

# **2000E: A GUIDE TO TIME-SHARED BASIC**

02000-90048

August 1972



## PREFACE

The Time-shared BASIC system (TSB) has provided a major breakthrough by reducing the cost of using a computer. Now, for the first time, it is practical for the programmer to use his time sharing terminal to teach himself more about the BASIC language. Accordingly, this publication is designed to meet two requirements:

1. To serve as a clear and concise reference text for Time-shared BASIC
2. To serve as an instructional aid to the TSB user.

All example programs may be used as practice exercises (as well as for reference). They were chosen for maximum teaching value, and include pertinent remarks. Beginners are encouraged to try the examples "on-line."

The syntax requirements of BASIC have been "translated" into English from the traditional Backus Normal Form. Each element of a statement is underlined separately.

This text is divided into learning units. Each page presents a separate item or feature, and sections are arranged in a coherent instructional sequence. All items are presented in a standard, consistent format.

# CONVENTIONS USED IN THIS TEXT

<u>SAMPLE</u>	<u>EXPLANATION</u>
PLEASE LOG IN	All capitals in examples indicates computer-output information...
2Ø PRINT X,Y LIST	or a statement or command typed by the programmer.
This section...	Mixed upper and lower case is used for regular text.
<u>line number</u> PRINT X,Y	Lower case italics indicates a general form, derived from BASIC syntax requirements (Sect. IX).  Underlining indicates an essential part of a general form; each underlined item is a separate, essential element.
<u>return</u> <u>linefeed</u> <u>esc</u> <u>ctrl</u> <u>alt-mode</u> <u>break</u>	Represents the terminal keys: Return, Linefeed, Escape, Control, Alt-Mode, and Break.
<i>Note: Both X and...</i>	Mixed upper and lower case italics is used for notes.
LISTING A PROGRAM	Oversized type is used for page headings.
0	The letter "0"
Ø	Zeroes are slashed.

# PAGE FORMAT

The reference page format is as uniform as possible. This sample shows how positioning and typeface relate to content. Black frames are used on reference pages.

EXAMPLES: \_\_\_\_\_ Several sample  
          \_\_\_\_\_ statements or commands  
          \_\_\_\_\_

GENERAL FORM: \_\_\_\_\_  
                  (Each essential element underlined.)

## PURPOSE

A clear and concise explanation of the purpose or function.

## COMMENTS

A series of several items containing:  
    Pertinent information  
    Additional explanation or examples  
    Helpful hints.

Reference to other sections or subsections related to the contents of this page.

"Continued on the next page" if the explanation fills more than one page.

Page No. \_\_\_\_

# HOW TO USE THIS BOOK

If your purpose is:

Read:

Quickly acquiring a minimum working knowledge of Time-shared BASIC:

Sections I and II.

Acquiring a good working knowledge of Time-shared BASIC:

Sections I, II, III, IV, V, VI, in that order.

Learning the complete Time-shared BASIC system:

The entire book, in sequence.

Reference only:

1. Contents
2. The index

# CONTENTS

iii	PREFACE
iv	CONVENTIONS USED IN THIS TEXT
v	PAGE FORMAT
vi	HOW TO USE THIS BOOK
1-1	SECTION I AN INTRODUCTION TO TIME SHARED BASIC
1-1	WHAT IS TIME SHARING?
1-2	COMMUNICATING WITH A COMPUTER
1-3	EXAMPLES OF BASIC STATEMENTS
1-4	STATEMENT NUMBERS
1-5	INSTRUCTIONS (STATEMENT TYPES)
1-6	OPERANDS
1-7	A PROGRAM
1-8	THE FORMAT OF STATEMENTS
1-10	BEFORE GOING ON-LINE
1-11	PRESS RETURN AFTER EACH STATEMENT
1-12	BACKSPACE
1-13	DELETING OR CHANGING A STATEMENT
1-14	LISTING A PROGRAM
1-16	CONNECTION TO THE COMPUTER
1-17	CHECKING THE CONNECTION
1-17	Your ID Code and Password
1-18	Control Characters
1-19	SAMPLE LOG IN AND LOG OUT
1-20	MISTAKES DURING LOG IN
1-21	ENTERING THE SAMPLE PROGRAM
1-22	HOW TO OBTAIN A DIAGNOSTIC MESSAGE
1-23	RUNNING THE SAMPLE PROGRAM
1-24	STOPPING A PROGRAM: THE <u>break</u> KEY
1-25	HOW THE PROGRAM WORKS

# CONTENTS CONTINUED

2-1	SECTION II THE ESSENTIALS OF BASIC
2-1	HOW TO READ THIS SECTION
2-2	TERM: NUMBER
2-2	TERM: "E" NOTATION
2-3	TERM: SIMPLE VARIABLE
2-4	TERM: ARITHMETIC EVALUATION
2-5	THE ASSIGNMENT OPERATOR
2-6	ARITHMETIC OPERATORS
2-7	RELATIONAL OPERATORS
2-8	MIN AND MAX OPERATORS
2-9	THE AND OPERATOR
2-10	THE OR OPERATOR
2-11	THE NOT OPERATOR
2-12	ORDER OF PRECEDENCE OF EXECUTION
2-13	STATEMENTS
2-14	THE ASSIGNMENT STATEMENT
2-15	REM
2-16	GO TO AND MULTIBRANCH GO TO
2-17	IF...THEN
2-18	FOR...NEXT
2-20	NESTING FOR...NEXT LOOPS
2-21	READ, DATA AND RESTORE
2-24	INPUT
2-26	PRINT
2-28	END AND STOP
2-29	Sample Program
2-32	Running the Sample Program
2-33	COMMANDS
2-34	HELLO
2-35	BYE
2-36	ECHO-
2-37	RUN
2-38	LIST
2-39	SCRATCH

# CONTENTS CONTINUED

2-40	RENUMBER
2-41	BREAK
2-42	PUNCH
2-43	XPUNCH
2-44	TAPE
2-45	KEY
2-46	TIME
2-47	DISC
2-48	MESSAGE
3-1	SECTION III ADVANCED BASIC
3-2	ROUTINE
3-3	ARRAY (OR MATRIX)
3-4	STRING
3-4	FUNCTION
3-5	WORD
3-5	RECORD
3-6	STORING AND DELETING PROGRAMS
3-7	LENGTH
3-8	NAME
3-9	SAVE-
3-10	GET- AND GET-\$
3-11	KILL-
3-12	APPEND-
3-13	DELETE-
3-14	LIBRARY
3-15	CATALOG
3-16	SUBROUTINES AND FUNCTIONS
3-17	GOSUB...RETURN
3-18	MULTIBRANCH GOSUB
3-19	NESTING GOSUB'S
3-20	FOR...NEXT WITH STEP
3-21	DEF FN
3-22	GENERAL MATHEMATICAL FUNCTIONS
3-23	TRIGONOMETRIC FUNCTIONS

## CONTENTS CONTINUED

3-24	THE TAB AND SGN FUNCTIONS
3-25	THