template"* / *"directory"* / *"template"* }]

Arguments

directory is any legal directory pathname starting from the root (:), expressed as a string literal or string variable.

template is any combination of up to 15 valid characters, including asterisk (*), dash (-), and plus sign (+), in accordance with AOS template rules. The template must be expressed as a string literal or string variable.

Remarks

1. If you omit both the directory and template options, then BASIC will print a list of all files in the BASIC library directory (:BASIC).
2. If you omit the directory option and specify a template, BASIC will print a list of all files in the library directory that match the template.
3. If you omit the template option and specify a directory, BASIC will print a list of all files in the directory specified.
4. BASIC provides the following information with each filename printed:
   - the type of file
   - the date last modified
   - the time last modified
   - the size of the file (in bytes)

Examples

```
*DIM A$(30))
*A$=":UDD:XBASIC")
*LIBRARY A$,"-TAPE-.CLI")
ROOMTAPE.CLI       TXT 11-APR-77 07:29:06 263
RELEASETAPE0100.CLI
                   TXT 11-APR-77 07:28:36 368
ROOMTAPE0100.CLI   TXT 11-APR-77 07:29:44 407
*
*A$=":UDD:XBASIC:DUDLEY")
*LIBRARY A$)
HOBURG             UDF 05-APR-77 08:51:42 106532
MARATHON.BASIC     UDF 19-FEB-77 15:14:36   1942
```

AOS			S	√
RDOS	√		C	√
DOS	√		F	

LIBRARY

prints all filenames in the library directory.

Format

LIBRARY

Remarks

One filename is printed per print zone.

Examples

```
*LIBRARY)
A1.          SHOT.SR      SQRT.SR
BACKGAMMON   SUPERGUESS   STOCKS.SR
CASINO.SR    SWAP.SV      SNOOP.
COMPILER.SR  FCOM.CM      BLACKJACK.SR
BANK.SR      FOOTBALL.SR  BILLBOARD.SR
KILLER.MS    K2.          MAT.SR
SNOOPY.SR    GUESS.SR     QUEEN.SR
TEST1.       HORSERACE.SR LUNAR.SR
BATNUM.SR    HEMAN.SR     SHOT1.SR
FISCAL.SR    HELLO.SR     HELLO.SV
FISCAL.BT
*
```

AOS	√	S	
RDOS	√	C	√
DOS	√	F	

LIST

outputs part or all of your current program in ASCII to the disk file or device specified by filename, or to your terminal if filename is not specified.

Format

$$\text{LIST}\left[\left\{\begin{matrix} \textit{line n1} \\ \begin{Bmatrix}\textit{TO}\\ ,\end{Bmatrix}\textit{line n2} \\ \textit{line n1}\begin{Bmatrix}\textit{TO}\\ ,\end{Bmatrix}\textit{line n2}\end{matrix}\right\}\right]\;[\text{"}\textit{filename}\text{"}]$$

Arguments

line n1 is the first statement to be listed.

line n2 is the last statement to be listed.

filename is a disk file or device expressed as a string literal.

Remarks

1. You can use the LIST command in the following four ways:

LIST)	List the entire program from the lowest numbered statement.
LIST n1)	List only the single statement at line number n1.
LIST $\begin{Bmatrix}\text{TO}\\ ,\end{Bmatrix}$ n2)	List from the lowest numbered line through line number n2.
LIST n1 $\begin{Bmatrix}\text{TO}\\ ,\end{Bmatrix}$ n2)	List from line numbers n1 through line number n2.

2. When you include the *filename* argument, the LIST command writes the specified lines to the disk file or device called *filename* in ASCII format.

3. If *filename* is a disk file that already exists in your directory, BASIC will print the message:

   TYPE CR TO DELETE (RDOS/DOS)

   TYPE NL TO DELETE (AOS)

   This message lets you confirm whether or not you would like to delete the existing *filename* and replace the file with the lines you specified in the LIST command. If you type a carriage return (new-line), BASIC accepts the replacement. Type anything preceding the carriage return (new-line), and you cancel your LIST command.

4. The file created by the LIST command can be read back into the program storage area by the ENTER or NEW commands.

Examples

```
*LIST)
```

Lists your current program on your terminal.

```
*LIST "$LPT")
```

Outputs your current program to the line printer.

```
*LIST 20)
```

Lists line number 20 on your terminal.

```
*LIST 700,9999)
```

Lists line numbers 700 through 9999 at the terminal.

```
*LIST "TEST.SR")
```

Outputs your current program in ASCII to your directory with the filename, TEST.SR, and replaces any previous file with that name, provided you respond with a) to the confirmation prompt.

```
*LIST 100, 200 "TEMP")
```

Lists line numbers 100 through 200 to disk file TEMP; TEMP replaces any previous file of that name, provided you respond with a) to the confirmation prompt.

AOS	√	S	
RDOS	√	C	√
DOS	√	F	

LOAD

loads a program into your program storage area that was previously SAVEd in core image format.

Format

LOAD "filename"

Arguments

filename   is the name of a core image file created by a previous SAVE command.

Remarks

1. The LOAD command executes an implicit NEW command (clearing the storage area) and then reads filename into core.
2. filename may be on disk or it may also be on a binary input device such as the paper tape reader.
3. If filename is a disk file, a search is made for filename in your directory first. If it is not found, a search is made in the library directory for filename.
4. After you have LOADed filename, it can be LISTed, modified, or RUN.

Examples

```
*LOAD "$PTR")
*LOAD "MATH3")
*LOAD "MT0:1")
```

AOS	√	S	√
RDOS	√	C	√
DOS	√	F	

LREAD

reads a string from the terminal, terminated by either a null, form-feed, or carriage return (new-line).

Format

LREAD *["str lit",]* svar *[,svar^1]*

Arguments

str lit   is a message or prompt.

svar   is a string variable which is assigned the value of the string read from your terminal.

svar^1   is a string variable which is assigned the value of the delimiter for the string read. Valid delimiters are null, form-feed, and carriage return (new-line).

Remarks

1. When BASIC executes an LREAD statement, it outputs a question mark to your terminal as an initial prompt, unless the LREAD statement contains the *str lit* option; then it uses *str lit* as a prompt.
2. The maximum string length allowed for svar is 133, which includes the delimiter. If your response to the LREAD prompt is longer than 133 characters, the length of *svar^1* will be set to 0 and the value of svar will be terminated at 133 characters.
3. If your response to the LREAD prompt is shorter than 133 characters, then the length of *svar^1* is equal to 1 and *svar^1* will contain the delimiter.

Examples

```
*LREAD A$,B$)
? ABCEDF
*PRINT A$)
ABCEDF
*PRINT B$)

*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

LWRITE

writes a string to your terminal.

Format

LWRITE svar *[,svar¹]*

Arguments

svar is a string variable whose value is written to your terminal.

svar¹ is a string variable which is the value of the delimiter for the string written.

Remarks

1. If the argument list includes *svar¹*, and its length is 1, then BASIC assumes *svar¹* is one of the valid delimiters and outputs it as the string terminator.
2. If the length of *svar¹* is 0, then no delimiter will be output.
3. If the argument list does not include *svar¹*, then a null will be output as the string terminator.
4. LWRITE allows direct output of control characters to your terminal. For example, LWRITE does not insert a line feed after a carriage return, and can be made not to output extraneous string terminators.

Examples

```
*A$="ABCDEFGH")
*A1$="")
*LWRITE A$,A1$)
ABCDEFGH*
*LWRITE A$)
ABCDEFGH*
```

AOS	√
RDOS	
DOS	

S	√
C	√
F	

MSG

transmits a message from your terminal to another programmer or to the system operator.

Format

$$\text{MSG} \begin{Bmatrix} \text{pid} \\ \text{"processname"} \\ \text{"console name"} \end{Bmatrix} ,\text{"message"}$$

Arguments

pid is a process identification.

processname is a process name expressed as a string literal or string variable.

console name is a console identification (e.g., @CON1).

message is the text of message, expressed as a string variable or string literal.

Remarks

1. If a receiving programmer is not on line, then the transmission will fail, and BASIC will print an error message at your terminal.
2. If your transmission succeeds, then BASIC will print the following at the receiving programmer's terminal:

   FROM PID XXX: message

   where XXX is your pid number and message is the text of your message.
3. Message length is limited to one line per MSG command.
4. Quotation marks are necessary if you express message as a string literal.

Examples

```
*MSG 11,"RSVP")
*
```

AOS	
RDOS	√
DOS	√

S	√
C	√
F	

MSG

transmits a message from your terminal to another programmer or to the operator.

Format

MSG userID message

Arguments

userID   is your identification.

message   is the text of message.

Remarks

1. The system operator's userID is: OPER
2. If a receiving programmer is not on line, the transmission will not succeed and BASIC will print an error message at your terminal.
3. If a receiving programmer has disabled message reception by using the CHAR command with the MSG argument, the transmission will not succeed and BASIC will print an error message.
4. If your transmission succeeds, then BASIC will print the following at the receiving programmer's terminal:

   FROM sendersID: message

   where *sendersID* is your identification and *message* is the text of your message.
5. Message length is limited to one line per MSG command.
6. Quotation marks are not necessary for message.

Examples

If userID JACK types:

```
*MSG OPER MOUNT MY CASSETTE-THANKS)
```

The master console receives:

```
FROM JACK: MOUNT MY CASSETTE-THANKS
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

NEW

clears the program and variables currently stored in your program storage area and closes any open files.

Format

NEW *["filename"]*

Arguments

filename   is a string literal or string variable for a SAVED file.

Remarks

1. You must clear your storage area with a NEW command (or statement) before entering a new program, to avoid intermixing lines from previous programs with the new program.
2. If you make NEW an executable statement in a program, the program will clear itself from memory when NEW is executed, and no STOP or END message will be issued.
3. You can combine the ON ESC or ON ERR statements with NEW to prevent unauthorized access to a program.
4. NEW closes any files left open by previously executed programs.
5. The variation NEW *"filename"* is equivalent to the following pair of statements (or commands):

   ```
   NEW)
   ENTER "FILENAME")
   ```

   The NEW command will clear your storage area even if *filename* doesn't exist.

Examples

```
*LIST)
0100 READ A,B,C,D
0110 LET E=A*23
0115 LET F=C*A
0120 PRINT E;F
0130 NEW
0135 DATA 1,2,3,4
*RUN)
23 3
*LIST)
ERROR 05 - LINE NUMBER
*
```

AOS	√		S	√
RDOS	√		C	
DOS	√		F	

ON ERR THEN

directs your program to an error handling routine in your program instead of the BASIC system error handler.

Format

ON ERR $\left\{\begin{array}{l}\text{THEN line no.}\\ \textit{[THEN]}\text{ statement}\end{array}\right\}$

Arguments

statement is any BASIC statement except those listed under *Remarks.*

line no. is a program statement line number.

Remarks

1. The following BASIC statements cannot be used in statement:

   FOR
   NEXT
   DEF
   END
   DATA
   REM

2. Normally, when a BASIC error occurs, any operation in progress is interrupted, an error message is printed at your terminal and the terminal is placed in interactive mode. If an ON ERR THEN statement is encountered during execution of your program, any subsequent error will cause the statement position of ON ERR THEN statement to be executed, or will cause control to be transferred to line no.

3. Place the ON ERR statement at the beginning of your program if you want all system errors handled by your error routine. Place the ON ERR statement anywhere else in your program and your error routine will be executed only for errors which occur after the ON ERR statement is encountered.

4. If statement is a GOSUB then, after the subroutine is finished (RETURN), control passes back to the statement following the one on which the error occurred. A RETRY statement should not be used in the body of the subroutine.

5. If statement is any statement other than STOP, GOTO, or GOSUB, then the statement portion is executed and program control passes back to the statement following the one on which the error occurred.

6. You can restore the normal handling of errors by including the following statement in an appropriate place in your program:

   ON ERR THEN STOP

7. The keyword *THEN* is optional if a statement is specified; it is required if you specify line no.

Examples

```
0010  ON ERR THEN GOTO 1000
0020  OPEN FILE (0,0), "X"
0030  ON ERR THEN STOP
  .
  .
  .
1000  OPEN FILE (0,0), "Y"
1010  GOTO 30
```

AOS	√
RDOS	√
DOS	√

S	√
C	
F	

ON ESC THEN

directs your program to an ESCape handling routine in your program instead of the BASIC system ESCape handler.

Format

ON ESC $\left\{\begin{array}{l}\text{THEN line no.}\\ \textit{[THEN]}\text{ statement}\end{array}\right\}$

Arguments

statement is any BASIC statement except those listed under *Remarks.*

line no. is a program statement line number.

Remarks

1. The following BASIC statements cannot be used in statement:

   FOR
   NEXT
   DEF
   END
   DATA
   REM

2. Normally, when the ESC key is pressed, any operation in progress is interrupted and the terminal is placed in interactive mode to wait for your next command. If an ON ESC THEN statement line is encountered during execution of a program, pressing the ESC key will cause the statement portion of ON ESC THEN statement to be executed.

3. Place the ON ESC statement at the beginning of your program if you want all ESCapes to be handled by your own routine. Place the ON ESC statement anywhere else in your program and your routine will be executed only for ESCapes which occur after the ON ESC statement is encountered.

4. If statement is a GOSUB then, after the subroutine is finished (RETURN), control passes back to the point of interruption.

5. Since ESC is not an error condition, the RETRY statement should never be used in association with an ON ESC statement.

6. If statement is any statement other than STOP, GOTO, or GOSUB, then the statement portion is executed and program control passes back to the point of interruption.

7. You can restore the normal handling of ESCape by including the following statement in an appropriate place in your program:

   ON ESC THEN STOP

8. The keyword *THEN* is optional if a statement is specified; it is required if you specify line no.

Examples

```
0100 ON ESC THEN PRINT X,Y,Z
.
.
.
0140 PRINT X
0141 Y=Z
.
.
.
```

In this example, when you press the ESCape key during program execution, control passes to the statement on line 100 and the values of X, Y, and Z are printed. After line 100 is executed, the program continues from the point of interruption. Therefore, if line 140 had been completed when ESC was pressed, line 100 would be executed followed by line 141.

```
0010 ON ESC THEN GOSUB 0500
0020 DIM X(2500)
0021 LET A=0
0022 LET B=0
0023 LET C=0
0030 FOR I=1 TO 2500
0040    LET X(I)=A*I↑2+B*I+C
0050 NEXT I
0060 STOP
0500 PRINT I,X(I)
0510 INPUT "CONTINUE (0), NEW INPUT (1)",D
0520 IF D=0 THEN RETURN
0530 INPUT "NEW VALUES FOR A,B,C= ",A,B,C
0540 RETURN
```

In this example, a RETURN from line number 520 or 540 positions you to the line after the last executed line when the ESC key was pressed, not to line 20.

AOS	√	S	√
RDOS	√	C	
DOS	√	F	

ON-GOTO and ON-GOSUB

transfers control to one of several lines in a program depending on the value of an expression at the time the statement is executed.

Format

ON expr {GOTO | GOSUB} line no. *[,line no.]* ...

Arguments

expr is a numeric expression which is evaluated to an integer.

line no. is a list of line numbers in the current program whose positions in the argument list are numbered from 1 through n.

Remarks

1. The expression expr is evaluated and if it is not an integer, the fractional portion is ignored.
2. The program transfers control to the line number whose position in the argument list corresponds to the computed value of expr.
3. If expr evaluates to an integer that is greater than the number of entries given in the argument list or that is less than or equal to zero, the ON statement is ignored and control passes to the next statement.
4. The ON-GOSUB statement must contain an argument list whose entries are the first line of subroutines within the current program.

Examples

```
*10  ON M-5 GOTO 500, 75, 1000)
```

If M-5 evaluates to 1, 2 or 3 then control passes to statement 500, 75, or 1000, respectively. If M-5 evaluates to any other value, control passes to the next statement in the program.

```
*10  ON (SGN(M-5)+2) GOTO 100, 200, 300)
```

This statement is equivalent to the following three statements.

```
*10  IF M-5<0 THEN GOTO 100)
*20  IF M-5=0 THEN GOTO 200)
*30  IF M-5>0 THEN GOTO 300)
```

AOS	√	S	√
RDOS	√	C	√
DOS	√	F	

PAGE

sets the right margin of your terminal.

Format

PAGE=expr

Arguments

expr is an arithmetic expression evaluating to a number between 15 and 132, inclusive, and not less than the current TAB setting. For AOS systems, the value of expr evaluates to a number between 8 and N, where N can be as high as 255 depending on the system.

Remarks

1. BASIC uses a default value of 80 as the maximum line width.
2. Strings may be as large as 32767 bytes. However, if a string is longer than the PAGE value it cannot be printed and an error message is output. To examine the entire string you must print substrings of lengths smaller than or equal to the PAGE value.

Examples

```
*LIST)
0010 PAGE=30
0020 FOR I=1 TO 25
0030    PRINT I;
0040 NEXT I
*RUN)

1  2  3  4  5  6  7  8  9

10  11  12  13  14  15  16

17  18  19  20  21  22  23

24  25
END AT 0040
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

PRINT

performs print operations at your terminal.

Format

$$\left\{\begin{array}{l} ; \\ \text{PRINT}\end{array}\right\} \left[\left\{\begin{array}{l}\textit{svar}\\ \textit{expr}\\ \textit{``str lit''}\end{array}\right\} \left[\left\{\begin{array}{l}, \\ ;\end{array}\right\}\left\{\begin{array}{l}\textit{svar}\\ \textit{expr}\\ \textit{``str lit''}\end{array}\right\}\right]\ldots\right]\left[\left\{\begin{array}{l}, \\ ;\end{array}\right\}\right]$$

Arguments

; (semicolon) is a synonym for the keyword PRINT.

svar is a string variable.

expr is a numeric or string expression.

str lit is a message or prompt.

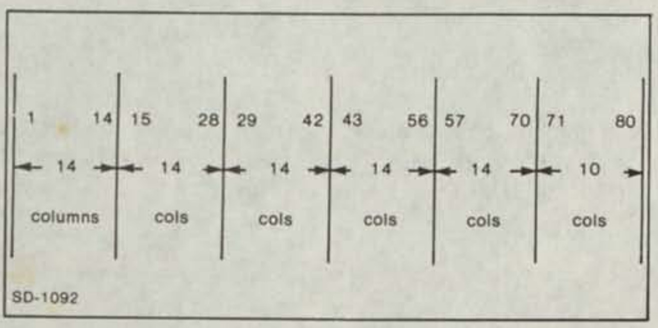
Remarks

The following PRINT operations are possible:

1. Print the result of a computation.
2. Print verbatim the characters in a string literal or string variable.
3. Print a combination of operations 1 and 2.
4. Print a blank line (skip a line).

Zone Spacing of Output

The print line on a terminal is divided into print zones. The width of a print zone is determined by the TAB statement. The default value for TAB is 14 and is used in the following examples. The first column on a line is column 1.



A comma(,) between items in the PRINT statement list causes the next item to be printed in the leftmost position of the next printing zone. If there are no more printing zones on the current line, printing continues in the first printing zone on the next line. If an item requires more than one print zone, the next item in the list is printed in the next free print zone. (See example 1.)

Before each item is printed its length is compared with the space remaining on the line. If insufficient space remains on the current line, the item is moved to the next line. If the length of the item is greater than the width of the page (see PAGE statement), then BASIC issues an error message.

Compact Spacing of Output

A semicolon (;) between items in the PRINT statement list causes the next item to be printed at the next character position. Note that a space is reserved for the plus (+) sign even though it is not printed, (see example 2) and there is always a trailing space.

Spacing to the Next Line

When the last item in a print list has been printed, BASIC outputs a carriage return and line feed unless the last item in the list is followed by a comma (,) or semicolon (;). In this case, the carriage return and line feed are not output and the next item is printed on the same line according to the comma or semicolon punctuation. (See example 3.)

If, however, the comma or semicolon would cause printing of the next item to occur beyond the allowable line width (see PAGE statement), a carriage return and line feed are output.

Printing Blank Lines

A PRINT statement with no list of print items or punctuation will output a carriage return and line feed. (See example 4.)

For more printing versatility, you can use the TAB (X) function, the TAB=, the PAGE=, and the PRINT USING statements.

Examples

```
*LIST
0010 LET X=25
0020 PRINT "SQUARE ROOT OF X IS: ",SQR(X)
*RUN
SQUARE ROOT OF X IS:        5

END AT 0020
*

↑             ↑             ↑
1             15            29
```

(column positions)

2.

```
*LIST
0010 LET X=5
0020 PRINT X;(X*2)↑6;X*2;(X*2)↑4;
0030 PRINT X-25;(X*2)↑8;X-100
*RUN
 5 1000000 10 10000-20 1E+08
-95

END AT 0030
*

↑ ↑       ↑  ↑     ↑  ↑
1 3       11 14    20 23
```

(column positions)

3.

```
*LIST
0005 PAGE=70
0010 LET X=5
0020 PRINT X,(X*2)↑6,
0030 PRINT X↑4
0040 PRINT "FIN"
*RUN
 5            1000000        625
FIN

END AT 0040
*

↑            ↑              ↑
1            14             29
```

(column positions)

4.

```
*LIST
0010 LET X=5
0020 PRINT X;(X*2)↑6,X*2
0030 PRINT X-25;(X*2)↑8
0040 PRINT X-100
0050 PRINT
0060 PRINT "DONE"
*RUN
 5 1000000      10
-20 1E+08
-95

DONE

END AT 0060
*

↑  ↑          ↑
1  4          15
```

(column positions)

In line 20, the comma and semicolon spacing characters are both used. Line 50 outputs a blank line before printing "DONE".

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

PRINT USING

outputs the values of expressions in the PRINT USING statement list using the format specified.

Format

PRINT USING format, expr $\left\{\begin{bmatrix} ;expr \\ ,expr \end{bmatrix}\right\}\cdots$

Arguments

format is a string literal or string variable which specifies the format for printing the items in the expr list. See *Remarks.*

expr is a numeric or string expression.

Remarks

1. All normal PRINT formatting conventions for TAB, the comma, and semicolon are ignored in a PRINT USING statement, except when you use a semicolon or comma at the end of a statement to inhibit a carriage return.

2. The format expression may have more than one format field and may include string literals as well as the following special characters which are used for formatting numeric output.

   #
   .
   +
   -
   $
   , (comma)
   ↑

   a. *Digit Representation by Number Sign (#)*

   For each # in the format field, a digit (0 to 9) is substituted from the expr argument. Please refer to Table 3-1.

   NOTE: In the following descriptions, a box (□) is used to clarify the presence of a blank space.

Table 3-1. Representing Digits by Number Sign

format	expr	BASIC Outputs	Remarks
####	25	□□25	Right-justify digits in field with leading blanks.
####	-30	□□30	Signs and other nondigits are ignored.
####	1.95	□□□2	Only integers are represented; the number is rounded to an integer.
####	598745	****	If the number in expr has more digits than specified by format, then all asterisks are output.

b. *Decimal Point (.) Representation*

The decimal character (.) is used to place a decimal point in the fixed position in which it appears in format. Digit (#) positions which follow the decimal point are filled; no blank spaces are left in these digits positions. When expr contains fewer fractional digits than specified by format, zeros are output to fill the positions. When expr contains more fractional digits than format allows, the fraction will be rounded out to the limits of format. Please refer to Table 3-2.

c. *Fixed Sign (+ or -) Representation*

A fixed sign character appears as a single plus (+) sign or minus (-) sign in either the first or last character position in the format field.

A fixed plus (+) sign is used to print the sign (+ or -) of expr in the position in which the fixed plus (+) sign is placed in format.

A fixed minus (-) sign is used to print a minus (-) sign for negative values of expr or a blank space for positive values of expr in the position in which the fixed minus (-) sign is placed in format.

When a fixed sign is used, any leading zeros appearing in expr will be replaced by blanks, except for a single leading zero preceding a decimal point. Refer to Table 3-3.

Table 3-2. Decimal Point Representation

format	expr	BASIC Outputs	Remarks
####.##	20	□□20.00	Fractional digit positions are filled with zeros.
####.##	29.347	□□29.35	Rounding occurs on fractions.
####.##	789012.34	*******	When expr has too many significant digits to the left of a decimal point, a field of all asterisks is output.

Table 3-3. Fixed Sign Representation

format	expr	BASIC Outputs	Remarks
+##.##	20.5	+20.50	Fractional digit positions are filled with zeros.
+##.##	1.01	+□1.01	Blanks precede the number.
+##.##	-1.236	-□1.24	Rounding out occurs on fractions.
+##.##	-234.0	*******	Too many digits to the left of the decimal point.
###.##-	20.5	□20.50□	Decimal digit positions are filled.
###.##-	000.01	□□0.01□	One leading zero is printed before the decimal point.
###.##+	1.236	□□1.24+	
###.##-	-234.0	234.00-	

Table 3-4. Floating Sign Representation

format	expr	BASIC Outputs	Remarks
---.##	-20	-20.00	Second and third minus signs are treated as # signs on output.
---.##	-200	******	Too many digits in expr to left of decimal point.
+++.##	2	□+2.00	Blanks between sign and digit are suppressed.
---.##	2	□□2.00	

d. *Floating Sign (++ or --) Representation*

A floating sign appears as two or more plus (++) or minus (--) signs at the beginning of the format field.

The floating plus (++) sign prints a plus or minus sign immediately before the value of expr with no separating blank spaces as would occur with fixed signs. A floating minus (--) prints either a minus or blank (for plus) immediately preceding the value.

When you use floating signs, the second and subsequent signs in format are treated as number signs (#); BASIC will replace them with numbers from expr as necessary. Please refer to Table 3-4.

NOTE: A format may include either a floating sign (plus or minus) or a floating $ sign (described in paragraph f.), but not both.

e. *Fixed Dollar Sign ($) Representation*

When you use a dollar sign ($) as either the first or second character in the format field, a dollar sign ($) is printed in that position. If the dollar sign ($) is in the second position, it must be preceded by a fixed sign (+ or -). A fixed dollar sign ($) causes leading zeros in the value of expr to be replaced by blanks. Refer to Table 3-5.

f. *Floating Dollar Sign ($$) Representation*

A floating dollar sign appears as two or more dollar signs ($$) beginning at either the first or second character in the format field. If the dollar signs ($$) start in the second position, they must be preceded by a fixed sign (+ or -).

When you use a floating dollar sign ($$), a dollar sign is printed immediately before the first digit of the expr value. Refer to Table 3-6

NOTE: A format may include either a floating dollar sign ($$) or a floating sign (plus or minus), described in preceding paragraph d, but not both.

g. *Separator (,) Representation*

When you use a comma separator (,) a comma is printed in the fixed position in which it appears in a string of digits (#) in the format field.

If a comma would be output in a field of suppressed leading zeros (blanks), then a blank space is output in the position for the comma. Refer to Table 3-7.

Table 3-5. Fixed Dollar Sign Representation

format	expr	BASIC Outputs
-$###.##	30.512	□$□30.51
$###.##+	-30.512	$□30.51-

Table 3-6. Floating Dollar Sign Representation

format	expr	BASIC Outputs	Remarks
+$$$#.##	13.20	+□□$13.20	Extra $ signs may be replaced by digits as with floating + and - signs.
$$##.##	-1.00	□$01.00-	Leading zeros are not suppressed in the # part of the field.

Table 3-7. Separator Representation

format	expr	BASIC Outputs	Remarks
+$#,###.##	30.6	+$□□□30.60	Space printed for comma.
+$#,##.##	2000	+$2,000.00	
++##,###	00033	□+00,033	Comma is printed when leading zeros are not suppressed.

h. *Exponent (↑) Representation*

Four consecutive up-arrows (↑↑↑↑) are used to indicate an exponent field in format. The four up-arrows will be output at E+nn, where each n is a digit.

If the exponent field in format does not have exactly four up-arrows, then a runtime error will result. Refer to Table 3-8.

Table 3-8. Exponent Representation

format	expr	BASIC Outputs
+##.##↑↑↑↑	170.35	+17.04E+01
+##.##↑↑↑↑	-.2	-20.00E-02
++##.##↑↑↑↑	6002.35	+600.24E+01

3. A format expression may include more than one format field and may include string literals in addition to the special formatting characters. Values of the expr argument list are sequentially assigned to format fields.

BASIC differentiates format fields from string literals by the characters that appear in format fields.

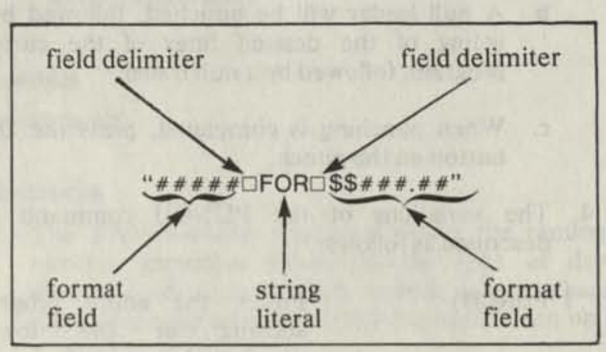
For example:

"TWO FOR $1.25"	$1.25 is part of the string literal.
"TWO FOR $$$.##"	$$$.## is a format field in the format expression.
"ANSWER IS -85"	-85 are characters of the string literal.
"ANSWER IS -###"	-### is a format field in the format expression.

4. A format expression may be specified by referring to a previously defined string variable, for example:

```
*05 DIM S$(10))
*10 LET S$="##.##")
*20 PRINT USING S$, 1.5, 2)
```

5. The format fields in a format expression must be delimited from each other by using any nonspecial formatting character after each format field. However, if the format expression in the PRINT USING statement is a string literal, then quotation marks (" ") cannot be used as a field delimiter. Delimiters are treated as string literals and are printed on output.



6. String literals may appear in the expr argument list of the PRINT USING statement and will be superimposed on a format field in the following manner:

   a. Each character of the string literal replaces a single format field character, which may be any of the special format characters ($, #,↑, +,-,., and ,).

   b. Strings are left-justified in the format field, and filled with spaces, if necessary.

   c. If the number of characters in the string is greater than the number of characters in the format field, then the string will be truncated to fit the field. For example,

```
*5 PRINT USING "#,###.##","TEST","CHARACTER")
*RUN
TEST□□□□ CHARACTE
```

7. When there are more items in the expr argument list than format fields in the format expression then the format fields will be used repetitively.

   In the following format expression, there are three format fields and two string literal fields. The string literal fields delimit the format fields.

   "####□@$###.##□PER□###"

   The first, fourth, seventh, etc., items in the expr argument list will be formatted using the format field, ####.

   The second, fifth, eighth, etc., items in the expr argument list will be formatted using the format field, $###.##.

   The third, sixth, ninth, etc., items in the expr argument list will be formatted using the format field, ###.

```
.
.
.
0100  PRINT USING "A(#)□=□##.#",I,A(I)
.
.
.
RUN)
A(1)□ = □ 17.9
```

   (Possible output includes two format fields and two string literals.)

```
.
.
.
0100  PRINT USING "###.##□",I,A,B
.
.
.
RUN)
□□ 1.00□□ 17.90□□ 25.77
```

   (Possible output with format expression repeated for each item in argument list.)

8. When the number of characters on a line exceeds the page size, printing continues on the next line.

AOS			S	
RDOS	√		C	√
DOS			F	

PUNCH

outputs part or all of the current program in ASCII to the terminal punch.

Format

$$\text{PUNCH}\left[\left\{\begin{array}{l} \textit{line n1} \\ \left\{\begin{array}{l}\textit{TO}\\ ,\end{array}\right\}\ \textit{line n2} \\ \textit{line n1}\ \left\{\begin{array}{l}\textit{TO}\\ ,\end{array}\right\}\textit{line n2}\end{array}\right\}\right]$$

Arguments

line n1 is the first statement to be punched.

line n2 is the last statement to be punched.

Remarks

1. A leader of null characters precedes the punched listing and a trailer of null characters follows the listing.

2. The number of null characters punched as leader and trailer equals the number defined as the page width (see PAGE command). This represents eight inches of leader for an 80-character line.

3. The PUNCH command does not turn on the terminal punch. The following procedure is required:

   a. Type the desired PUNCH command followed by a carriage return and immediately press the ON button on the terminal punch.

   b. A null leader will be punched, followed by a listing of the desired lines of the current program, followed by a null trailer.

   c. When punching is completed, press the OFF button on the punch.

4. The variations of the PUNCH command are described as follows:

PUNCH)	Punch the entire program starting at the lowest numbered statement.
PUNCH n1)	Punch only the single statement at line number n1.
PUNCH $\left\{\begin{array}{l}\text{TO}\\ ,\end{array}\right\}$ n2)	Punch from the lowest numbered line through line number n2.
PUNCH n1 $\left\{\begin{array}{l}\text{TO}\\ ,\end{array}\right\}$ n2)	Punch from line number n1 through line number n2.

Examples

```
*PUNCH 200 TO 500)
```

Punch line numbers 200 through 500 of the current program.

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

RANDOMIZE

causes the random number generator to start at a different point in the sequence of random numbers generated by the RND function.

Format

RANDOMIZE

Remarks

1. The RANDOMIZE statement resets the random number generator based on the time of day, thereby producing different random numbers each time a program using the RND function is run on a given day.

2. Without RANDOMIZE, the same sequence of random numbers is generated by the RND function each time a program is RUN. This feature is useful for debugging programs. When the program has been found to run successfully, the RANDOMIZE statement should be included in the program before the first occurrence of a RND function if you desire different start points in the sequence.

Examples

This program will print a different value each time it is run.

```
*10  RANDOMIZE)
*20  FOR I=1 TO 3)
*30      PRINT RND(0);)
*40  NEXT I)
*RUN)
.619604 .298047 .698036
END AT 0040
*RUN)
.776468 7.84348E-02 .603916
END AT 0040
*RUN)
.784302 .117651 .800002
END AT 0040
*RUN)
.956853 .98038 9.41276E-02
END AT 0040
*RUN)
.10981 .760783 6.31819E-06
END AT 0040
*
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

READ

reads values from the DATA list (DATA statements) and assigns them to variables.

Format

$$\text{READ}\begin{Bmatrix}\text{var}\\\text{svar}\end{Bmatrix}\left[\begin{Bmatrix},\mathit{var}\\,\mathit{svar}\end{Bmatrix}\right]\ldots$$

Arguments

var and svar are numeric and string variables separated by commas.

Remarks

1. READ statements must always be used in conjunction with DATA statements.
2. The variables listed in the READ statement may be subscripted or nonsubscripted and may be numeric or string.
3. The order in which variables appear in the READ statement is the order in which values for the variables are retrieved from the DATA list.
4. A data element pointer is moved to the next available value in the DATA list as values are retrieved for variables in READ statements. If the number of variables in the READ statement exceeds the number of values in the DATA list, BASIC prints an END OF DATA error message.
5. The type of variable (numeric or string) in the READ statement must match the type of the corresponding DATA element value or else BASIC prints a READ/DATA TYPES error message.
6. The RESTORE statement can be used to reset the data element pointer to the first item of the lowest numbered DATA statement or to the first item of a specific DATA statement.

Examples

```
*LIST)
0010 READ A,B,C
0020 READ D(1),D(2),D(3)
0030 PRINT C ↑ 2,D(2) ↑ 2
0040 READ E
0050 PRINT E
0060 READ F$
0070 PRINT F$
0080 DATA 1,2,3,4,5,6,7,"ABC"
0090 END
*RUN)
9    25
7
ABC

END AT 0090
*
```

In this example the variables are assigned values as follows:

Variable	Value
A	1
B	2
C	3
D(1)	4
D(2)	5
D(3)	6
E	7
F$	ABC

AOS	√
RDOS	√
DOS	√

S	√
C	
F	

REM

inserts explanatory remarks within a program.

Format

REM *[message]*

Arguments

message is text comment.

Remarks

1. REM statements do not affect program execution. BASIC stores them with a program and outputs them with each LISTing.
2. If control is transferred to a REM statement from a GOTO or GOSUB statement, then execution continues with the next executable statement. If no executable statement follows the REM statement, then the program will END and control will return to interactive mode.
3. For AOS systems, comments can be concatenated to any BASIC statement by using a comment (!) sign. (See example 2.)

Examples

1.
```
*LIST)
0010 REM REMARKS IN A PROGRAM.
0020 REM HELPS EXPLAIN THE PURPOSE OF
0030 REM STATEMENTS. LINES 10, 20, 30
0040 REM AND 40 AREN'T EXECUTED.
0050 PRINT "END"
*RUN)
END
*
```

2.
```
*LIST)
0010 LET P=61.9   ! P IS THE PRICE OF GAS
0020              ! IN CENTS PER GALLON.
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

RENAME

renames a file in your directory.

Format

RENAME "oldfilename", "newfilename"

Arguments

oldfilename is a disk file in your directory that can be a string literal or string variable.

newfilename is a new filename that can be a string literal or a string variable.

Remarks

BASIC searches your directory for oldfilename; if it is found, its name is changed to newfilename.

An error message will be printed at your terminal if:

- oldfilename does not exist.
- newfilename already exists.
- oldfilename is attribute-protected.

Examples

```
*RENAME "TEST.SR", "A.SR")
*
```

File TEST.SR is renamed as A.SR for future referencing.

AOS	√	S	
RDOS	√	C	√
DOS	√	F	

RENUMBER

renumbers the statements in the current program.

Format

$$\text{RENUMBER}\left[\left\{\begin{array}{l} \textit{line n1} \\ \left\{\begin{array}{l}\textit{STEP}\\ ,\end{array}\right\} \textit{n2} \\ \textit{line n1} \left\{\begin{array}{l}\textit{STEP}\\ ,\end{array}\right\} \textit{n2}\end{array}\right\}\right]$$

Arguments

line n1 is the initial line number for the current program.

n2 is the desired increment between line numbers.

Remarks

1. The RENUMBER command has several variations. They are:

•RENUMBER)	Renumber the current program starting with default line number 0010, with a default increment of 10 between line numbers
•RENUMBER n1)	Renumber the current program starting with line number n1 and by incrementing line numbers by n1.
•RENUMBER STEP n2)	Renumber the current program starting with default line number 0010 and incrementing line numbers by n2. You may use a comma instead of the word STEP.
•RENUMBER n1 STEP n2)	Renumber the current program starting with line number n1 and by incrementing line numbers by n2. Again, you may use a comma instead of the word STEP.

2. Line numbers are limited to four digits. If a RENUMBER command causes a line number to exceed 9999, the command is re-executed as:

   RENUMBER 1 STEP 1

3. The RENUMBER command modifies the line numbers specified in IF-THEN, GOTO, and GOSUB statements to agree with the new line numbers.

4. Line numbers which cannot be resolved are changed to 0000 and an error message is printed.

5. The RENUMBER command does not renumber the arguments of ERASE a~~-~~d CHAIN statements.

Examples

```
•LIST)
0010 TAB =5
0015 DIM A(3,4)
0020 LET A(1,2)=6
0025 LET A(3,4)=10
0030 MAT PRINT A
0035 MAT A=ZER(3,3)
0037 PRINT
0040 MAT PRINT A
•RENUMBER 10 STEP 5)
•LIST)
0010 TAB =5
0015 DIM A(3,4)
0020 LET A(1,2)=6
0025 LET A(3,4)=10
0030 MAT PRINT A
0035 MAT A=ZER(3,3)
0040 PRINT
0045 MAT PRINT A
•
```

AOS	√	S	√
RDOS	√	C	√
DOS	√	F	

RESTORE

resets the position of the data element pointer.

Format

RESTORE *[line no.]*

Arguments

line no. is a DATA statement line number.

Remarks

1. If you use the RESTORE statement without a line number argument, then the data element pointer is reset to the beginning of the data list.

2. If you use the RESTORE statement with a DATA statement line number argument, then the data element pointer is moved to the first value in the DATA statement line.

3. If *line no.* is not a DATA statement, the data element pointer will point to the first DATA statement following *line no.* If *line no.* doesn't exist in the program, an error occurs.

Examples

```
*05  READ A,B,C)
*10  READ D,E,F)
*15  RESTORE 50)
*20  READ G,H,I)
*25  RESTORE)
*30  READ J,K,L)
*40  DATA 2,4,6)
*50  DATA 8,10,12)
```

In the above example the variables are assigned values as follows:

Variable	Values
A	2
B	4
C	6
D	8
E	10
F	12
G	8
H	10
I	12
J	2
K	4
L	6

AOS	√	S	√
RDOS	√	C	
DOS	√	F	

RETRY

repeats the statement which caused an error.

Format

RETRY

Remarks

1. You can use the RETRY statement in conjunction with the ON ERR statement to return control to the statement which caused the error, and attempt to re-execute that statement.

2. For the RETRY statement to work properly, an error condition must have occurred. If no error condition has occurred, a line-number error results.

Examples

```
*005  ON ERR THEN 100)
*010  OPEN FILE (0,2), "TEST")
  .
  .
  .
```

(If statement 10 causes an error then RETRY directs the program to repeat the statement.)

```
*100  RETRY)
  .
  .
  .
```

NOTE: If statement 10 caused an error, then the program would loop indefinitely between statements 10 and 100. The program should, therefore, include some provision for exiting from the RETRY statement after a certain number of failures.

AOS	√	S	
RDOS	√	C	√
DOS	√	F	

RUN

executes a program either from the first line or from a specified line.

Format

RUN $\left[\begin{Bmatrix} \textit{line no.} \\ \textit{``filename''} \end{Bmatrix}\right]$

Arguments

line no. is the line in the current program from which execution is to begin.

filename is the name of a disk file or device.

Remarks

You may use the variations of the RUN command as follows:

*RUN)	Clear all variables, undimension all arrays and strings, do a RESTORE, initialize the random number generator, and then run the current program from the first line number.
*RUN n)	RUN from line n. This form of the RUN command allows resumption of program execution retaining current values of all variables and parameters. It may be used after a STOP or after an error and will incorporate any alterations you make to the program after the STOP or error occurred.
*RUN "filename")	If the file is on disk, BASIC first searches your directory and then the library directory for filename. When filename is found, the command executes a NEW, clearing the current program area, then LOADs and executes program filename.

Examples

```
*RUN)
*RUN "$PTR")
*RUN 250)
*RUN "MATH3")
*RUN "MT1:0")
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

SAVE

writes the current program and data in binary format to the device or disk file named by filename.

Format

SAVE "filename"

Arguments

filename is the name of a disk file or a device, expressed as a string literal or string variable.

Remarks

1. If filename is a disk file, then filename is entered into your directory. If filename is a disk file that already exists in your directory, and you type SAVE as an immediate command, BASIC will print one of the following messages depending on your operating system:

   TYPE CR TO DELETE (RDOS/DOS)
   TYPE NL TO DELETE (AOS)

   This message lets you confirm whether or not you would like to delete the existing filename, and replace the file with the lines you specified in the SAVE command. If you type a carriage return (new-line), you'll get the replacement. Type anything preceding the carriage return (new-line), and you cancel the SAVE command.

2. You can LOAD, CHAIN, or RUN a SAVEd program.

3. When you SAVE a program, the current values of all variables are also stored with the program as well as the point where the program stopped last. Therefore, you can LOAD the SAVEd program and CONtinue or RUN line no. as though no interruption had occurred. Note: File status is not preserved when you SAVE a program.

4. A SAVEd program may not run under all configurations of BASIC. In particular, if the precision of the floating-point representation in the RUN environment differs from that of the SAVE environment, you will not be able to load the program.

Examples

```
*SAVE "FA.BC")    }
*SAVE "$PTP")     } Commands
*SAVE S$(1,7))    }

*10 SAVE"OURSHIP")) }
*20 SAVE B$)        } Statements
```

AOS	√
RDOS	
DOS	

S	√
C	√
F	

SIZE

prints the number of bytes and pages used by the program, and the total number of bytes and pages that are still available.

Format

SIZE

Remarks

1. The number of bytes used are broken down into two groups: the number of bytes used for the program segment (P), and the number of bytes used for the data segment (D).

2. The number of pages used and left are reported. One page equals 2048 bytes.

3. The total number of bytes left is reported.

Examples

```
*SIZE)
USED: 14850 (P), 312 (D) BYTES 8(P), 1(D) PAGE(S)
LEFT: 29594 BYTES      13 PAGE(S)
```

AOS	
RDOS	√
DOS	√

S	√
C	√
F	

SIZE

prints the number of bytes and pages used by the program, and the total number of bytes and pages that are still available.

Format

SIZE

Remarks

1. The number of bytes used are broken down into two groups; the number of bytes used for the program segment (P), and the number of bytes used for the data segment (D).

2. If the RDOS/DOS Extended BASIC system uses swapping or extended memory, the number of pages used and left are reported. One page equals 512 bytes.

3. The total number of bytes left is reported.

Examples

Example of a swapping system:

```
*SIZE)
USED: 14850 (P), 312 (D) BYTES 8 (P), 1(D) PAGE(S)
LEFT: 29594 BYTES      13 PAGE(S)
```

Example of a nonswapping system:

```
*SIZE)
USED: 3793 (P), 2821 (D) BYTES
LEFT: 8077 BYTES
*
```

From a total of 14,691 bytes of memory available for program and data storage, 3793 are occupied by the program, 2821 by the data and 8077 remain unused.

AOS	√	S	√
RDOS	√	C	
DOS	√	F	

STOP

terminates execution of the current program and returns control to interactive mode.

Format

STOP

Remarks

1. You can place STOP statements anywhere in the program to terminate execution. When STOP is encountered, BASIC prints the following message on your terminal:

   ```
   STOP AT XXXX
   *
   ```

   where *XXXX* is the line number of the STOP statement.

2. After resumption of interactive mode, you can modify the program if you wish. To restart the program from the beginning, use RUN; to continue from the STOP statement, use CON or RUN line no.

Examples

```
*LIST)
0010 REM--TERMINATE PROGRAM BY STOP
0020 INPUT A
0030 IF A < 0 THEN GOTO 0050
0040 GOTO 0020
0050 STOP
*RUN)
? 1)
? 3)
? -5)

STOP AT 0050
*
```

AOS	√	S	√
RDOS	√	C	√
DOS	√	F	

TAB

sets the zone spacing between the data output by PRINT statements.

Format

TAB=expr

Arguments

expr is an arithmetic expression in the range: 1 <= expr <= page width (see PAGE command).

Remarks

1. The default zone spacing is 14 columns.

2. Since the maximum range of zone spacing depends upon the PAGE command setting, it is good practice to set the page width first and then the zone spacing.

Examples

```
*LIST)
0010 PAGE =50
0020 TAB =10
0030 FOR I=1 TO 25
0040   PRINT I,
0050 NEXT I
*RUN)
1        2        3        4        5
6        7        8        9        10
11       12       13       14       15
16       17       18       19       20
21       22       23       24       25
END AT 0050
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

TIME

establishes the time limit for timed input (TINPUT) operation.

Format

TIME=expr

Arguments

expr is a numeric expression which represents time in seconds to the nearest tenth of a second. In AOS, the maximum number for expr is approximately 4,000,000. In RDOS/DOS, the maximum number for expr is approximately 65,000.

Remarks

1. Assigning a value to TIME sets the SYS(14) function to the value of expr.
2. The value of SYS(14) is decremented at the clock tick rate (1/10 of a second per tick) from the time a TINPUT statement is executed.
3. Decrementing of SYS(14) stops when you respond to the TINPUT prompt. Decrementing of SYS(14) is resumed when the next TINPUT is executed.
4. If you do not respond to the TINPUT prompt before the SYS(14) function has decremented to zero, then an error message is printed and the program stops (unless an ON ERR THEN statement was executed in your program).
5. TIME may be reset to another value and may appear as often as required by the program logic.

Examples

```
*LIST)
0010 DIM A$(50)
0020 PRINT "LET'S TEST YOUR RECALL SPEED"
0030 PRINT
0040 TIME =10
0050 TINPUT "WHAT COLOR ARE YOUR MOTHER'S EYES? ",A$
0060 GOSUB 0140
0070 TINPUT "WHAT'S YOUR SOCIAL SECURITY NUMBER? ",A$
0080 GOSUB 0140
0090 TINPUT "HOW OLD IS YOUR FATHER? ",A$
0100 GOSUB 0140
0130 GOTO 0190
0140 LET I=I+1
0150 LET A(I)=(10-SYS(14))
0160 PRINT "TIME USED- ";A(I);" SECONDS"
0170 TIME =10
0180 RETURN
0190 FOR J=1 TO I
0200      LET B=B+A(J)
0210 NEXT J
0220 LET C=B/I
0230 PRINT "AVERAGE RESPONSE TIME= ";C;" SECONDS"
0240 END
*RUN)
LET'S TEST YOUR RECALL SPEED

WHAT COLOR ARE YOUR MOTHER'S EYES? BROWN)
TIME USED- 6.8 SECONDS
WHAT'S YOUR SOCIAL SECURITY NUMBER? 11234567)
TIME USED- 9.2 SECONDS
HOW OLD IS YOUR FATHER? 65)
TIME USED- 5.5 SECONDS
AVERAGE RESPONSE TIME= 7.1666667 SECONDS

END AT 0240
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

TINPUT

assigns, within a prescribed time, the values supplied by input from the terminal to a list of variables.

Format

TINPUT *[(line no.[,time]),]["str lit,"]* $\left\{\begin{matrix}\text{var}\\ \text{svar}\end{matrix}\right\}\left[\left\{\begin{matrix}\textit{,var}\\ \textit{,svar}\end{matrix}\right\}\right]\ldots$ *[;]*

Arguments

line no. is a program line number.

time is a numeric expression which evaluates to an integer representing time, in seconds, to the nearest tenth of a second.

var and svar are variables separated by commas or carriage returns (new-lines).

str lit is a message or prompt.

Remarks

1. INPUT statement remarks apply to TINPUT.
2. Use the TINPUT statement in conjunction with the TIME= statement, or the optional *(line no. [,time],)* argument to TINPUT, and the SYS(14) function.
3. The TIME= statement sets SYS(14) to the value, in seconds, allowed for your response to the TINPUT prompt.
4. If you do not respond to the TINPUT prompt before the SYS(14) function decrements to zero and the TINPUT statement has the optional *line no.*, the program will branch to that statement number. If you used no line number argument, and SYS(14) has decremented to zero, then BASIC prints an error message at your terminal and the program stops (unless an ON ERR THEN statement in your program was previously executed).
5. If the TINPUT statement includes a *time* argument, the *time* argument only applies to that specific TINPUT statement, and does not affect SYS(14). This form of TINPUT will only set a time limit for your response, but does not determine the amount of time taken by your response.

Examples

```
0005 ON ERR GOTO 300
0010 TINPUT (100, 25), "ENTER:", A$
```

The program will branch to line 100 if there is no response to the ENTER prompt within 25 seconds. The line number argument overrides the ON ERR statement.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

WHATS

determines the status of file filename.

Format

WHATS "filename"

Arguments

filename is the name of a file in your directory or in the library directory expressed as a string literal.

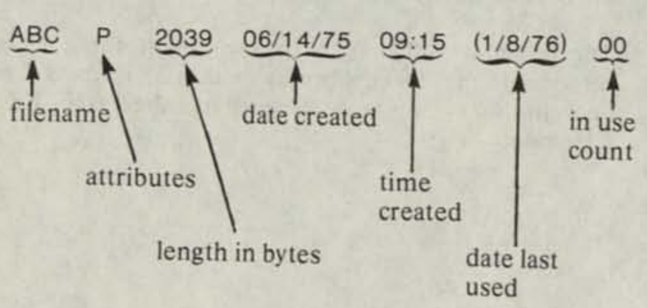
Remarks

1. BASIC searches your directory for filename; if not found, the library directory is searched.
2. The output from an AOS WHATS command is different from RDOS/DOS, and is shown in Example 2.

Examples

1.

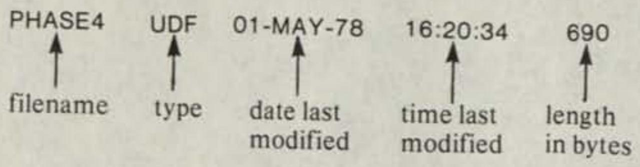
```
*WHATS "ABC")
```



```
ABC  P  2039  06/14/75  09:15  (1/8/76)  00
```

2.

```
*WHATS "PHASE4")
```



```
PHASE4  UDF  01-MAY-78  16:20:34  690
```

AOS	√
RDOS	
DOS	

S	√
C	√
F	

WHO

identifies other people on the system or determines your own identification.

Format

WHO $\left[\begin{Bmatrix} \textit{processID} \\ \textit{``processname''} \end{Bmatrix} \right]$

Arguments

processID is a process identification number.

processname is a process name, assigned by AOS, expressed as a string variable or string literal.

Remarks

1. You can identify other people using the system by using the WHO command with either process identification numbers or process names. BASIC will respond to the command by printing both the process identification number and the process name.
2. The WHO command, without any arguments, will print your process identification number and process name on your terminal.

Examples

```
*WHO 7)
PID: 7 XBASIC:007
*
```

End of Chapter

Chapter 4
Extended BASIC Functions

Introduction to Extended BASIC Functions

The Extended BASIC functions perform calculations which you would otherwise need to execute yourself--either with a series of program statements or by consulting mathematical tables. The functions generally have a three-character mnemonic name and are followed by a parenthesized expression (expr) which is the function argument. Generally, a function may be used as an expression, or may be included as part of an expression.

The functions included in Extended BASIC are listed below and are described in this chapter.

Function	Value Produced
ABS(x)	Absolute value of x.
ATN(x)	Arctangent of x (result expressed in radians).
COS(x)	Cosine of x (x expressed in radians).
CPU(x)	Console switch value.
DEF FNa(d)	User-defined function.
EOF(x)	Returns a +1 if an end of file is detected; otherwise zero.
EXP(x)	e^x (-178 <= x <= 175).
INT(x)	The largest integer not greater than x.
LEN(x$)	The number of characters in string x$.
LOG(x)	Natural logarithm of x (x > 0).
POS(x$,y$,z)	The position of a substring in a string.
RND(x)	Random number between 0 and 1.
SGN(x)	The algebraic sign of x.
SIN(x)	Sine of x (x expressed in radians).
SQR(x)	Square root of x (x >= 0).
STR$(x)	The string value of a numeric expression.
SYS(x)	System functions.
TAB(x)	Used with PRINT to position to column x.
TAN(x)	Tangent of x (x expressed in radians).
VAL(x$)	The numeric value of a string.

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

ABS(X)

returns the absolute (positive) value of expr.

Format

ABS (expr)

Arguments

expr is a numeric expression.

Examples

```
*LIST)
0010 PRINT ABS(-30)
*RUN)
30

END AT 0010
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

ATN(X)

calculates the angle (in radians) whose tangent is expr. $(-\pi/2 < \text{result} < \pi/2)$.

Format

ATN(expr)

Arguments

expr is a numeric expression.

Examples

```
*LIST)
0010 REM-CALCULATE ANGLE WHOSE TAN=2
0020 PRINT ATN(2)
*RUN)
1.1071487

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

COS(X)

calculates the cosine of an angle which is expressed in radians.

Format

COS(expr)

Arguments

expr is a numeric expression specified in radians.

Remarks

SYS(15) is assigned the value of PI (3.1416). For more information see the SYS(X) function.

Examples

```
*LIST)
0010 REM-PRINT COSINE OF 30 DEGREES
0020 LET P=SYS(15)/180
0030 PRINT COS(30*P)
*RUN)
.8660254

END AT 0030
*
```

AOS	
RDOS	√
DOS	√

S	
C	
F	√

CPU(X)

returns a value equal to the status of a CPU data switch or the numeric value of all 16 console switches.

Format

CPU(expr)

Arguments

expr is an expression which evaluates to the number of a CPU console switch or a -1.

Remarks

1. The value returned is:

   0 if console switch is down.
   1 if console switch is up.

2. If expr is a -1, the CPU function returns the decimal value of all 16 console switches. This value is in the range 0 through 65535.

3. Each switch corresponds to a bit in a 16-bit word with the bits numbered 0 through 15 from left to right.

Examples

```
*10 IF CPU(0) THEN GOTO 85)
```

Proceed to statement 85 if console switch 0 is up.

```
*PRINT CPU(-1))
□33
```

Switches 11, 12, 14, and 15 are up.

AOS	√		S	√
RDOS	√		C	
DOS	√		F	√

DEF FNa(d)

permits you to define as many as 26 different functions which can be repeatedly referred to throughout a program.

Format

DEF FNa(d)=expr

Arguments

a is a single letter from A to Z.

d is a dummy arithmetic variable that may appear in expr.

expr is an arithmetic expression which may contain a variable d.

Remarks

1. Each function returns a numeric value.
2. BASIC does not relate the dummy variable named in the DEF statement to variables in the program with the same name; the DEF statement simply defines the function and does not cause any calculation to be carried out.
3. In the function definition, the expr can be any legal arithmetic expression and may include other user-defined functions. Functions may be nested to a depth specified by your system manager.
4. Function definition is limited to a single line DEF statement. Complex functions which require more than one program statement should be constructed as subroutines.

Examples

```
*LIST)
0010 DEF FNE(J)=(J↑2)+2*J+1
0020 LET Y=FNE(5)
0030 PRINT Y
*RUN)
36

END AT 0030
*
```

In line 10 the FNE function is defined. In line 20 the FNE function is referred to and evaluated with numeric argument 5.

You can also redefine a function, as in the following example:

```
*LIST)
0010 DEF FNA(X)=X↑2
0020 PRINT FNA(2)
0030 DEF FNA(Z)=Z↑3
0040 PRINT FNA(2)
*RUN)
4
8
END AT 0040
*
```

The following example illustrates the nesting of user-defined functions.

```
*LIST)
0005 TAB=16
0010 LET P=SYS(15)
0020 DEF FNR(X)=X*P/180
0030 DEF FNS(X)=SIN(FNR(X))
0040 DEF FNC(X)=COS(FNR(X))
0050 FOR X=0 TO 45 STEP 5
0060   PRINT X,FNS(X),FNC(X)
0070 NEXT X
*RUN)
0      0                1
5      8.7155743E-02    .9961947
10     .17364818        .98480775
15     .25881905        .96592583
20     .34202014        .93969262
25     .42261826        .90630779
30     .5               .8660254
35     .57357644        .81915204
40     .64278761        .76604444
45     .70710678        .70710678

END AT 0070
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

EOF(X)

detects the end of file when transferring data from a file.

Format

EOF (file)

Arguments

file is a numeric expression which evaluates to the number of a file opened for reading in mode 0 or 3.

Remarks

1. The EOF function returns an integer indicating whether or not the last READ FILE or INPUT FILE from file included an end-of-file delimiter. If an end of file was detected, the function returns a value of 1; otherwise the function returns a 0.
2. When the EOF function is used in conjunction with the IF-THEN statement, a conditional transfer can be made if an end of file is detected.
3. Testing for an end of file should occur immediately after the INPUT FILE or READ FILE statement. If an end of file was values, which are usually the last data input or read before the end of file. This concept also applies to Matrix file I/O statements.
4. A random (mode 0) file returns an EOF if you attempt to read a record number larger than the last written in the file. Processing of the random file may be continued after an EOF by using the RESET FILE statement.
5. All physical records written to magnetic tape and cassettes are of a fixed length and padded with nulls. Therefore, the EOF function is not necessarily set at logical end of file; the function is set at logical end of file only when it coincides with physical end of file.

Examples

```
*LIST)
0100 OPEN FILE(1,3), "$PTR"
0110 READ FILE(1),A,B,C,D,E
0120 IF EOF(1) THEN GOTO 0200
0130 PRINT A,B,C,D,E
0140 GOTO 0110
0200 CLOSE FILE(1)
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

EXP(X)

calculates the value of e (2.71828) to the power of expr.

Format

EXP(expr)

Arguments

expr is a numeric expression from -178 through 175.

Examples

```
*LIST)
0010 REM-CALCULATE VALUE OF E ↑ 1.5
0020 PRINT EXP(1.5)
*RUN)
4.4816891

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

INT(X)

returns the value of the largest integer not greater than expr.

Format

INT(expr)

Arguments

expr is a numeric expression.

Remarks

The INT function truncates numbers to return integers, but print formatting rounds numbers for output. There may appear to be discrepancies, but the internal number representation does not change.

Examples

```
*LIST)
0010 PRINT "INT(15.8)= ";INT(15.8)
0020 PRINT "INT(-15.8)= ";INT(-15.8)
0030 PRINT "INT(15.8+.5)= ";INT(15.8+.5)
*RUN)
INT(15.8)= 15
INT(-15.8)= -16
INT(15.8+.5)= 16

END AT 0030
*
```

Line 30 of this example demonstrates a technique for rounding real numbers to the nearest integer.

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

LEN(X$)

returns a value equal to the number of characters currently assigned to string variable svar.

Format

LEN (svar)

Arguments

svar is a string variable.

Examples

```
*LIST)
0005 DIM A$(80),B1$(80)
0010 INPUT A$,B1$
0020 LET B=LEN(A$)
0040 IF B > LEN (B1$) THEN GOTO 0060
0050 GOTO 0100
0060 PRINT "LENGTH OF A$= ";LEN(A$)
0070 PRINT "LENGTH OF B1$= ";LEN(B1$)
0080 PRINT "A$ > B1$"
0090 GOTO 0110
0100 PRINT "B1$ > A$"
0110 END
*RUN)
? CHEESE) ? CAKE)
LENGTH OF A$= 6
LENGTH OF B1$= 4
A$ > B1$

END AT 0110
*
```

AOS	√		S	
RDOS	√		C	
DOS	√		F	√

LOG(X)

calculates the natural logarithm of expr.

Format

LOG (expr)

Arguments

expr is a numeric expression.

Examples

```
*LIST)
0010 REM-CALCULATE THE LOG OF 959
0020 PRINT LOG(959)
*RUN)
6.8658911

END AT 0020
*
```

AOS	√		S	
RDOS	√		C	
DOS	√		F	√

POS(X$,Y$,Z)

determines the POSition of a substring in a string.

Format

$$\text{POS}\left(\begin{Bmatrix}\text{svar 1}\\ \text{“str lit 1”}\end{Bmatrix},\begin{Bmatrix}\text{svar 2}\\ \text{“str lit 2”}\end{Bmatrix},\ \text{expr}\right)$$

Arguments

svar is a string variable.

str lit is a string literal.

expr is a numeric expression.

Remarks

1. Starting at position expr in a string (svar1 or str lit1) BASIC searches for the specified substring (svar2 or str lit2).
2. The POS function returns a value equal to the first position of the substring in the string. If the substring cannot be found in the string, the POS function returns a value of zero. If the value of expr is less than zero, an error message will occur.
3. If expr is greater than the length of svar, the POS function will return a value of zero.
4. If expr is equal to zero, the search will begin at the first position of svar.

Examples

```
*LIST)
0005 DIM A$(25)
0010 LET A$="AMNOPFGHIJKLMNOPQRS"
0020 LET A=POS(A$,"MNOP",6)
0030 PRINT A
*RUN)
13

END AT 0030
*
```

In this example, a search is made for "MNOP" starting from the sixth character (F) in string A$. A match is found which begins at the 13th character in string A$. Therefore, the POS function returns a value of 13 which is assigned to variable A. If expr equalled 1, the POS function would have found the first MNOP, and assigned value 2 to A.

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

RND(X)

produces a pseudo-random number n, such that 0 <= n < 1.

Format

RND (expr)

Arguments

expr is a numeric expression (required, but not used).

Remarks

1. The RND function requires a numeric argument (expr), although the argument does not affect operation of the function.

2. Each time the RND function is called, it provides a pseudo-random number n, such that 0 <= n < 1. The sequence of these numbers is fixed. The sequence repeats starting at the 58384th value (the 58383rd value is zero). RANDOMIZE may start at some point in this sequence, or may provide a sequence with a shorter repeat cycle. The sequence is the same for all systems, but the values in a double-precision system are output to more significant digits.

3. Each occurrence of the RND function in a program yields the value of the next random number in the list.

4. Each time you issue a NEW, CHAIN, or RUN, BASIC returns to its original starting place in the sequence of random numbers. Because the sequence is fixed, and the starting place is the same for each RUN, the RND function will provide the same numbers each time you execute your program. The capability of reproducing the sequence can be a useful debugging aid.

5. To alter the starting place in the sequence, use the RANDOMIZE statement described in Chapter 3. RANDOMIZE resets the starting place based on the time of day, thus providing a different sequence for each run.

Examples

```
*LIST)
0005 TAB =13
0010 FOR I=1 TO 4
0020    PRINT RND(I)
0030 NEXT I
*RUN)
.21176298
.26666685
.5411776
.90979748

END AT 0030
*
```

Running the above program a second time will produce the same five random numbers.

```
*LIST)
0005 TAB =13
0010 FOR J=1 TO 4
0020    PRINT INT(10*RND(I))
0030 NEXT J
*RUN)
2
2
5
9

END AT 0030
*
```

This program will produce four random integers in the range 0 to 9.

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

SGN(X)

returns a value which represents the sign of an expression.

Format

SGN(expr)

Arguments

expr is a numeric expression.

Remarks

The value returned is:

1	if positive
0	if 0
-1	if negative

Examples

```
*LIST)
0010 LET A=-3
0020 PRINT SGN(A)
*RUN)
-1

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

SIN(X)

calculates the sine of an angle which is expressed in radians.

Format

SIN(expr)

Arguments

expr is a numeric expression specified in radians.

Remarks

SYS(15) is assigned the value of PI (3.1416). See the SYS(x) function for more information.

Examples

```
*LIST)
0010 REM - PRINT SINE OF 30 DEGREES
0020 PRINT SIN(30*SYS(15)/180)
*RUN)
.5

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

SQR(X)

computes the square root of expr.

Format

SQR(expr)

Arguments

expr is a nonnegative numeric expression.

Examples

```
*LIST)
0010 LET A=5
0020 PRINT SQR(A↑2+75)
*RUN)
10

END AT 0020
*
```

AOS	√
RDOS	√
DOS	√

S	
C	
F	√

STR$(X)

converts the numeric value of an expression to a string.

Format

STR$(expr)

Arguments

expr is a numeric expression.

Remarks

1. Converting numerics to string manipulation with no leading or trailing spaces permits string manipulation by other string handling functions and statements.

2. This function is useful for combining numbers when you don't want spaces between them.

Examples

```
*LIST)
0010 READ A
0015 IF A=0 THEN STOP
0020 LET A$=STR$(A)
0030 IF A$(4,6)="222" THEN GOTO 0050
0040 GOTO 0070
0050 PRINT A;"-THIS IS MODEL 222"
0060 GOTO 0010
0070 PRINT A;"-THIS ISN'T OUR MODEL"
0080 DATA 111222,212222,123456,0
0090 GOTO 0010
*RUN)
111222-THIS IS MODEL 222
212222-THIS IS MODEL 222
123456-THIS ISN'T OUR MODEL

STOP AT 0015
*
```

AOS	√	S	
RDOS	√	C	
DOS	√	F	√

SYS(X)

returns system information based on the value of expr which is evaluated to an integer (0 to 18).

Format

SYS (expr)

Arguments

expr is a numeric value or expression, between 0 and 18.

Remarks

The values returned by the SYS function are listed below:

SYS(0)	the time of day (seconds past midnight)	
SYS(1)	the day of the month (1 to 31)	
SYS(2)	the month of the year (1 to 12)	current
SYS(3)	the year in four digits (e.g., 1977)	date
SYS(4)	the terminal port number (-1 if operator's console)	
SYS(5)	CPU time used in seconds to the nearest tenth	
SYS(6)	I/O usage (numbers of file I/O statements executed)	
SYS(7)	the error code of the last runtime error	
SYS(8)	the file number of the file most recently referred to in a file I/O statement	
SYS(9)	page size	
SYS(10)	tab size	
SYS(11)	hours	
SYS(12)	minutes current time of day	
SYS(13)	seconds	
SYS(14)	seconds remaining before expiration of timed input	
SYS(15)	PI (3.14159)	
SYS(16)	e (2.71828)	
SYS(17)	1/10 second clock (Not applicable to AOS)	
SYS(18)	total number of BASIC I/O calls (ENTER, LIST, etc.)	

Examples

*PRINT SYS (0))
63736
*PRINT SYS(1); SYS(2)(; SYS(3))
31 10 1976

AOS	√	S	
RDOS	√	C	
DOS	√	F	√

TAB(X)

tabulates to column number expr.

Format

TAB (expr)

Arguments

expr is an expression which is evaluated to an integer.

Remarks

1. The TAB function can only be used in conjunction with PRINT statements. It cannot be used with any other BASIC statement. More than one TAB(X) function may appear in a PRINT statement. The next item in the print list is printed at position X.

2. The first column on a line is column 1. The column number specified by expr is always relative to column 1. The position at which BASIC prints an item in the print list depends on the value of expr and on the PRINT statement punctuation (; or ,) following the TAB(X) function.

3. If expr evaluates to a column number lower than the present column number, then printing proceeds at that same position on the next line.

4. If expr evaluates to a column number greater than the page length, the expression is reduced modulo the page length and positioning proceeds as in 2.

Examples

```
*LIST
0005 LET A=-6
0010 LET B=5
0015 PRINT  TAB(B);A; TAB(2*B);2*A
0020 END
*RUN
    -6   -12

END AT 0020
*


↑   ↑     ↑
1   5     10
```

Notice the use of the semicolon (;) in line 15 after *A* to prevent spacing to the next print zone and passing position *2*B* (Column 10).

AOS	√		S	
RDOS	√		C	
DOS	√		F	√

TAN(X)

calculates the tangent of an angle which is expressed in radians.

Format

TAN (expr)

Arguments

expr is a numeric expression specified in radians.

Examples

```
*LIST)
0010 REM - PRINT TANGENT OF X DEGREES
0020 INPUT "X DEGREES ",X
0030 LET P=SYS(15)/180
0040 PRINT TAN(X*P)
*RUN)
X DEGREES 45
1

END AT 0040
*
```

AOS	√		S	
RDOS	√		C	
DOS	√		F	√

VAL(X$)

returns the numeric representation of a string value.

Format

VAL $\left(\begin{Bmatrix}\text{svar}\\ \text{“str lit”}\end{Bmatrix}\right)$

Arguments

str lit or svar is a string beginning with a number.

Remarks

1. The string variable or string literal argument to the VAL function must begin with a number or else an error message will be output. The number may include digits, plus and minus signs, decimal points, and the letter E (scientific notation). Any nonnumeric characters which appear after the number portion of the string are ignored. For example:

   "+35.5E-03ABCD7N"

   Substring "+35.5E-03" is returned as a numeric value and substring "ABCD7N" is ignored.

2. Misplaced signs terminate the input scan in a similar fashion:

   "123 + 47 - 17"

   Substring "123" is returned as a numeric value and "+47-17" is ignored.

Examples

```
*LIST)
0010 LET A$="12345ABCD"
0020 LET B=54321
0030 LET C=VAL(A$)
0040 LET D=B+C
0050 PRINT D
*RUN)
66666

END AT 0050
*
```

End of Chapter

Chapter 5
Array Manipulation

Dimensioning Arrays

One-dimensional arrays are called vectors; two-dimensional arrays are called matrixes. Matrix statements also work for vectors wherever the argument row is optional.

Arrays can be dimensioned by any of the following three methods:

1. Using a DIM statement to declare the number of elements for a vector or rows and columns for a matrix.
2. Including the dimensions in a matrix statement.
3. Allowing a default size of 10 elements, or 10 rows and 10 columns, by not specifying dimensions in a DIM or matrix statement.

Please note that a matrix does not have row 0 or column 0, and as in all BASIC arrays, matrix elements are stored by row in ascending locations in memory.

A matrix that is dimensioned in a DIM statement is automatically initialized to all zeros.

Matrix statements allow dimensioning and redimensioning as long as the total number of elements in the new dimensions does not exceed the total number of elements of the matrix declared in the original DIM or other matrix statement. For example:

```
*20  DIM A(15,14))          (210 elements in matrix
                             A)
*40  MAT A=CON(20,7))       (140 elements)
*60  MAT A=ZER(10,10))      (100 elements)
```

Statements 40 and 60, above, redimension matrix A as well as perform matrix operations described later in this chapter.

We cover the following categories of matrix statements in this chapter:

- Matrix Manipulation Statements
- Matrix I/O Statements
- Matrix Calculation Statements

In addition, you will find more matrix file input/output statements described in Chapter 6.

Matrix Manipulation Statements

AOS	✓		S	✓
RDOS	✓		C	✓
DOS	✓		F	

Matrix Assignment

copies the elements of matrix mvar2 into matrix mvar1.

Format

MAT mvar1 = mvar2

Arguments

mvar	is a matrix variable name.

Remarks

This is the matrix assignment statement. Matrix mvar1 will assume the identical dimensions and values of matrix mvar2, if the storage space allocated to mvar1 is sufficient to accommodate the current dimension of mvar2. If mvar1 has not been dimensioned, it will be dimensioned by this statement.

Examples

```
*LIST)
0010 DIM A(2,2)
0020 LET A(1,1)=5
0030 LET A(1,2)=10
0035 MAT PRINT A
0040 MAT B=A
0045 PRINT
0050 MAT PRINT B
*RUN)
5   10               (Matrix A)
0   0

5   10               (Matrix B)
0   0

END AT 0050
*
```

Line 40 assigns matrix B the same dimensions as matrix A and assigns each element value in matrix A to the corresponding element in matrix B. Therefore, B(1,1) = 5 and B(1,2) = 10.

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	✓
F	✓

Zero Matrix (ZER)

sets the value of each element in a matrix to zero.

Format

MAT mvar = ZER *[([row,]col)]*

Arguments

mvar is a matrix variable name.

row is the number of rows in matrix.

col is the number of columns in matrix, or number of elements in a vector.

Remarks

1. Use form MAT mvar = ZER for previously dimensioned matrixes.
2. Use the form MAT mvar = ZER *[([row,]col)]* if the matrix was not previously dimensioned or if the matrix is to be redimensioned.
3. All matrix elements are set to zero regardless of any previously assigned values.

Examples

```
*LIST)
0010 DIM A(3,4)
0020 LET A(1,2)=6
0030 LET A(3,4)=10
0040 MAT PRINT A
0045 PRINT
0050 MAT A=ZER(3,3)
0060 MAT PRINT A
*RUN)
0  6  0  0
0  0  0  0
0  0  0  10

0  0  0
0  0  0   (Matrix A after line 50.)
0  0  0

END AT 0060
*
```

In line 50, matrix A is redimensioned and all elements are assigned a value of zero.

AOS	✓
RDOS	✓
DOS	✓

S	✓
C	✓
F	

Unit Matrix (CON)

sets the value of each element in a matrix to one.

Format

MAT mvar = CON *[([row,]col)]*

Arguments

mvar is a matrix variable name.

row is the number of rows in matrix.

col is the number of columns in matrix, or elements in a vector.

Remarks

1. Use the form MAT mvar = CON for previously dimensioned matrixes.
2. Use the form MAT mvar = CON *[([row,]col)]* if the matrix was not previously dimensioned or if the matrix is to be redimensioned.
3. All matrix elements are set to one regardless of any previously assigned values.

Examples

```
*LIST)
0010 DIM A(2,5)
0020 READ A(1,1),A(1,2),A(1,5)
0030 DATA 8,9,10,11,12
0040 MAT PRINT A
0045 PRINT
0050 MAT A=CON(2,4)
0060 MAT PRINT A
*RUN)
8  9  0  0  10
0  0  0  0  0

1  1  1  1   (Matrix A after line 50.)
1  1  1  1

END AT 0060
*
```

In line 50, matrix A is redimensioned and all elements of the matrix are assigned a value of one.

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

Identity Matrix (IDN)

sets each element of the major diagonal of the matrix to one and each remaining element to zero.

Format

MAT mvar = IDN *[([row,]col)]*

Arguments

mvar   is a matrix variable name.

row   is the number of rows in matrix.

col   is the number of columns in matrix, or elements in a vector.

Remarks

1. The major diagonal is defined as the diagonal that starts at the last element of the array and runs diagonally upward until the first row or first column is encountered.

2. Use the form MAT mvar = IDN for previously dimensioned matrixes.

3. Use the form MAT mvar = IDN *[([row,]col)]* if the matrix was not previously dimensioned or if the matrix is to be redimensioned.

Examples

```
*LIST)
0050 DIM A(4,4)
0100 MAT A=IDN
0150 MAT PRINT A
*RUN)
1  0  0  0
0  1  0  0
0  0  1  0
0  0  0  1

END AT 0150
*

*LIST)
0010 DIM B(4,3)
0015 MAT PRINT B
0020 PRINT
0025 MAT B=IDN(2,3)
0030 MAT PRINT B
*RUN)
0  0  0
0  0  0
0  0  0
0  0  0

0  1  0   (Matrix B after line 25.)
0  0  1

END AT 0030
*
```

Matrix I/O Statements

MAT READ

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

reads values from the data list and assigns them to the elements of the matrix or matrixes listed in the MAT READ statement.

Format

MAT READ mvar *[([row,]col)] [,mvar[([row,]col)]]...*

Arguments

mvar is a matrix variable name.

row is the number of rows in matrix.

col is the number of columns in matrix, or elements in a vector.

Remarks

If a matrix was not previously dimensioned, you can dimension it in a MAT READ statement.

Examples

```
*LIST)
0010 MATREADM(5,6)
0020 DATA 0,2,4,6,8,10,-9,-8,-7,-6,-5
0030 DATA -4,-3,-2,-1,0,1,3,5,7,9,11
0040 DATA .1,0,.5,7,-8,15,-15,35,41,13,18
0050 MATPRINTM
*RUN)
0     2     4     6     8     10
-9    -8    -7    -6    -5    -4
-3    -2    -1    0     1     3
5     7     9     11    .1    0
.5    7     -8    15    -15   35

ENDAT0050
*
```

Values from the data list are read into the 30-element matrix dimensioned as 5 by 6 in the MAT READ statement.

MAT INPUT

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

reads values from your terminal and assigns them to the elements of a matrix or list of matrixes when your program is run.

Format

MAT INPUT *["str lit",]* mvar *[([row,]col)]*
[,mvar[([row,]col)]]...

Arguments

mvar is a matrix variable name.

row is the number of rows in matrix.

col is the number of columns in matrix, or elements in a vector.

str lit is a message or prompt.

Remarks

1. You can dimension or redimension a matrix with a MAT INPUT statement.

2. You enter data values, separated by either a comma or a carriage return, for each element of the matrix. Terminate the list with a carriage return.

3. If you do not supply enough data to fill the matrix before typing the carriage return, the program will continue to request data until each element of the matrix has been filled.

Examples

```
*LIST)
0010 MATINPUTX(2,3)
0015 PRINT
0020 MATPRINTX
*RUN)
? 2,4,6)
? 77,7,9)

2  4  6
77 7  9

ENDAT0020
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT TINPUT

reads values from your terminal and assigns them to the elements of a matrix or list of matrixes, within a prescribed time.

Format

MAT TINPUT *[(line [,time]),][“str lit”,]* mvar *[([row,]col)] [,mvar[([row,]col)]]...*

Arguments

line is a valid statement line number.

time is a numeric expression which evaluates to an integer and represents time, in seconds.

str lit is a message or prompt.

mvar is a matrix variable name.

row is the number of rows in matrix.

col is the number of columns in matrix, or elements in a vector.

Remarks

The remarks for the MAT INPUT statement and the TINPUT statement (Chapter 3) apply here to the MAT TINPUT statement.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT PRINT

prints the values of the elements of a matrix or list of matrixes to your terminal.

Format

MAT PRINT mvar $\left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\} \textit{mvar}\right] \cdots \left[\left\{\begin{matrix}, \\ ;\end{matrix}\right\}\right]$

Arguments

mvar is a matrix variable name.

Remarks

1. A matrix must be dimensioned by a DIM statement or other matrix statement before you can use it in a MAT PRINT statement.

2. A semicolon after a variable name in the MAT PRINT statement indicates the matrix will be printed in compact format. A comma or carriage return after the variable name indicates the matrix will be printed in zone format.

3. Column vectors (arrays) are printed one value per line.

Examples

```
*LIST)
0010 DIM A(10,10)
0020 READ N
0030 MAT A=CON(N,N)
0050 FOR I=1 TO N
0060    FOR J=1 TO N
0070       LET A(I,J)=1/(I+J-1)
0080    NEXT J
0090 NEXT I
0130 MAT PRINT A
0190 DATA 3
*RUN)
1              .5             .33333333
.5             .33333333      .25
.33333333      .25            .2
```

Matrix Calculation Statements

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

Addition and Subtraction

perform the scalar addition or subtraction of two matrixes.

Format

MAT mvar1 = mvar2 $\left\{\begin{matrix}+\\-\end{matrix}\right\}$ mvar3

Arguments

mvar is a matrix variable name.

Remarks

1. Matrixes mvar2 and mvar3 must have the same dimensions.
2. Matrix mvar1 may appear on both sides of the equal sign.
3. Arithmetic is performed on an element-by-element basis of mvar2 and mvar3 with the result assigned to the element of mvar1.

Examples

```
*LIST)
0010 DIM A(3,2),B(3,2),C(3,2)
0040 MAT READ B,C
0050 MAT A=B+C
0060 DATA -2,-5,3,4,.5,.1,6,4,-2,15,1.5,
0070 MAT PRINT B
0075 PRINT
0080 MAT PRINT C
0085 PRINT
0090 MAT PRINT A
*RUN)
-2      -5
3       4
.5      .1

6       4
-2      15
1.5     4

4       -1
1       19
2       4.1

END AT 0090
*
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

Multiplication

multiplies a matrix by a numeric expression or another matrix.

Format

MAT mvar1 = $\left\{\begin{matrix}\text{mvar2}\\\text{(expr)}\end{matrix}\right\}$ * mvar3

Arguments

mvar is a matrix variable name.

expr is any numeric expression enclosed in parentheses.

Remarks

1. Matrix mvar1 may not be the same as either matrix mvar2 or mvar3 if you are multiplying the two matrixes. Otherwise, mvar1 and mvar3 can be the same if you are multiplying by a scalar, as in the statement:

   mvar1 = (expr) * mvar3

2. If two matrixes (mvar2 and mvar3) are multiplied, the number of columns in mvar2 must equal the number of rows in mvar3. The resultant matrix (mvar1) will have the same number of columns as mvar3, and the same number of rows as mvar2.
3. If a matrix is multiplied by a numeric expression, a scalar multiplication is performed on each element of the matrix.
4. To obtain the product of two matrixes (mvar2 * mvar3), the elements of each row in mvar2 are multiplied by the elements of each column in mvar3. Each row/column set is added together to provide the resultant value of the matrix element in mvar1.

Examples

1.
```
*LIST)
0001 REM-MATRIX MULTIPLICATION
0010 DIM A(2,2),B(2,2)
0020 MAT READ B
0030 MAT A=(5)*B
0040 DATA -.5,.8,1.5,-1
0050 MAT PRINT B
0055 PRINT
0060 MAT PRINT A
*RUN)
-.5     .8
1.5     -1

-2.5    4
7.5     -5

END AT 0060
*
```

2.
```
*LIST)
0001 REM-PRODUCT OF TWO MATRIXES
0010 DIM A(3,2),B(3,2),C(2,2)
0020 MAT READ B,C
0030 MAT PRINT B
0035 PRINT
0040 MAT PRINT C
0050 MAT A=B*C
0055 PRINT
0060 MAT PRINT A
0070 DATA 2,3,1,5,0,4,-1,-2,7,8
*RUN)
2       3
1       5
0       4

-1      -2
7       8

19      20
34      38
28      32

END AT 0070
*
```

Matrix A is calculated as shown in Figure 5-1.

$$\begin{array}{ll} [B(1,1)*C(1,1)+B(1,2)*C(2,1)] & [B(1,1)*C(1,2)+B(1,2)*C(2,2)] \\ [B(2,1)*C(1,1)+B(2,2)*C(2,1)] & [B(2,1)*C(1,2)+B(2,2)*C(2,2)] \\ [B(3,1)*C(1,1)+B(3,2)*C(2,1)] & [B(3,1)*C(1,2)+B(3,2)*C(2,2)] \end{array}$$

$$= \begin{array}{ll} [2*(-1)+3*7] & [2*(-2)+3*8] \\ [1*(-1)+5*7] & [1*(-2)+5*8] \\ [0*(-1)+4*7] & [0*(-2)+4*8] \end{array} = \begin{vmatrix} 19 & 20 \\ 34 & 38 \\ 28 & 32 \end{vmatrix}$$

Figure 5-1. Product of Two Matrixes

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

Inverse Matrix (INV)

provides a matrix inversion of mvar2 and assigns the resultant matrix element value to mvar1.

Format

MAT mvar1 = INV(mvar2)

Arguments

mvar is a matrix variable name.

Remarks

1. An inverse matrix is defined such that the product of a matrix and the inverse of the matrix is the identity matrix (see IDN).

2. Matrix mvar2 must be a square matrix.

3. Matrixes may be inverted into themselves (i.e., mvar1 = mvar2 in the matrix INV statement).

4. If ERROR 08 - SINGULAR MATRIX results, mvar2 is singular or nearly singular and DET(X) will be zero. If DET(X) is very small in absolute value relative to the elements of mvar2, the matrix mvar2 should be considered singular or nearly singular and the result, mvar1, should be used with discretion.

5. The arithmetic of matrix inversion requires a knowledge of matrix determinants and of matrix cofactors. For further information on these subjects, consult a mathematical text.

Examples

```
*LIST)
0010 DIM A(2,2)
0015 MAT READ A
0020 DATA 1,2,3,4
0030 MAT A=INV(A)
0040 MAT PRINT A
*RUN)
-2         1
1.5        -.5

END AT 0040
*
```

This example may be analyzed as follows:

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = \text{matrix A}$$

Then:

$$\begin{vmatrix} 4 & -2 \\ -3 & 1 \end{vmatrix} = \text{cofactor of matrix A}$$

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = (1*4) - (2*3) = -2 = \text{determinant of matrix A}$$

$$\text{INV(A)} = (1/-2) \begin{matrix} 4 & -2 \\ -3 & 1 \end{matrix} = \begin{matrix} -2 & 1 \\ 1.5 & -.5 \end{matrix}$$

When the original matrix is multiplied by its inverse,

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} * \begin{vmatrix} -2 & 1 \\ 1.5 & -.5 \end{vmatrix}$$

the result is the identity matrix:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

Matrix Determinant (DET)

obtains the determinant of the last matrix inverted by an INV statement.

Format

var = DET (X)

Arguments

var is a numeric variable.

X is a dummy argument which is necessary but not used.

Remarks

The value of the determinant calculated for the matrix is assigned to numeric variable var.

Examples

```
*LIST)
0020 DIM A(2,2)
0030 MAT READ A
0040 DATA 1,2,3,4
0050 MAT PRINT A
0080 MAT A=INV(A)
0085 PRINT
0090 MAT PRINT A
0100 LET B=DET(X)
0120 PRINT
0130 PRINT "DETERMINANT= ";B
*RUN)
1          2
3          4

-2         1
1.5        -.5

DETERMINANT=-2

END AT 0130
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

Matrix Transposition (TRN)

transposes matrix mvar2 and assigns the resultant element values to mvar1.

Format

MAT mvar1 = TRN (mvar2)

Arguments

mvar is a matrix variable name.

Remarks

1. A matrix is transposed by reversing the row and column assignments of the matrix elements.
2. Variable names mvar1 and mvar2 cannot be the same in a TRN statement.
3. The resultant matrix, mvar1, is redimensioned to the reversed row and column dimensions of mvar2.

Examples

```
*LIST)
0020 DIM B(3,4)
0030 MAT READ B
0040 DATA 4,5,7,9,0,0,0,0,1,3,5,7
0050 MAT PRINT B
0060 PRINT
0080 MAT A=TRN(B)
0090 MAT PRINT A
*RUN)
4  5  7  9
0  0  0  0
1  3  5  7

4  0  1
5  0  3
7  0  5
9  0  7

END AT 0090
*
```

End of Chapter

Chapter 6
File Input and Output

File Concepts

Definition of a File

A file is a collection of related data treated as a unit. Each file has one or more names (called filenames) which allow both you and the system to address it.

Devices are the physical means for storing and retrieving information. There are two distinct types of devices:

- unit record
- multifile

Unit-record devices such as card readers, terminals, and line printers usually transmit and/or receive only single records; they're used for I/O, data storage, and retrieval.

Multifile devices include magnetic tape units and disks. As the name implies, they allow you to read and write more than one file per device. In particular, the flexibility of disks allows you to structure the organization of the file so that you can randomly access individual records for reading or writing.

A single record is the result of a single READ FILE or WRITE FILE statement. When you specify a record size (or use the mode 0 default record size) in an OPEN FILE statement, the output of a WRITE FILE statement with fewer than the specified number of bytes will be padded with nulls to fit the size of the record.

Output having greater than the specified number of bytes will cause an error. If a READ FILE statement transfers fewer bytes than the record size, the remaining bytes in the record (including any nulls) are passed over.

If you specify no record size, WRITE FILE does not pad with nulls, and READ FILE does not pass over any bytes.

DOS and RDOS Disk Filenames

Each disk file created under BASIC in a DOS or RDOS operating system has a filename made up of one to ten characters. Legal characters include:

A through Z	
a through z	(converted to uppercase by the operating system)
0 through 9	
$ (dollar sign)	

In addition, you may append an optional one- or two-character alphanumeric extension to the filename which is separated from the filename by a period in the form, filename.ex.

Unlike RDOS utility programs such as MAC and RLDR, BASIC does not recognize any special two-character alphanumeric extensions. You can create extensions to suit your needs.

TEST.SR	is a meaningful way to signify a SouRce file.
TEST.CI	could signify the Core Image file obtained by SAVEing TEST.SR.
TEST.LS	could be a LiSting file output from the program.

AOS Disk Filenames

Each disk file created under BASIC in an AOS operating system has a filename of 1 to 31 characters. Legal characters include:

A through Z	
a through z	(converted to uppercase by the operating system)
0 through 9	
. (period)	
$ (dollar sign)	
_ (underscore)	

You can select any combination of legal characters to create a filename. A few examples of legal filenames are as follows:

```
FILE_NAME.NEW
SAVE.FILE$.SR
LIFE.JAN.5
```

Unlike AOS utility programs such as MASM (the macroassembler), BASIC does not recognize any special alphanumeric extensions. You can create filename extensions to suit your own needs.

TEST.SR	is a meaningful way to signify a SouRce file.
TEST.CI	could signify the Core Image file obtained by SAVEing TEST.SR.
TEST.LS	could be a LiSting file output from the program.

DOS and RDOS Reserved Filenames

Unit record devices and magnetic tape devices are given special names and do not have extensions. Devices with reserved names are listed below:

$TTI and $TTI1	Input consoles
$TTO and $TTO1	Output consoles
$CDR and $CDR1	Punched card readers
CTn	Cassette units ($0 < n < 17_8$)
$LPT and $LPT1	Line printers
MTn	Magnetic tape units ($0 < n < 17_8$)
$PLT and $PLT1	Incremental plotters (access via assembly language subroutines; see Appendix B)
$PTP and $PTP1	Paper tape punches
$PTR and $PTR1	Paper tape readers
QTY:n	Multiplexor consoles

For a complete list of RDOS reserved filenames, see the *RDOS CLI User's Manual.*

AOS Reserved Filenames

Unit record devices and magnetic tape devices are given special names and do not have extensions. Devices with reserved names are listed below:

@CDR, @CDR1,...@CDRn	First and succeeding card readers.
@CON0, @CON1,...@CONn	First and succeeding console display/keyboards or asynchronous communications lines.
@DPn through @DPn17	Moving head disk units 0 through 7 on the first controller, and 10 octal through 17 octal on the second controller. n is a single alphabetic character indicating the disk type. These types are described in the *AOS System Manager's Guide.*
@LPT, @LPT1,...@LPTn	First and succeeding line printers.
@MTA0 through @MTA17	Magnetic tape units 0 through 7 on the first controller, and 10 octal through 17 octal on the second controller.
@MCA and @MCA1	First and second multiprocessor communications adaptor controllers.
@PLT, @PLT1,...@PLTn	First and succeeding digital plotters.
@PTP, @PTP1,...@PTPn	First and succeeding paper tape punches.
@PTR, @PTR1,...@PTRn	First and succeeding paper tape readers.

AOS	√	S	√
RDOS	√	C	√
DOS	√	F	

OPEN FILE

assigns a file number and access mode to filename for future referencing in file I/O statements in your program.

Format

OPEN FILE(file,mode), "filename" *[,record size [,file size]]*

Arguments

file is a numeric expression which evaluates to a number from 0 through 7 for RDOS and DOS, 0 through 15 for AOS. BASIC uses this number to simplify the reference to "filename" in other file I/O statements.

mode is a numeric expression which evaluates to a number from 0 to 3. This number specifies the access mode of the file. The modes are described under *Remarks.*

filename is a string literal or string variable which evaluates to a filename.

record size is an optional numeric expression which evaluates to a fixed length (in bytes) for each record in a file. *record size* may be any value from 1 to 32768. In mode 0, a default value of 128 bytes per record is assigned.

When *record size* is specified (or the default record size in mode 0 is used), if the output of a WRITE FILE statement has fewer than the specified number of bytes, the output will be padded with nulls until it reaches the specified length. If the output has greater than the specified number of bytes, an error occurs.

file size is used only in RDOS and DOS (not AOS), and only in conjunction with *record size,* to create a contiguously organized file. *file size* is an optional numeric expression, which evaluates to the maximum number of records in the file, and thereby limits its size.

Remarks

1. You can calculate record length as follows:

   - Numeric Data

     Single-precision: 4 bytes per data item
     Double-precision: 8 bytes per data item

   - String Data

     One byte per character in string, plus one byte for string delimiter

   - Arrays

     (No. of rows)*(No. of columns)*(precision)

   NOTE: precision is 4 for single-precision, 8 for double-precision.

2. Modes 0 to 3 are described as follows:

   Mode 0 - Random-access file (for input and/or output), the file is not exclusively opened. Only disk files may be opened in random mode for reading and writing. If the disk file named filename does not exist in your directory, BASIC creates it. Record length is fixed by *record size* or by the default value (128 bytes).

   Mode 1 - Output (creates a new file for writing). You can open either a disk file or an appropriate output device in mode 1. The file is exclusively opened and only writes are permitted. If filename already exists in your directory the previous copy is deleted from the disk. In either case, a new file is created (initialized with 0 length). In RDOS and DOS only, the system does not check to insure that only writes are used. Use the CHATR command to prevent input use of an output file.

OPEN FILE (continued)

Mode 2 - Output (appends to an existing file). You can use this mode to open any file previously opened in mode 1 or mode 2. When an existing file is opened in mode 2, the file pointer moves to the end of the file so that subsequent data written to the file will extend it. If the file does not exist in your directory, it will be created. The file is exclusively opened, and only writes are permitted. In RDOS and DOS only, the system does not check to insure that only writes are used. Use the CHATR command to prevent input use of an output file.

Mode 3 - Input (for reading only). You can open either a disk file or appropriate input device in mode 3. If a disk file is opened in this mode, the file must already exist. Only reads are permitted from a file opened in mode 3, and the file is not exclusively opened. If the file is not found in your directory, BASIC searches for it in the library directory.

3. Default value for the mode argument is zero. OPEN FILE (1) is the same as OPEN FILE (1,0).

4. In RDOS and DOS you can't have a contiguous file with variable length records.

Examples

1. `*100 OPEN FILE (1,1), "NETSAK.JR")`

   This statement opens file 1, named NETSAK.JR, as an output file.

2. `*100 OPEN FILE (2,0), "RESSEHC.TO",20)`

   This statement opens the file named RESSEHC.TO as file number 2. Mode 0 specifies random access read or write; records are 20 bytes long.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

CLOSE FILE

disassociates a filename and a file number so that the file can no longer be referred to and the file number can be reused.

Format

CLOSE *[FILE (file)]*

Arguments

file is a numeric expression which evaluates to a file number previously associated with a filename in an OPEN FILE statement.

Remarks

1. You can use the CLOSE FILE statement to close a file, and then reOPEN it with a new mode argument.

2. The CLOSE form of the statement closes all open files.

Examples

```
*100  CLOSE FILE (1))
*200  CLOSE FILE (X+3))
*300  CLOSE)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

GPOS FILE

determines the current file pointer position in an open file.

Format

GPOS FILE (file), var

Arguments

file is a numeric expression which evaluates to a file number previously associated with an OPEN FILE statement.

var is the name of the variable to which BASIC will assign the current byte position value.

Remarks

We emphasize that this statement returns the value, in var, of the current byte position of the file pointer; this does not necessarily coincide with the beginning of a record position.

Examples

```
*10  GPOS FILE (2), B1)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

INPUT FILE

reads data in ASCII format from a disk file or device.

Format

INPUT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right),\begin{Bmatrix}\text{var}\\ \text{svar}\end{Bmatrix}\left[,\begin{Bmatrix}\textit{var}\\ \textit{svar}\end{Bmatrix}\right]\ldots$

Arguments

file is a numeric expression which evaluates to the number of a file opened for sequential access or for random access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

var and svar are numeric variables and string variables whose values are read from a file.

Remarks

1. The type of variable in the INPUT FILE variable list must correspond to the data type of the corresponding data item being read from the file.

2. The data must be formatted as for the INPUT statement, with commas separating data items and a carriage return at the end of a variable list.

3. The EOF function (see Chapter 4) may be used to detect an end of file in the file that is being read.

4. The first record number in a random-access file is 0.

Examples

```
*40  OPEN FILE (1,3), "$PTR")
*70  INPUT FILE (1), Z,Y,X,A$,B$)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

LREAD FILE

reads a string from a record, in either a random- or sequential-access file, which has a null, form feed, or carriage return (new-line) terminator.

Format

LREAD FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,svar *[,svar¹]*

Arguments

file is a numeric expression which evaluates to a file number previously associated with an OPEN FILE statement.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

svar is a string variable to which BASIC assigns the value of the string read from the file.

svar¹ is a string variable to which BASIC assigns the value of the delimiter for the string read. Valid delimiters are null, form feed, and carriage return (new-line).

Remarks

1. The maximum string length allowed for svar is 133 characters, which includes the delimiter. If the record read is longer than 133 characters, BASIC will set the length of *svar¹* to 0, and truncate the value of svar at 133 characters.

2. If the record read is shorter than 133 characters, then the length of *svar¹* is 1, and *svar¹* contains the delimiter.

3. The number of the first record in a random-access file is 0.

4. You can use the EOF function to detect the end of the file you're reading.

Examples

```
*LIST)
0010 DIM A$(60)
0020 REM THIS ROUTINE USES FILE TESTFILE1 CREATED
0030 REM FOR THE LWRITE FILE EXAMPLE.
0040 OPEN FILE (0,0),"TESTFILE1",50
0050 FOR I=1 TO 5
0060   LREAD FILE (0,I),A$,B$
0065   PRINT B$
0070   PRINT A$
0080 NEXT I
0090 CLOSE
*RUN)

MONDAY

TUESDAY

WEDNESDAY

THURSDAY

FRIDAY

END AT 0090
*
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

LWRITE FILE

writes a string to a record into either a random- or sequential-access file.

Format

LWRITE FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$, svar *[,svar^{1}]*

Arguments

file is a numeric expression which evaluates to a file number previously associated with an OPEN FILE statement.

record is a numeric expression which evaluates to the number of a record opened for random access.

svar is a string variable whose value BASIC writes to a file.

svar^{1} is a string variable which contains the value of the delimiter for the string written.

Remarks

1. The LWRITE statement is especially useful for creating records with nonstandard delimiters, or records in small pieces to be read later in larger chunks.

2. If you include *svar^{1}* in the argument list, and set its length to one, then BASIC assumes *svar^{1}* is one of the valid delimiters, and will output it as the string terminator.

3. If you set the length of *svar^{1}* to zero, then no delimiter will be output.

4. If you don't include *svar^{1}* in the argument list, then a null will be output as the string terminator.

Examples

```
*LIST)
0010 DIM A$(60)
0020 LET Z$="<13>"
0030 LET A$="MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY  "
0040 OPEN FILE (0,0),"TESTFILE1",50
0050 LET A=1
0060 LET J=10
0070 LET K=10
0080 FOR I=1 TO 5
0090     LWRITE FILE(0,I),A$(A,J),Z$
0100    PRINT A$(A,J)
0110     LET A=A+K
0120     LET J=J+K
0130 NEXT I
0140 CLOSE
*RUN)
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY

END AT 0140
*
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT INPUT FILE

reads a record of matrix data in ASCII format from a file.

Format

MAT INPUT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar *[,mvar]* ...

Arguments

file is a numeric expression which evaluates to the number of a file opened for sequential or random access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

mvar is a matrix array whose values are read from a record in a sequential- or random-access file.

Remarks

1. Previously dimensioned matrix arrays may be listed in the statement by name only. Matrix arrays which have not been dimensioned must be dimensioned in the MAT INPUT FILE statement.
2. Data items are read from the file sequentially and are assigned to the array elements by row.
3. Data items in the file must be separated by a comma or carriage return.
4. The EOF function may be used to detect an end of file in the file that is being read.
5. The first record in a random-access file is 0.

Examples

```
*05 DIM Y(7,6),Z(13,2))
*10 OPEN FILE (2,3), "XX.AA")
*50 MAT INPUT FILE (2),X(5,5),Y,Z)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT PRINT FILE

writes a record of matrix data in ASCII format into a sequential file or random-access file.

Format

MAT PRINT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar $\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\ \textit{mvar}\ldots\begin{Bmatrix},\\ ;\end{Bmatrix}\right]$

Arguments

file is a numeric expression which evaluates to the number of a file opened for sequential or random access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

mvar is a matrix whose values are written into a record of a random- or sequential-access file.

Remarks

1. This statement is intended for outputting to an ASCII device such as a line printer, or to a disk file for off-line printing.
2. The MAT INPUT FILE statement cannot be used to input data which was output by MAT PRINT FILE, because the MAT PRINT FILE statement does not output delimiters between matrix elements. Use the MAT WRITE FILE statement to output data that will later be re-input using the MAT READ statement.
3. If you use a semicolon after a matrix variable in the MAT PRINT FILE statement, rather than a comma or carriage return, it indicates that the matrix which immediately precedes the semicolon is printed in compact format rather than zone format.
4. The first record number in a random-access file is 0.

Examples

```
*05 DIM B(20,20))
*10 OPEN FILE (0,1),"NUHROK")
*20 MAT PRINT FILE (0), B)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT READ FILE

reads a record of data in binary format, for the elements of matrix arrays, from a sequentially-accessed file or from a randomly-accessed file.

Format

MAT READ FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file, record}\end{Bmatrix}\right)$, mvar *[,mvar]* ...

Arguments

file is a numeric expression which evaluates to the number of a file opened for random access or for sequential access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

mvar is a matrix which is assigned values read sequentially from a randomly-accessed record or sequentially-accessed record.

Remarks

1. The number of the first record in a random-access file is 0.
2. Previously dimensioned matrix arrays may be listed in the statement by name only. Matrix arrays which have not been dimensioned must be dimensioned in the MAT READ FILE statement.
3. In random-access files, records which have not been written into will contain all zeros when read.
4. Data items are read from the record sequentially and are assigned to the array elements by row.
5. The EOF function may be used to detect an end of file in the file that is being read.
6. The amount of data to be read must not exceed the record size specification for files OPENed for random access.

Examples

```
*10  DIM A(7,3), B(12,7))
*30  OPEN FILE (1,3), "MATRIXA")
*40  MAT READ FILE (1), A, B, C(3,4), D(5))
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

MAT WRITE FILE

writes a record of matrix data in binary form into a sequential-access file or a random-access file.

Format

MAT WRITE FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$, mvar *[,mvar]* ...

Arguments

file is a numeric expression which evaluates to the number of a file opened for random access or for sequential access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

mvar is a matrix whose values are written into a record of a random-access file or a sequential-access file.

Remarks

1. The number of the first record in a random-access file is 0.
2. Matrix arrays listed in the MAT WRITE FILE statement must be previously dimensioned.

Examples

```
*50  OPEN FILE (0,1), "AAA")
*80  MAT WRITE FILE (0),B,C,X)
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

PRINT FILE

writes data in ASCII format into a sequential- or random-access file.

Format

$$\left\{\begin{array}{l};\\ \text{PRINT}\end{array}\right\}\text{FILE}\left(\left\{\begin{array}{l}\text{file}\\ \text{file,record}\end{array}\right\}\right),\left\{\begin{array}{l}\text{expr}\\ \text{var}\\ \text{svar}\\ \text{“str lit”}\end{array}\right\}\left[\left\{\begin{array}{l},\\ ;\end{array}\right\}\left\{\begin{array}{l}\textit{expr}\\ \textit{var}\\ \textit{svar}\\ \textit{“str lit”}\end{array}\right\}\right]\cdots\left[\left\{\begin{array}{l},\\ ;\end{array}\right\}\right]$$

Arguments

file is a numeric expression which evaluates to the number of a file opened for sequential or random access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

expr, var, svar, and str lit... make up a list of one or more numeric expressions, numeric variables, string variables, and string literals, whose values are written into a file.

Remarks

1. This statement is intended for outputting to an ASCII device such as a line printer, or to a disk file for later off-line printing.
2. Each item in the expression list must be separated from the next by a comma, semicolon, or carriage return. Output formatting is identical to that discussed in *Remarks* pertaining to the PRINT statement.
3. If the data written by PRINT FILE is to be subsequently read as numeric data by an INPUT FILE statement, then it is necessary to print the comma (data separator) as a string literal between expressions as shown in line 300 of example 1 below.
4. The first record number in a random-access file is 0.

Examples

1.
```
*010  OPEN FILE(3,1), "$LPT")
*100  PRINT FILE(3), "OUT6")
*200  PRINT FILE(3),"X=";X,"X SQUARED=";X↑2)
*300  PRINT FILE(3),A;",";B;",";C)
*400  CLOSE)
```

2.
```
*010  OPEN FILE(3,1), "$LPT", 80)
*100  FOR I = 1 TO 10000)
*110  PRINT FILE(3),I;)
*120  NEXT I)
*130  CLOSE)
```

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

PRINT FILE USING

outputs the values of the expressions in the PRINT FILE USING statement to a previously OPENed file in the format specified.

Format

$$\left\{\begin{array}{l};\\ \text{PRINT}\end{array}\right\}\text{FILE(file }\textit{[,record]}\text{),USING format,expr }\textit{[,expr]}\cdots$$

Arguments

file is a numeric expression which evaluates to the number of a previously opened file (see PRINT FILE).

record is a numeric expression which evaluates to the number of a record in a file.

format is a string literal or string variable which specifies the output format (see *Remarks*) for items in the expr list.

expr... make up a list of one or more numeric expressions, numeric variables, string variables, and string literals whose values will be written into a file.

Remarks

The remarks listed for the PRINT FILE statement and the argument descriptions in the PRINT USING statement (Chapter 3) all apply to the PRINT FILE USING statement.

AOS	√		S	√
RDOS	√		C	√
DOS	√		F	

READ FILE

reads data in binary format from a sequentially-accessed file or a randomly-accessed file.

Format

$$\text{READ FILE}\left(\begin{Bmatrix}\text{file}\\ \text{file, record}\end{Bmatrix}\right),\begin{Bmatrix}\text{var}\\ \text{svar}\end{Bmatrix}\left[,\begin{Bmatrix}\mathit{var}\\ \mathit{svar}\end{Bmatrix}\right]\ldots$$

Arguments

file is a numeric expression which evaluates to the number of a file opened for random or sequential access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

var and svar are numeric variables and string variables which are assigned values read sequentially from a randomly-accessed or sequentially-accessed record.

Remarks

1. The type of variable in the READ FILE variable list must correspond to the data type of the corresponding data item being read from the record.
2. The number of the first record in a random-access file is 0.
3. In random-access files, records which have not been written into will contain all zeros when read. An attempt to read a record which is after the last record written will cause an end-of-file condition. A RESET FILE can be used to continue processing.
4. The EOF function (see Chapter 4) may be used to detect an end-of-file condition in the file that is being read.

Examples

```
*LIST)
0001 REM-READ FILE
0005 TAB =10
0010 DIM B(3,4)
0020 OPEN FILE(1,0),"TESTFILE",20
0030 FOR I=1 TO 12
0040      LET I1=INT((I-1)/4)+1
0050      LET J1=I-(4*(I1-1))
0060      READ FILE(1,I),B(I1,J1)
0070 NEXT I
0080 MAT PRINT B
0090 CLOSE
*RUN)
36      33      30      27
24      21      18      15
12      9       6       3

END AT 0090
*
```

NOTE: This program uses the file TESTFILE which was created in the program example provided with the WRITE FILE statement.

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

RESET FILE

positions the file pointer to the beginning of a file.

Format

RESET *[FILE (file)]*

Arguments

file is a numeric expression which evaluates to a file number previously associated with an OPEN FILE statement.

Remarks

1. You can use the RESET FILE statement to reset the position of the file pointer to the beginning of a file without having to CLOSE and reOPEN the file.
2. The RESET (without *file*) form of the statement repositions the file pointers for all open files in your program to the beginning of their files.

Examples

```
*ENTER "RESET")
*LIST)
0010 DIM B(3,4)
0020 OPEN FILE (1,0), "TESTFILE",20
0030 FOR I=1 TO 12
0040      LET I1=INT((I-1)/4)+1
0050      LET J1=I-(4*(I1-1))
0060      READ FILE (1,I),B(I1,J1)
0065 NEXT I
0070 GPOS FILE (1),B1
0080 PRINT B1;
0086 RESET FILE (1)
0087 GPOS FILE (1),B1
0088 PRINT B1
0090 CLOSE
0095 PRINT
0100 MAT PRINT B
*
*LIST "RESET")
TYPE CR TO DELETE OLD:
*RUN)
248 0

36      33      30      27
24      21      18      15
12      9       6       3

END AT 0100
*AUDIT)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

SPOS FILE

moves the file pointer to the byte position specified by expr.

Format

SPOS FILE (file),expr

Arguments

file is a numeric expression which evaluates to a file number previously associated with an OPEN FILE statement.

expr is a numeric expression which evaluates to the number of a byte position in a file.

Remarks

We emphasize that this statement moves the file pointer to a byte position, and not necessarily to a record position. Therefore, you must be certain that the value of expr is indeed the calculated position you intend to place the file pointer.

Examples

```
*100 SPOS FILE (1), I+132)
```

AOS	√
RDOS	√
DOS	√

S	√
C	√
F	

WRITE FILE

writes a record of data in binary format into a sequential-access file or a random-access file.

Format

$$\text{WRITE FILE}\left(\begin{Bmatrix}\text{file}\\ \text{file, record}\end{Bmatrix}\right),\begin{Bmatrix}\text{expr}\\ \text{var}\\ \text{svar}\\ \text{"str lit"}\end{Bmatrix}\left[,\begin{Bmatrix}\textit{expr}\\ \textit{var}\\ \textit{svar}\\ \textit{"str lit"}\end{Bmatrix}\right]\ldots$$

Arguments

file is a numeric expression which evaluates to the number of a file opened for random or sequential access.

record is a numeric expression which evaluates to the number of a record in a file opened for random access.

expr, var, svar, and str lit... form a list of one or more numeric expressions, numeric variables, string variables and literals whose values are written as a record into a sequential-access file or a random-access file.

Remarks

1. The first record number in a random-access file is 0.

2. Data files you created using WRITE FILE statements can be accessed by READ FILE statements but not by INPUT FILE statements.

Examples

```
*LIST)
0001 REM-FILE WRITE
0010 DIM A(3,4)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 4
0040      LET A(I,J)=((I-1)*4+J)*3
0050   NEXT J
0060 NEXT I
0070 MAT PRINT A
0080 PRINT
0090 OPEN FILE(1,0),"TESTFILE",20
0100 FOR I1=1 TO 3
0110   LET I=4-I1
0120   FOR J1=1 TO 4
0130      LET J=5-J1
0140      LET R=(3-I)*4+(5-J)
0150      WRITE FILE(1,R),A(I,J)
0160     PRINT A(I,J),
0170   NEXT J1
0180   PRINT
0190 NEXT I1
0200 CLOSE
*RUN)
3      6      9      12
15     18     21     24
27     30     33     36

36     33     30     27
24     21     18     15
12     9      6      3

END AT 0200
*
```

End of Chapter

Appendix A
Error Messages

Extended BASIC error messages are printed as two-digit codes, followed by a brief explanatory message. The following categories of errors may occur when operating BASIC.

1. Errors recognized by BASIC during program input (Table A-1).

   a. If an error is detected in a statement input from a terminal, the error message refers to the last statement typed.

   b. If the statement in error was input from a file or other input device, BASIC prints the incorrect statement followed by the error message.

   c. All syntax errors are recognized during program input.

   d. The form of the error message is:

      ERROR xx text

      where:

      xx is a two-digit decimal error code.

      text is a brief description of the error.

2. Runtime errors (except file I/O) (Table A-1).

   BASIC system runtime errors cause printout of an error message in the following form:

   ERROR *[xx AT yyyy]* text

   where:

   xx is a two-digit decimal error code.

   yyyy is the line number at which the error occurred, if used in a statement.

   text is a brief description of the error.

3. RDOS/DOS Extended BASIC File I/O errors (Table A-2).

   The format for file I/O error messages is as follows:

   I/O ERROR xx *[AT yyyy]* text

   where:

   xx is a two-digit decimal error code.

   yyyy is the line number at which the file I/O error occurred, if used in a statement.

4. AOS Extended BASIC File I/O errors. Please refer to the *AOS Programmer's Reference Manual*, Appendix A, for I/O error messages.

Table A-1. BASIC Error Messages

Code	AOS Text	RDOS/DOS Text	Meaning
00	INVALID OPERATOR	FORMAT	Unrecognizable statement format.
01	CHARACTER NOT RECOGNIZED	CHARACTER	Illegal ASCII character or unexpected character.
02	ILLEGAL STATEMENT SYNTAX	SYNTAX	Invalid syntax or argument type.
03	DATA TYPES DON'T MATCH	READ/DATA TYPES	READ specifies different data type than DATA statement.
04	HARDWARE OR SOFTWARE FAULT	SYSTEM	Hardware or software malfunction.
05	MISSING OR ILLEGAL LINE NUMBER	LINE NUMBER	Statement number not in the range 1 <= n <= 9999.
06	348 VARIABLES HAVE ALREADY BEEN DEFINED	EXCESSIVE VARIABLES	Attempt to declare too many variables.
07	KEYWORD NOT VALID AS COMMAND	COMMAND	Attempt to execute an illegal command.
08	SINGULAR MATRIX - CANNOT BE INVERTED	SINGULAR MATRIX	Attempt to invert a singular matrix.
09	FILE CANNOT BE LOADED - WRONG REVISION		Core image file incompatible with system.
10	ILLEGAL ATTRIBUTE	ATTRIBUTE	Attempt to assign an illegal attribute to a file.
11	UNMATCHED PARENTHESIS	PARENTHESIS	Parentheses in an expression are not paired properly.
12	MANTISSA OVERFLOW		Hardware or software malfunction.
13	ARITHMETIC UNDERFLOW		Result of arithmetic expression is too small.
14	PROGRAM OVERFLOW	PGM OVFL	Not enough storage to ENTER source program.
15	END OF DATA	END OF DATA	Not enough DATA arguments to satisfy READ.
16	ARITHMETIC	ARITHMETIC	Value too large or too small to evaluate; or a divide by 0.
17	ARITHMETIC OVERFLOW		Result of arithmetic expression too large.
18	GOSUB NESTING	GOSUB NESTING	More nested GOSUBs than specified at BASIC system generation.
19	RETURN - NO GOSUB	RETURN - NO GOSUB	RETURN statement encountered without a corresponding GOSUB.
20	FOR NESTING	FOR NESTING	More nested FORs than specified at BASIC system generation.

Table A-1. BASIC Error Messages (continued)

Code	AOS Text	RDOS/DOS Text	Meaning
21	FOR - NO NEXT	FOR - NO NEXT	Unexecutable FOR-NEXT loop; FOR without a NEXT.
22	NEXT - NO FOR	NEXT - NO FOR	NEXT statement encountered without a corresponding FOR.
23	DATA OVERFLOW	DATA OVFL	Not enough storage left to assign space for variables.
24	ATTEMPT TO DIVIDE BY ZERO		Attempt to divide by 0.
24		DIRECTORY EMPTY	No files in your directory.
25	FEATURE NOT AVAILABLE	OPTION	Feature specified not available.
26	(not used)	(not used)	
27	ILLEGAL FILE NUMBER	FILE NUMBER	Invalid file designation in an I/O statement.
28	UPWARD RE-DIMENSION	DIM OVFL	An array or string exceeds its original dimensions.
29	EXPRESSION IS TOO COMPLEX FOR EVALUATION	EXPRESSION	An expression is too complex for evaluation.
30	MODE	ILLEGAL FILE MODE	Invalid mode designation in an I/O statement.
31	SUBSCRIPT OUT OF BOUNDS	SUBSCRIPT	Subscript exceeds array's dimensions.
32	UNDEFINED FUNCTION	UNDEFINED FUNCTION	Attempt to use a function never defined by DEF.
33	FUNCTION NESTING	FUNCTION NESTING	User function nesting exceeds BASIC systems generation specification.
34	FUNCTION ARGUMENT	FUNCTION ARGUMENT	Argument range out of bounds.
35	ILLEGAL MASK	ILLEGAL MASK	PRINT USING format is illegal.
36	CANNOT EXECUTE COMMANDS NOW	NO COMMANDS NOW	An ENTERed file has a command instead of a statement.
37	USER ROUTINE NOT FOUND	USER ROUTINE	CALL statement specifies a user routine not in storage.
38	(not used)	(not used)	
39	DUPLICATE MATRIX	DUP MATRIX	Same matrix appears on both sides of a MAT multiply or the transpose statement.
40	MATRIXES ARE NOT THE SAME SIZE	MATRIXES SIZES	Matrixes have different sizes.
41	UNDIMENSIONED VARIABLE	UNDIMENSIONED VARIABLE	Attempt to use an undimensioned matrix.
42	(not used)	(not used)	

Table A-1. BASIC Error Messages (continued)

Code	AOS Text	RDOS/DOS Text	Meaning
43	MATRIX NOT SQUARE	MATRIX NOT SQUARE	Attempt to invert a nonsquare matrix.
44	(not used)	(not used)	
45	DATA IS GREATER THAN SPECIFIED RECORD SIZE	DATA > LRECL	Logical record length limit exceeded.
46	MORE DATA SUPPLIED THAN REQUESTED	INPUT	Too many responses to [MAT] INPUT.
47	CHECKSUM	(not used)	File did not load correctly.
48	NOT A CORE IMAGE FILE	(not used)	A filename not created by SAVE was specified in a LOAD, RUN, or CHAIN command.
49	(not used)	NO ROOM FOR DIRECTORY	A FILE or LIBRARY command cannot find 256 words in your program storage area to read the disk directory.
50	(not used)	(not used)	
51	(not used)	USER NOT ACTIVE	Attempt to send message to an inactive or nonexistent user.
52	(not used)	USER IN NOMSG STATE	Attempt to send message to user whose terminal is in NOMSG state.
53	(not used)	(not used)	
54	GENERATED STATEMENT IS GREATER THAN 132 BYTES	STATEMENT LENGTH	A statement exceeded 132 characters in either internal or ASCII format, when expanded.
55	EXECUTE-ONLY	EXECUTE-ONLY	Attempt to examine a program originating from a file with the execute-only attribute.
56	RANGE	RANGE	Attempt to refer to a random record beyond 262,144.
57	(not used)	(not used)	
58	INCOMPATIBLE CORE IMAGE FILE	INCOMPATIBLE CORE IMAGE	Attempt to LOAD a core image file SAVEd under a different version of BASIC.
59	ZERO STEP	ZERO STEP	FOR-NEXT with step 0.
60	KEYBOARD RESPONSE NOT IN TIME	TIME-OUT	Timed input decremented to 0.
61	INVALID DECIMAL STRING	INVALID DECIMAL STRING	Attempt to perform string arithmetic with nonnumeric characters.
62	STRING ARITHMETIC OVERFLOW	STAR OVFL	The result of string arithmetic requires more than 18 digits for precision representation.
63	(not used)	(not used)	
64	(not used)	SYSTEM ACTIVE	Attempt by system manager to execute a BYE command while people were still on system.

Table A-2. RDOS/DOS Extended BASIC File I/O Error Messages

Code	Text	Meaning
01	ILLEGAL FILENAME	A to Z, 0 to 9 and $ are only valid characters.
02	ILLEGAL SYSTEM COMMAND	Command not defined in operating system.
03	ILLEGAL COMMAND FOR DEVICE	INIT "$PTR", WRITE to $CDR, etc.
04	NOT A CORE IMAGE FILE	File not in SAVE format.
06	END OF FILE	Attempt to read beyond EOF marker.
07	READ PROTECTED FILE	Attempt to read from a read-protected file.
08	WRITE PROTECTED FILE	Attempt to write to a write-protected file.
09	FILE ALREADY EXISTS	Attempt to create an existent file.
10	FILE NOT FOUND	Attempt to refer to a nonexistent file.
11	PERMANENT FILE	Attempt to alter a permanent file.
12	ATTRIBUTE PROTECTED	Attempt to change file attributes when file is protected with RDOS attribute A.
13	FILE NOT OPENED	Attempt to refer to an unopened file.
14	SWAPPING DISK DATA CHECK	Disk error on swapping file.
15	REVISION CHECK	Object file of LOAD or CHAIN not created by this revision of BASIC.
16	CHECKSUM	Disk error.
17	CHANNEL NOT AVAILABLE	Open two files with the same file number, or attempt to open too many files. Operating system file pool overflowed.
18	LINE LIMIT	Line limit exceeded on read or write line.
20	PARITY	Parity error on read line.
23	NO FILE SPACE	Out of disk space. Delete files to make more room.
24	READ ERROR	File read error.
25	SELECT STATUS	Unit not ready or is write-protected.
29	DIFFERENT DIRECTORIES	Files specified on different directories.
30	ILLEGAL DEVICE CODE	Device not in system or illegal device code.
31	ILLEGAL OVERLAY	This is an unexpected system software error. If it occurs, please notify your local Data General representative.
37	DEVICE ALREADY INITIALIZED	Device already INITed.
38	INSUFFICIENT CONTIGUOUS BLOCKS	Insufficient number of free contiguous disk blocks. Reorganize partition.

Table A-2. RDOS/DOS Extended BASIC File I/O Error Messages (continued)

Code	Text	Meaning
41	NO MORE DCB'S	Attempt to open more devices or directories than are configured in the operating system.
42	ILLEGAL DIRECTORY SPECIFIER	Illegal directory specifier.
43	UNKNOWN DIRECTORY SPECIFIER	Directory specifier unknown.
44	DIRECTORY TOO SMALL	Directory is too small (operator only). Minimum directory size is 48 blocks.
45	DIRECTORY DEPTH	Directory depth exceeded (operator only).
46	DIRECTORY IN USE	Attempt to release a directory in use by another program.
47	LINK DEPTH	Link depth exceeded.
48	FILE IN USE	Contact System Operator if file is in your directory.
52	FILE POSITION	Attempt to read out of bounds.
54	DIRECTORY NOT INITIALIZED	Directory/device not initialized.
58	DIRECTORY SHARED	No file space left.
69	DISK IS FULL	No file space left.

End of Appendix

Appendix B
Calling an Assembly Language Subroutine from Extended BASIC

You can call a subroutine written in assembly language from an Extended BASIC program. The format of the BASIC call is:

CALL sub# *[, A_1, ..., A_n]*

Where:

sub# is a numeric expression evaluating to a positive integer (in the range 0 to 32767) representing the subroutine number.

A_1, ..., A_n are optional arguments to be passed to the subroutine (*n* must be in the range 1 to 8) and may be arithmetic variables or expressions, or string variables or expressions. Dimensioned numeric variable names should include subscripts. (Statement numbers are not permitted as arguments.)

Character String Storage and Definitions

You must refer to the following information if you wish to handle character strings in a CALLed subroutine. BASIC keeps a count of the number of characters currently defined in each string variable (referred to as the current length of the string variable). The current length is stored as part of the header immediately preceding the contents of each string variable. (See Figure B-1.) The current length must be updated each time characters are added to or taken away from the string variable.

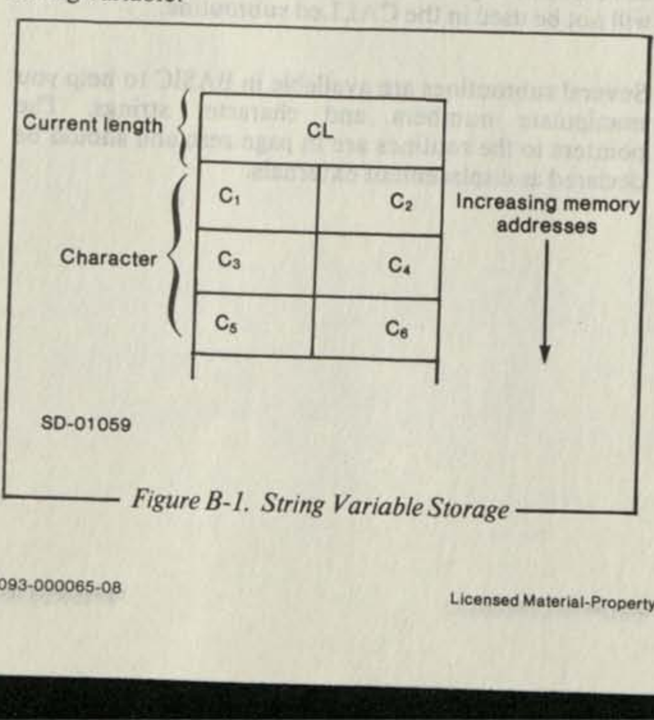


Figure B-1. String Variable Storage

In the following examples, assume that A$ is dimensioned to 10, and A$ = "ABCDE". The current length of A$ is 5.

A *substring* is any contiguous part of a string variable. For example:

A$(2,4) and A$ are substrings of A$

The *current length of a substring* is the number of defined characters within the substring. For example, the current length of A$(4,7) is 2, if only A$(4,4) and A$(5,5) are defined.

The *maximum length of a substring* is the number of character positions within the substring. For example, the maximum length of substring A$(4,7) is 4.

Linking the Assembly Language Subroutine

Improper use of assembly language subroutines, system calls, or task calls can crash the system.

Assembly language subroutines must be submitted to the System Manager at system load time. The subroutines in a file named SBRTB.RB are input to the relocatable loader when the BASIC system save file is created. You must include a subroutine table with your subroutines. The table must have the entry point SBRTB.

The subroutine table is a list of all assembly language subroutines available to a BASIC program. For each assembly language subroutine, a four-word list is required in the table containing the following:

- subroutine number
- subroutine entry point
- number of arguments
- argument control word

You terminate the table by using a subroutine number of -1.

The argument control word is used by BASIC to check runtime errors on the types of arguments. The control word is divided into eight two-bit fields for the eight possible arguments A_1 ...A_8. The value of the two-bit field determines the allowable argument.

00	argument may be any string expression
01	argument must be a string variable
10	argument may be any numeric expression
11	argument must be a numeric variable

The argument control word is written in an assembly language program such that the arguments are connected by a plus (+) sign and are described as shown in Figure B-2.

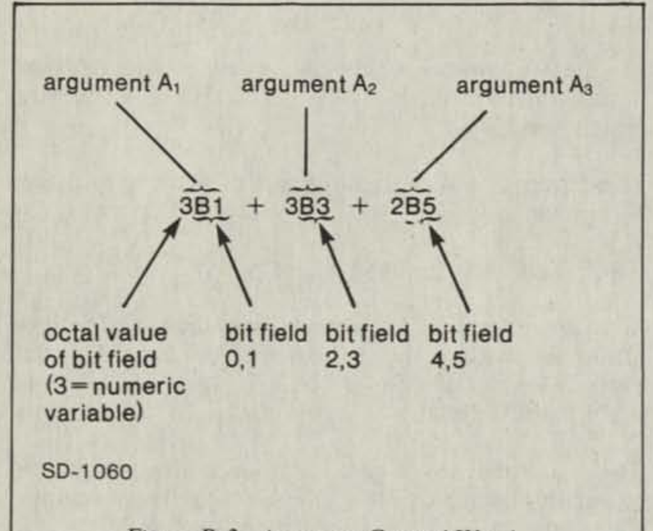


Figure B-2. Argument Control Word

If you use an argument to return a value to the calling program, you must set the argument's flags to either a 01 (for string) or 11 (for numeric).

BASIC calls the assembly language subroutines by the sequence:

```
        LDA     2,.+2     ;AC2 POINTS TO TOP
                          ;OF ADDRESS LIST
        JMP     <SUB>     ;JMP TO ASSEMBLY
                          ;LANGUAGE SUBROUTINE
        ADLST

ADLST:  <arg A1>
        <arg A2>
         .
         .
         .
        <arg An>
        JMP     BASIC     ;RETURN TO BASIC
                          ;INTERPRETER
```

If A_n is a substring of a string variable, the address list contains the address of the string descriptor words that contain the following information:

word 1	byte address of the first character of the substring
word 2	current length of the substring
word 3	maximum length of the substring
word 4	word address of the current length of the string variable

If A_n is a string expression, the address list contains the address of the string descriptor words that contain the following information:

word 1	byte address of the first character of the string
word 2	length of the string

If A_n is a numeric variable, the address list contains the storage address of the variable. (All numeric variables are represented in standard floating-point format.)

If A_n is a numeric expression, the address list contains the storage address of the value of the expression.

The following example shows legal and illegal calls to a subroutine, the subroutine table, and the subroutine itself. The argument list in a BASIC call to this subroutine must match the argument control word specified in the subroutine table.

An illegal CALL will result from an attempt to pass a variable in the CALL that does not have a previously assigned value. All variables passed in the CALL must have been assigned values even if their current value will not be used in the CALLed subroutine.

Several subroutines are available in BASIC to help you manipulate numbers and character strings. The pointers to the routines are in page zero and should be declared as displacement externals.

```
                        .TITLE  SBRTB   ; BASIC ASSEMBLY LANGUAGE SUBROUTINES
                        .ENT    SBRTB   ; ENTRY POINT : SBRTB
                        .NREL           ; NORMAL RELOCATABLE CODE

; SUBROUTINE TABLE

SBRTB:          1                       ; SUBROUTINE NUMBER
                SUB1                    ; SUBROUTINE ENTRY POINT
                2                       ; NUMBER OF ARGUMENTS
                3B1 + 3B3               ; ARGUMENT CONTROL WORD, BOTH ARGS ARE
                                        ; NUMERIC VARIABLES
                -1                      ; END OF TABLE

SUB1:
; CALLING SEQUENCE:  CALL 1,A,B
; THIS ROUTINE IS THE EQUIVALENT OF LET B = A
; THIS ROUTINE IS NOT REENTRANT
                STA     2,RET           ; SAVE ADDRESS LIST
                LDA     3,0,2           ; ADDRESS OF ARG 1
                LDA     3,0,3           ; WORD 1 OF ARG 1
                LDA     2,1,2           ; ADDRESS OF ARG 2
                STA     3,0,2           ; WORD 1 OF ARG 1 TO WORD 1 OF ARG 2
                LDA     2,RET           ; ADDRESS LIST
                LDA     3,0,2           ; ADDRESS OF ARG 1
                LDA     3,1,3           ; WORD 2 OF ARG 1
                LDA     2,1,2           ; ADDRESS OF ARG 2
                STA     3,1,2           ; WORD 2 OF ARG 1 TO WORD 2 OF ARG 2
                LDA     3,RET           ; ADDRESS LIST = RETURN ADDRESS -2
                JMP     2,3             ; RETURN TO BASIC (2 = NO. OF ARGS)
RET:            .BLK    1
                .END
```

Figure B-3. Example of an Assembly Language Subroutine

The following routines are helpful when linking your assembly language subroutines. In systems having floating-point hardware, the floating-point number is stored and returned in the Floating-Point Accumulator (FPAC) rather than in AC0-AC1.

Routines	Result

.FIX	Converts floating-point number in AC0-AC1 (or floating-point AC0 in the case of hardware floating-point support) to an integer in AC0-AC1. If there is overflow, the largest possible integer is returned in AC0-AC1. Bit 0 of AC0 is the sign of the number. Bit 0 of AC1 is a significant bit. There are two returns from .FIX:
	
	return 1: overflow
	return 2: OK

.FLOT	Converts an integer in AC0-AC1 to floating-point format in AC0-AC1.
.ADDF F0+F1 .SUBF F0-F1 .MPYF F0*F1 .DIVF F0/F1	Arithmetic routines to perform floating-point add, subtract, multiply, divide. In each routine, AC0-AC1 initially contains the floating-point value of F1 and AC2 contains the address of the value of F0. The result is returned in AC0-AC1. Underflow returns a zero result; overflow results in error number 16.

Routines	Result
.MPY A1*A2 A0,A1 .MPYA A0+A1*A2 A0,A1	In the integer multiply routines, AC1 contains the unsigned integer multiplicand and AC2 contains the unsigned integer multiplier. The result is a double length product with high-order bits in AC0 and low-order bits in AC1. Contents of AC2 are unchanged. The difference between the routines is that .MPYA adds the result of the multiplication to the contents of AC0.
.DVD (A0,A1)/A2 A1,A0 .DVDI A1/A2 A1,A0	In the integer divide routines the dividend is an AC1 (single-length) or in AC0 and AC1 (double-length with high-order bits in AC0). The divisor is in AC2 and the result is left with the quotient in AC1 and the remainder in AC0. Contents of AC2 are unchanged.

End of Appendix

Appendix C
Programming on Mark-Sense Cards

You may write BASIC programs on Data General's Extended BASIC mark-sense programming cards for input to the mark-sense card reader.

You may mark a stack of cards to include an entire BASIC program, and input your stack to the card reader as a batch job. Your system manager will know about any special cards your system may require.

The mark-sense reader has an option which permits either markings or punches. With this option, you may punch mark-sense cards. Marked and punched cards may be intermixed in a deck; a single card may be both marked and punched. You must use a No. 2 pencil to mark cards.

A Data General Extended BASIC mark-sense card has 37 columns, as shown in Figure C-1. The first four columns assign statement numbers; the next three assign the BASIC statement keyword. A single BASIC statement or part of a statement may be written on each card.

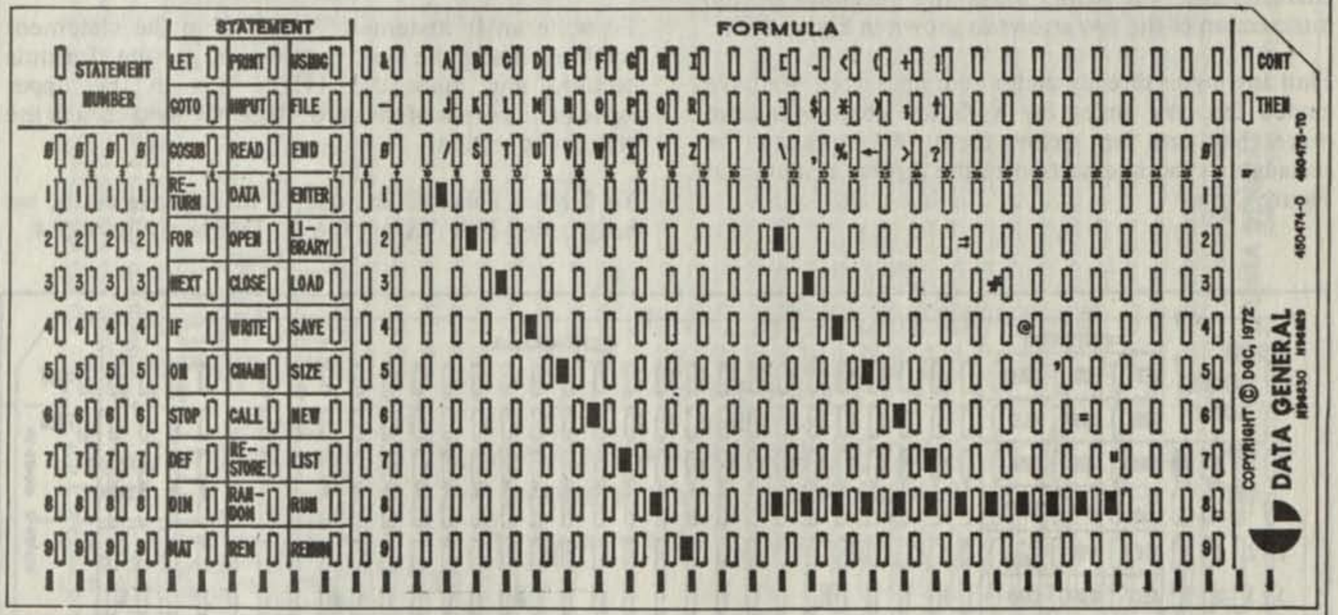


Figure C-1. Data General Extended BASIC Mark-Sense Programming Card

The BASIC statement field of the mark-sense card is three columns which allow all possible combinations of statement keywords. Cards are marked in the appropriate column; for example, we have marked the statement 450 GOTO 200 in Figure C-2.

The formula section of the card is 29 columns long, and 12 rows deep. You proceed from the left-most column to the right, up to 29 characters; the CONT box on the far right allows you to continue your statement on the next card.

You must fill out the formula section of each card in Hollerith code. Each Data General mark-sense card contains a Hollerith code key (the black squares in the formula section), which indicates the lines that you must mark for each character. On all mark-sense cards numbers are marked directly in the appropriate row, without the key. Letters require two marks in a row, and special characters, either two or three marks.

On mark-sense cards, find the character column and the character you want to mark. We've indicated the first column with an arrow and the letter V as the character we will mark. Mark the rectangle at the intersection of the two arrows as shown in Figure C-3a.

Find any boxes directly under your character. We have circled the box under the V. Some characters have more than one box below them. Again, mark the rectangle at the intersection of the arrows as shown in Figure C-3b.

Do not draw the arrows on the mark-sense cards as BASIC will try to interpret them. The first column contains the completed markings for the character V as shown in Figure C-3c.

Move over one column and repeat this process for your next character.

You may be using cards without a key. If so, fill them out according to the Hollerith character set in Appendix D. The mark-sense card key and the Hollerith character set work exactly the same way; you may use whichever you find easier. If you use the Hollerith code set, the top horizontal line is number 12, the second from the top is number 11, and the other lines are numbered from 0 through 9. To indicate 4, put a mark on line 4; to indicate an asterisk (*), put marks on lines 11, 4, and 8; to indicate a number sign (#), put marks on lines 3 and 8.

On any card, you can continue a statement to the next card by marking the CONT box in the upper right-hand corner of the first card. Continue the statement on the following card in the FORMULA section.

To write an IF statement, mark IF in the statement section, mark the test expression in the formula section, and mark the THEN box in the upper right-hand corner of the card. Begin the next card in the statement section.

To further illustrate the use of mark-sense cards, we have coded 10 IF V$ = "CAT" THEN in Figure C-4.

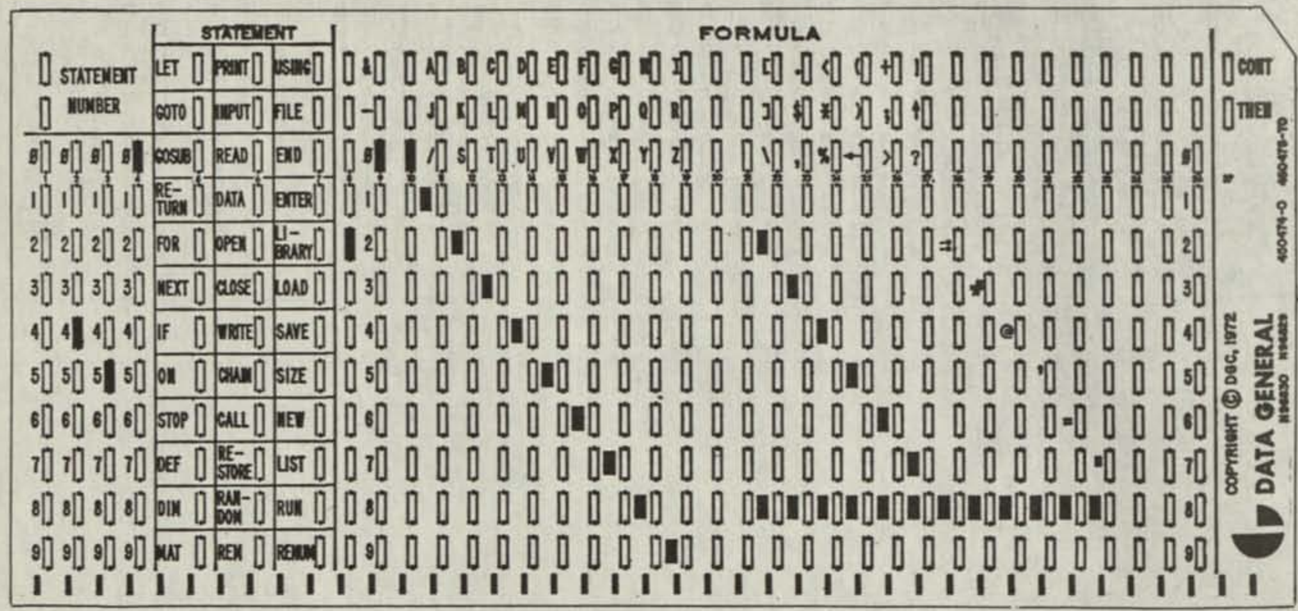


Figure C-2. 450 GOTO 200

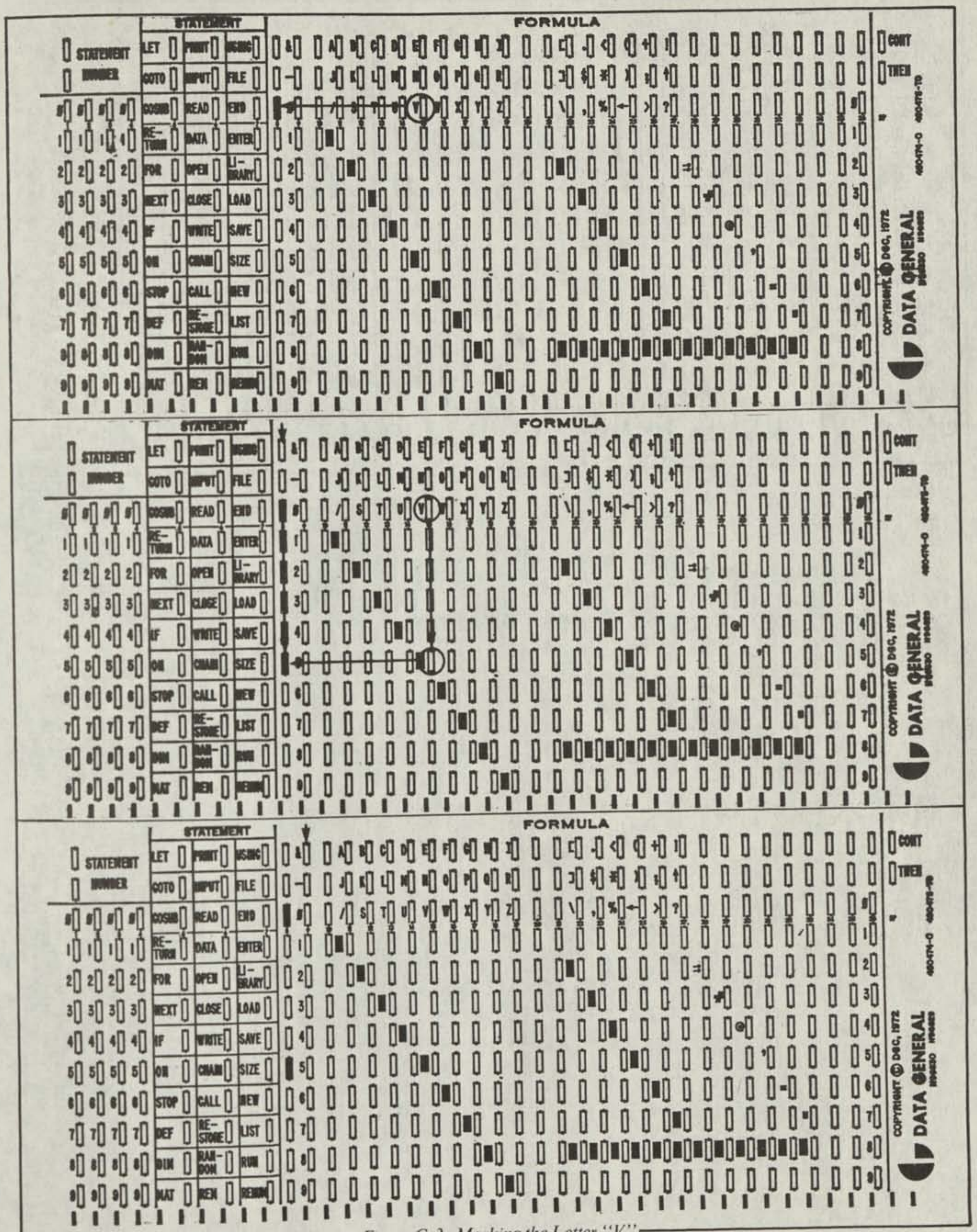


Figure C-3. Marking the Letter "V"

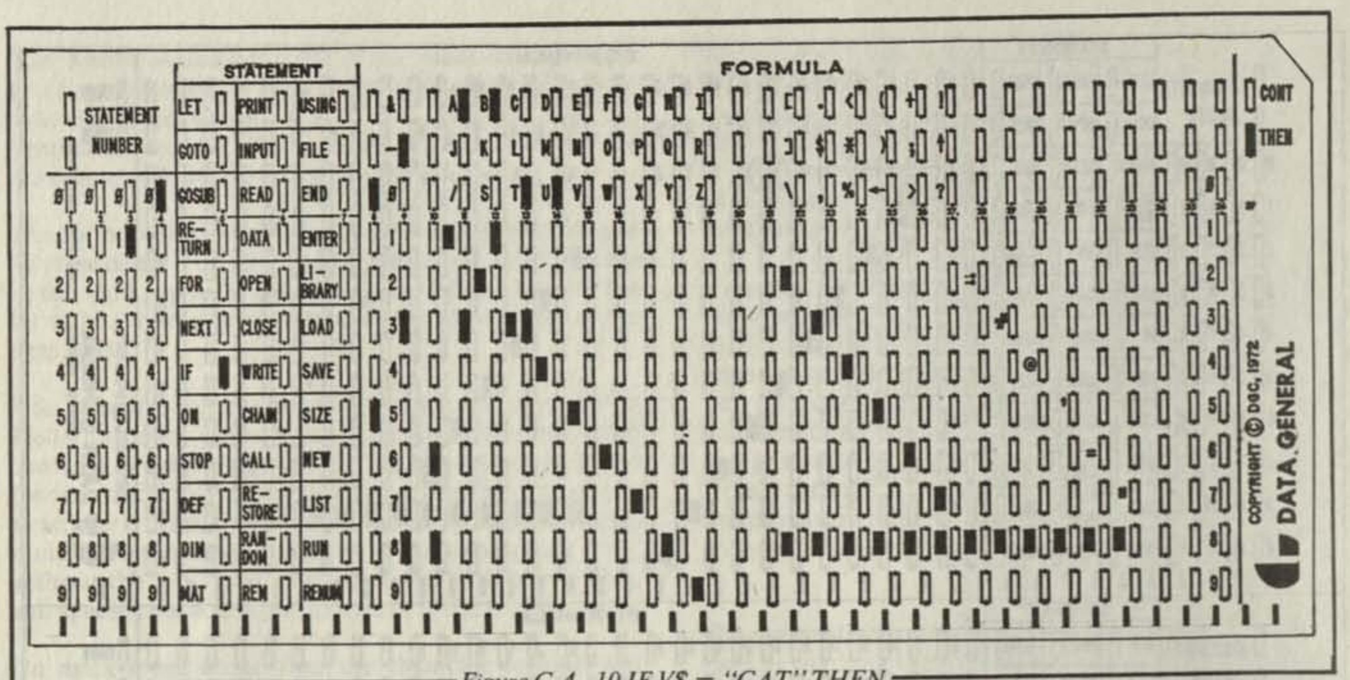


Figure C-4. 10 IF V$ = "CAT" THEN

End of Appendix

Appendix D
Hollerith Character Set

Character	Lines		
0	0	-	-
1	1	-	-
2	2	-	-
3	3	-	-
4	4	-	-
5	5	-	-
6	6	-	-
7	7	-	-
8	8	-	-
9	9	-	-
A	12	1	-
B	12	2	-
C	12	3	-
D	12	4	-
E	12	5	-
F	12	6	-
G	12	7	-
H	12	8	-
I	12	9	-
J	11	1	-
K	11	2	-
L	11	3	-

Character	Lines		
M	11	4	-
N	11	5	-
O	11	6	-
P	11	7	-
Q	11	8	-
R	11	9	-
S	0	2	-
T	0	3	-
U	0	4	-
V	0	5	-
W	0	6	-
X	0	7	-
Y	0	8	-
Z	0	9	-
[	12	2	8
.	12	3	8
<	12	4	8
(	12	5	8
+	12	6	8
!	12	7	8
]	11	2	8
$	11	3	8

Character	Lines		
*	11	4	8
)	11	5	8
;	11	6	8
↑	11	7	8
/	0	1	-
\	0	2	8
, (comma)	0	3	8
%	0	4	8
—	0	5	8
>	0	6	8
?	0	7	8
:	2	8	-
#	3	8	-
@	4	8	-
'	5	8	-
=	6	8	-
"	7	8	-
&	12	-	-
- (minus)	11	-	-

SD-01075

End of Appendix

Appendix E
ASCII Character Sets

ANSI Standard SET (AOS)

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

LEGEND:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character

		10_
0	64 / 7C	@

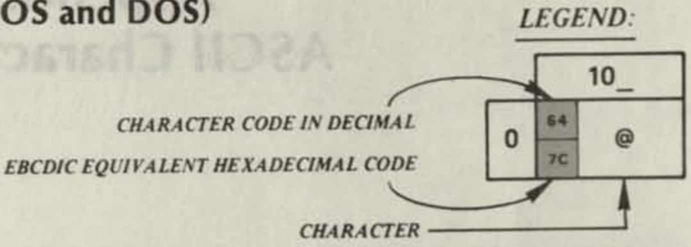
OCTAL	00_			01_			02_			03_			04_			05_			06_			07_		
0	0	00	NUL	8	16	BS (BACK-SPACE)	16	10	DLE ↑P	24	18	CAN ↑X	32	40	SPACE	40	4D	(	48	F0	0	56	F8	8
1	1	01	SOH ↑A	9	05	HT (TAB)	17	11	DC1 ↑Q	25	19	EM ↑Y	33	5A	!	41	5D	)	49	F1	1	57	F9	9
2	2	02	STX ↑B	10	15	NL (NEW LINE)	18	12	DC2 ↑R	26	3F	SUB ↑Z	34	7F	" (QUOTE)	42	5C	*	50	F2	2	58	7A	:
3	3	03	ETX ↑C	11	0B	VT (VERT TAB)	19	13	DC3 ↑S	27	27	ESC (ESCAPE)	35	7B	#	43	4E	+	51	F3	3	59	5E	;
4	4	37	EOT ↑D	12	06	FF (FORM FEED)	20	3C	DC4 ↑T	28	1C	FS ↑\	36	5B	$	44	6B	, (COMMA)	52	F4	4	60	4C	<
5	5	2D	ENQ ↑E	13	0D	RT (RETURN)	21	3D	NAK ↑U	29	1D	GS ↑]	37	6C	%	45	60	-	53	F5	5	61	7E	=
6	6	2E	ACK ↑F	14	0E	SO ↑N	22	32	SYN ↑V	30	1E	RS ↑↑	38	50	&	46	4B	. (PERIOD)	54	F6	6	62	6E	>
7	7	2F	BEL ↑G	15	0F	SI ↑O	23	26	ETB ↑W	31	1F	US ↑←	39	7D	' (APOS)	47	61	/	55	F7	7	63	6F	?

OCTAL	10_			11_			12_			13_			14_			15_			16_			17_		
0	64	7C	@	72	C8	H	80	D7	P	88	E7	X	96	79	` (GRAVE)	104	88	h	112	97	p	120	A7	x
1	65	C1	A	73	C9	I	81	D8	Q	89	E8	Y	97	81	a	105	89	i	113	98	q	121	A8	y
2	66	C2	B	74	D1	J	82	D9	R	90	E9	Z	98	82	b	106	91	j	114	99	r	122	A9	z
3	67	C3	C	75	D2	K	83	E2	S	91	8D	[	99	83	c	107	92	k	115	A2	s	123	C0	{
4	68	C4	D	76	D3	L	84	E3	T	92	E0	\	100	84	d	108	93	l	116	A3	t	124	4F	|
5	69	65	E	77	D4	M	85	E4	U	93	9D	]	101	85	e	109	94	m	117	A4	u	125	D0	}
6	70	C6	F	78	D5	N	86	E5	V	94	5F	↑ or ^	102	86	f	110	95	n	118	A5	v	126	A1	~ (TILDE)
7	71	C7	G	79	D6	O	87	E6	W	95	6D	← or _	103	87	g	111	96	o	119	A6	w	127	07	DEL (RUBOUT)

SD-00217 Character code in octal at top and left of charts.

↑ means CONTROL

Non-ANSI Standard Set (RDOS and DOS)



↑ *means* CONTROL

OCTAL	00_			01_			02_			03_			04_			05_			06_			07_		
0	0	00	NUL	8	16	BS *(BACK-SPACE)*	16	10	↑P	24	18	↑X	32	40	SPACE	40	4D	(	48	F0	Ø	56	F8	8
1	1	01	↑A	9	05	HT *(TAB)*	17	11	↑Q	25	19	↑Y	33	5A	!	41	5D	)	49	F1	1	57	F9	9
2	2	02	↑B	10	15	LINE FEED	18	12	↑R	26	3F	↑Z	34	7F	" *(QUOTE)*	42	5C	*	50	F2	2	58	7A	:
3	3	03	↑C	11	0B	VT *(VERT. TAB)*	19	13	↑S	27	27	ESC *(ESCAPE)*	35	7B	#	43	4E	+	51	F3	3	59	5E	;
4	4	37	↑D	12	06	FF *(FORM FEED)*	20	3C	↑T	28	1C	↑\	36	5B	$	44	6B	, *(COMMA)*	52	F4	4	60	4C	<
5	5	2D	↑E	13	0D	CR *(RETURN)*	21	3D	↑U	29	1D	↑]	37	6C	%	45	60	–	53	F5	5	61	7E	=
6	6	2E	↑F	14	0E	↑N	22	32	↑V	30	1E	↑↑	38	50	&	46	4B	. *(PERIOD)*	54	F6	6	62	6E	>
7	7	2F	BELL ↑G	15	0F	↑O	23	26	↑W	31	1F	↑←	39	7D	' *(APOS)*	47	61	/	55	F7	7	63	6F	?

OCTAL	10_			11_			12_			13_			14_			15_			16_			17_		
0	64	7C	@	72	C8	H	80	D7	P	88	E7	X	96	79	` *(GRAVE)*	104	88	h	112	97	p	120	A7	x
1	65	C1	A	73	C9	I	81	D8	Q	89	E8	Y	97	81	a	105	89	i	113	98	q	121	A8	y
2	66	C2	B	74	D1	J	82	D9	R	90	E9	Z	98	82	b	106	91	j	114	99	r	122	A9	z
3	67	C3	C	75	D2	K	83	E2	S	91	8D	[	99	83	c	107	92	k	115	A2	s	123	C0	{
4	68	C4	D	76	D3	L	84	E3	T	92	E0	\	100	84	d	108	93	l	116	A3	t	124	4F	|
5	69	C5	E	77	D4	M	85	E4	U	93	9D	]	101	85	e	109	94	m	117	A4	u	125	D0	}
6	70	C6	F	78	D5	N	86	E5	V	94	5F	↑ or ∧	102	86	f	110	95	n	118	A5	v	126	A1	~ *(TILDE)*
7	71	C7	G	79	D6	O	87	E6	W	95	6D	← or _	103	87	g	111	96	o	119	A6	w	127	07	DEL *(RUBOUT)*

CHARACTER CODE IN OCTAL AT TOP AND LEFT OF CHARTS.

SD-00476

End of Appendix

Appendix F
Statement, Command and Function Summary

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
ABS(expr) The absolute value of an expression.			●	●	●	●
ACL "filename" *[,UserID","attributes"]...* Prints a report of, or changes, the access control list for a file.	●	●		●		
ATN(expr) The arctangent of an angle. Result expressed in radians.			●	●	●	●
AUDIT *["filename"]* Audits your input and output.	●	●		●		
BYE Sign-off command.	●	●		●	●	●
CALL subr *[,expr]...* Calls an assembly language subroutine.	●	●		●	●	●
CARDS "filename" Enters program in mark sense card format.	●	●			●	●
CHAIN "filename" *[THEN GOTO line no.]* Transfers control to the program named in the statement..	●	●		●	●	●
CHAR $\left[\begin{Bmatrix}\text{"ON"}\\ \text{"OFF"}\\ \text{"characteristics"}\\ \text{"LPP", svar}\\ \text{"CPL", svar}\\ \text{"device"}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{"ON"}\\ \text{"OFF"}\\ \text{"characteristics"}\\ \text{"LPP", svar}\\ \text{"CPL", svar}\\ \text{"device"}\end{Bmatrix}\right]\ldots$ Prints a report of, or changes the current device characteristics.	●	●		●		
CHAR $\left[\begin{Bmatrix}\text{"ON"}\\ \text{"OFF"}\\ \text{"characteristics"}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{"ON"}\\ \text{"OFF"}\\ \text{"characteristics"}\end{Bmatrix}\right]\ldots$ Prints a report of, or changes the current device characteristics.	●	●			●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
CHATR "filename", attributes Changes file attributes.	●	●			●	●
CLI *[command]* Provides access to the CLI without terminating the BASIC process.	●	●		●		
CLOSE *[FILE(file)]* Closes an open file or files.	●	●		●	●	●
CON Continues execution of a STOPped program.		●		●	●	●
COS(expr) The cosine of an angle. Angle expressed in radians.			●	●	●	●
CPU(expr) Returns a value (0 or 1) equal to the status of a CPU console switch.			●		●	●
DATA $\left\{\begin{matrix}\text{val}\\ \text{"str lit"}\end{matrix}\right\}\left[\left\{\begin{matrix}, val\\ , \text{"}str\ lit\text{"}\end{matrix}\right\}\right]\ldots$ Defines data to be used by READ and MAT READ.	●			●	●	●
DEF Used with FNa(d) function to define a user function.	●			●	●	●
DELAY=expr Delays program execution for a specified amount of time.	●	●		●	●	●
DELETE "filename" Deletes a file from your directory.	●	●		●	●	●
DIM $\left\{\begin{matrix}\text{svar (n)}\\ \text{array(m)}\\ \text{array(row,col)}\end{matrix}\right\}\left[\left\{\begin{matrix}, svar(n)\\ , array(m)\\ , array(row,col)\end{matrix}\right\}\right]\ldots$ Specifies the size of string variables and numeric arrays.	●	●		●	●	●
DISK Prints the number of blocks used and number available in the partition in which your directory resides.	●	●		●	●	●
END Stops program execution.	●			●	●	●
ENTER $\left\{\begin{matrix}\text{"filename"}\\ \text{svar}\end{matrix}\right\}$ Merges the program named into the current program.	●	●		●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
ERASE n1, n2 Deletes statements from a program.	●	●		●	●	●
EOF (file) Returns a 1 if an end of file is detected, otherwise, 0.			●	●	●	●
EXP(expr) The value of e to the power of an expression.			●	●	●	●
FILE *["template"]* Prints the names of files in your directory that match the template.	●	●		●		
FILE Prints the filenames in your directory.	●	●			●	●
FNa(d) A user function which is defined in a DEF statement and returns a value.			●	●	●	●
FOR control var = expr1 TO expr2 *[STEP expr3]* Begins a FOR-NEXT loop and defines the number of times the loop is executed.	●			●	●	●
GDIR Verifies the name of your directory.	●	●			●	●
GOSUB line no. Transfers program control to the first statement of a subroutine.	●			●	●	●
GOTO line no. Transfers program execution to a specified line.	●			●	●	●
GPOS FILE (file), var Determines the current filepointer position in an open file.	●	●		●	●	●
HELP "verb" Displays information about each BASIC statement and command.	●	●		●	●	●
IF $\begin{Bmatrix}\text{rel-expr}\\ \text{expr}\end{Bmatrix} \begin{Bmatrix}\textit{[THEN]}\ \text{statement}\\ \text{THEN line no.}\end{Bmatrix}$ Executes a statement based on whether an expression is true or false.	●	●		●	●	●
INPUT $\textit{["str lit",]} \begin{Bmatrix}\text{var}\\ \text{svar}\end{Bmatrix} \left[\begin{Bmatrix}\textit{,var}\\ \textit{,svar}\end{Bmatrix}\right] \ldots \textit{[;]}$ You input data for variables from terminal.	●	●		●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
INPUT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right), \begin{Bmatrix}\text{var}\\ \text{svar}\end{Bmatrix}\left[, \begin{Bmatrix}\textit{var}\\ \textit{svar}\end{Bmatrix}\right]\ldots$ Reads data in ASCII from a sequential-access file.	•	•		•	•	•
INT(expr) The largest integer not greater than the expression.			•	•	•	•
LEN(svar) Returns the number of characters currently assigned to a string variable.			•	•	•	•
$\textit{[LET]}\begin{Bmatrix}\text{svar}\\ \text{var}\end{Bmatrix} = \text{expr}$ Assigns values or solutions to formulas to a variable.	•	•		•	•	•
LIBRARY $\left[\begin{Bmatrix}\textit{"directory","template"}\\ \textit{"directory"}\\ \textit{"template"}\end{Bmatrix}\right]$ Prints the names of files in the BASIC library directory that match the template.	•	•		•		
LIBRARY Prints the filenames in the library directory.	•	•			•	•
LIST $\left[\begin{Bmatrix}\textit{line n1}\\ \begin{Bmatrix}\textit{TO}\\ ,\end{Bmatrix}\textit{line n2}\\ \textit{line n1}\end{Bmatrix}\begin{Bmatrix}\textit{TO}\\ ,\end{Bmatrix}\textit{line n2}\right]$ *["filename"]* Outputs part or all of the current program to the terminal or other ASCII device.		•		•	•	•
LOAD "filename" Loads a previously SAVEd program into the program storage area.		•		•	•	•
LOG(expr) The natural logarithm of an expression.			•	•	•	•
LREAD *["str lit",]* svar *[,svar1]* Reads a string from the terminal which is terminated by either a null, form feed, or carriage return (new-line).	•	•			•	•
LREAD FILE $\left\{\begin{pmatrix}\text{file}\\ \text{file, record}\end{pmatrix}\right\}$, svar *[,svar1]* To read a string from a record in either a random- or sequential-access file which is terminated by either a null, form feed, or carriage return (new line).	•	•		•	•	•

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
LWRITE svar *[,svar1]* Writes a string to your terminal which is delimited by either a null, form feed, or carriage return (new line).	●	●		●	●	●
LWRITE FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file, record}\end{Bmatrix}\right)$, svar *[,svar1]* Writes a string to a record into either a random- or sequential-access file which will be terminated by either a null, form feed, or carriage return (new line).	●	●		●	●	●
MAT mvar1 = mvar2 Assigns the dimensions and values of mvar2 to mvar1.	●	●		●	●	●
MAT mvar1 = mvar2 $\begin{Bmatrix}+\\ -\end{Bmatrix}$ mvar Performs matrix addition or subtraction.	●	●		●	●	●
MAT mvar1 = $\begin{Bmatrix}\text{mvar2}\\ \text{(expr)}\end{Bmatrix}$ * mvar3 Multiplies a matrix by a numeric expression or another matrix.	●	●		●	●	●
MAT mvar = CON *[([row,]col)]* Sets the value of each matrix element to one.	●	●		●	●	●
MAT mvar = IDN *[([row,]col)]* Sets the elements of the major diagonal of a matrix to ones and all other elements to zeros.	●	●		●	●	●
MAT mvar1 = INV (mvar2) Performs matrix inversion.	●	●		●	●	●
MAT mvar1 = TRN (mvar2) Transposes matrix mvar2.	●	●		●	●	●
MAT mvar = ZER (*[row,]* col) Sets the value of each matrix element to zero.	●	●		●	●	●
MAT INPUT *["str lit"]* mvar *[([row,]col)][,mvar[(row,]col)]]* ... Specifies matrixes for which you enter data from the terminal when the statement is executed.	●	●		●	●	●
MAT $\begin{Bmatrix}\text{;}\\ \text{PRINT}\end{Bmatrix}$ mvar $\left[\begin{Bmatrix}\text{,}\\ \text{:}\\ \text{;}\end{Bmatrix} \textit{mvar}\right]$ ... *[;]* Prints the contents of the specified matrixes.	●	●		●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
MAT READ mvar *[([row,]col)][,mvar[([row,]col)]]* ... Reads data into the specified matrixes from the data list defined by DATA statement(s).	•	•		•	•	•
MAT INPUT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar *[,mvar]* ... Reads a matrix data in ASCII from a sequential-access file.	•	•		•	•	•
MAT PRINT FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar $\left[\begin{Bmatrix},\\ ;\\ ,\end{Bmatrix} \textit{mvar} \cdots \begin{Bmatrix};\\ ,\end{Bmatrix}\right]$ Outputs matrix file data to an ASCII device.	•	•		•	•	•
MAT READ FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar *[,mvar]* ... Reads matrix data in binary format from a file.	•	•		•	•	•
MAT TINPUT *[(line no. [,time]),] ["str lit",]* mvar *[([row,] col)] [,mvar [([row,]col)]]* ... Reads values from your terminal and assigns them to the elements of a matrix or list of matrixes, within a prescribed time.	•	•		•	•	•
MAT WRITE FILE $\left(\begin{Bmatrix}\text{file}\\ \text{file,record}\end{Bmatrix}\right)$,mvar *[,mvar]* ... Writes matrix data in binary format to a file.	•	•		•	•	•
MSG $\begin{Bmatrix}\text{pid}\\ \text{"processname"}\\ \text{"console name"}\end{Bmatrix}$, "message" Transmits a messge from your terminal to another programmer or to the system operator.	•	•		•		
MSG userID message Transmits messages to other users or the operator or cancels NOMSG.	•	•			•	•
NEW *["filename"]* Clears your storage area.	•	•		•	•	•

Formats and Descriptions	**S**	**C**	**F**	**AOS**	**RDOS**	**DOS**
NEXT control var The last statement in a FOR-NEXT loop; changes the value of the control variable.	●			●	●	●
ON ERR $\left\{\begin{matrix}\text{THEN line no.}\\ \textit{[THEN]}\text{ statement}\end{matrix}\right\}$ Directs the program to an error handling routine when an error occurs.	●			●	●	●
ON ESC $\left\{\begin{matrix}\text{THEN line no.}\\ \textit{[THEN]}\text{ statement}\end{matrix}\right\}$ Directs the program to a user handling routine when ESCape is pressed.	●			●	●	●
ON expr $\left\{\begin{matrix}\text{GOTO}\\ \text{GOSUB}\end{matrix}\right\}$ line no. *[,line no.]* ... Transfers program control to a line number whose position in the argument list is computed from an expression.	●			●	●	●
OPEN FILE(file,mode),"filename" *[,recordsize [,filesize]]* Opens a file which can then be referred to by other file I/O statements.	●	●		●	●	●
PAGE = expr Sets the right margin of the terminal.	●	●		●	●	●
POS $\left(\left\{\begin{matrix}\text{svar1}\\ \text{"str lit 1"}\end{matrix}\right\}, \left\{\begin{matrix}\text{svar2}\\ \text{"str lit 2"}\end{matrix}\right\}\text{,expr}\right)$ Locates the position of a substring in a string.			●	●	●	●
$\left\{\begin{matrix};\\ \text{PRINT}\end{matrix}\right\}\left[\left\{\begin{matrix}\textit{expr}\\ \textit{"str lit"}\\ \textit{svar}\end{matrix}\right\}\left[\left\{\begin{matrix},\\ :\\ ;\end{matrix}\right\}\left\{\begin{matrix}\textit{expr}\\ \textit{"str lit"}\\ \textit{svar}\end{matrix}\right\}\right]\ldots\right]\left[\left\{\begin{matrix},\\ :\\ ;\end{matrix}\right\}\right]$ Prints specified data.	●	●		●	●	●
PRINT FILE $\left(\left\{\begin{matrix}\text{file}\\ \text{file,record}\end{matrix}\right\}\right), \left\{\begin{matrix}\text{expr}\\ \text{var}\\ \text{svar}\\ \text{"str lit"}\end{matrix}\right\}\left[\left\{\begin{matrix},\\ :\\ ;\end{matrix}\right\}\left\{\begin{matrix}\textit{expr}\\ \textit{var}\\ \textit{svar}\\ \textit{"str lit"}\end{matrix}\right\}\right]\ldots\left[\left\{\begin{matrix},\\ :\\ ;\end{matrix}\right\}\right]$ Outputs data to an ASCII device.	●	●		●	●	●
PRINT FILE $\left(\left\{\begin{matrix}\text{file}\\ \text{file,record}\end{matrix}\right\}\right)$, USING format, expr *[,expr]* ... Formats output to files.	●	●		●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
PRINT USING format, expr $\left\{\begin{matrix} ;expr \\ ,expr \end{matrix}\right\}$ ... Formats printed output.	●	●		●	●	●
PUNCH $\left[\left\{\begin{matrix} line\ n1 \\ \left\{\begin{matrix} TO \\ , \end{matrix}\right\} line\ n2 \\ line\ n1 \left\{\begin{matrix} TO \\ , \end{matrix}\right\} line\ n2 \end{matrix}\right\}\right]$ Outputs part or all of the current program to the terminal punch.		●			●	
RANDOMIZE Resets the random number generator.	●	●		●	●	●
READ $\left\{\begin{matrix} \text{var} \\ \text{svar} \end{matrix}\right\} \left[\left\{\begin{matrix} ,var \\ ,svar \end{matrix}\right\}\right]$ ... Reads data from DATA statements.	●	●		●	●	●
READ FILE $\left(\left\{\begin{matrix} \text{file} \\ \text{file,record} \end{matrix}\right\}\right), \left\{\begin{matrix} \text{var} \\ \text{svar} \end{matrix}\right\} \left[, \left\{\begin{matrix} var \\ svar \end{matrix}\right\}\right]$ Reads data in binary format from a file.	●	●		●	●	●
REM *[message]* Inserts explanatory comments within a program.	●			●	●	●
RENAME "oldfilename", "newfilename" Renames files.	●	●		●	●	●
RENUMBER $\left[\left\{\begin{matrix} line\ n1 \\ STEP\ n2 \\ line\ n1\ STEP\ n2 \end{matrix}\right\}\right]$ Renumbers statements in the current program.		●		●	●	●
RESET *[FILE (file)]* Positions the file pointer to the beginning of a file.	●	●		●	●	●
RESTORE *[line no.]* Moves the data element pointer to the beginning of a data list or DATA statement line.	●	●		●	●	●
RETRY Repeats the statement which caused an error.	●			●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
RETURN Last statement of a subroutine; returns program control to statement following last GOSUB statement executed.	●			●	●	●
RND(expr) Random number n, such that 0 <= n < 1.			●	●	●	●
RUN $\left[\begin{Bmatrix} \textit{line no.} \\ \textit{"filename"} \end{Bmatrix}\right]$ Executes the current program or another program named by filename.		●		●	●	●
SAVE "filename" Writes the current program into your directory or to a device in binary format.	●	●		●	●	●
SGN(expr) The algebraic sign of an expression.			●	●	●	●
SIN(expr) The sine of an angle. Angle expressed in radians.			●	●	●	●
SIZE Provides program and data storage usage information.	●	●		●	●	●
SPOS FILE (file), expr Moves the file pointer to the byte position specified by expr.	●	●		●	●	●
SQR(expr) The square root of an expression.			●	●	●	●
STOP Stops program execution.	●			●	●	●
STR$(expr) Converts a numeric expression to its string representation.			●	●	●	●
SYS(0) The time of day (seconds past midnight).			●	●	●	●
SYS(1) The day of the month.			●	●	●	●
SYS(2) The month of the year.			●	●	●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
SYS(3) The year.			●	●	●	●
SYS(4) The terminal port number (-1 for operator's console).			●	●	●	●
SYS(5) CPU time used in seconds.			●	●	●	●
SYS(6) The number of file I/O statements executed.			●	●	●	●
SYS(7) The error code of the last runtime error.			●	●	●	●
SYS(8) The number of the file most recently opened.			●	●	●	●
SYS(9) Page size.			●	●	●	●
SYS(10) Tab size.			●	●	●	●
SYS(11) Hour of the day.			●	●	●	●
SYS(12) Minutes past last hour.			●	●	●	●
SYS(13) Seconds past last minute.			●	●	●	●
SYS(14) Seconds remaining on timed input.			●	●	●	●
SYS(15) The constant PI (3.14159).			●	●	●	●
SYS(16) The constant e (2.71828).			●	●	●	●
SYS(17) 1/10 second clock.			●		●	●

Formats and Descriptions	S	C	F	AOS	RDOS	DOS
SYS(18) Total number of BASIC I/O calls.			●	●	●	●
TAB(expr) Function used with PRINT for tabulating to a column.			●	●	●	●
TAB=expr Sets the zone spacing for PRINT statements.	●	●		●	●	●
TAN(expr) The tangent of an angle. Angle expressed in radians.			●	●	●	●
TIME=expr Establishes the time limit for timed input operation.	●	●		●	●	●
TINPUT $[(\mathit{line\ no.}[,\mathit{time}]),]\ [\text{"}\mathit{str\ lit}\text{"},] \left\{\begin{matrix}\text{var}\\ \text{svar}\end{matrix}\right\} \left[\left\{\begin{matrix},\mathit{var}\\ ,\mathit{svar}\end{matrix}\right\}\right]\ldots\ [;]$ Sets a time limit for programmer response.	●	●		●	●	●
VAL $\left(\left\{\begin{matrix}\text{svar}\\ \text{"str lit"}\end{matrix}\right\}\right)$ Returns decimal representation of a string.			●	●	●	●
WHATS "filename" Prints attributes and other information relating to a file.	●	●		●	●	●
WHO $\left[\left\{\begin{matrix}\mathit{processID}\\ \text{"}\mathit{processname}\text{"}\end{matrix}\right\}\right]$ Identifies others on the system or provides your own identification.	●	●		●		
WRITE FILE $\left\{\begin{matrix}\text{file}\\ \text{file,record}\end{matrix}\right\}, \left\{\begin{matrix}\text{expr}\\ \text{var}\\ \text{svar}\\ \text{"str lit"}\end{matrix}\right\} \left[,\left\{\begin{matrix}\mathit{expr}\\ \mathit{var}\\ \mathit{svar}\\ \text{"}\mathit{str\ lit}\text{"}\end{matrix}\right\}\right]\ldots$ Writes data in binary format to a file.	●	●		●	●	●

End of Appendix

Index

Within this index, the legend "f" after a page number means "and the following page" (or "pages"). In addition, primary page references for each topic are listed first. Commands, calls, and acronyms are in uppercase letters (e.g., CREATE); all others are lowercase.

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title ________________ No. ________

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- ☐ EDP Manager
- ☐ Senior System Analyst
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ Other ________

What programming language(s) do you use? ________

How Do You Use This Manual?

(List in order: 1 = Primary use)

____ Introduction to the product
____ Reference
____ Tutorial Text
____ Operating Guide
____ ________

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name ________ Title ________ Company ________

Address ________ Date ________

SD-00742

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

SD-00742A STAPLE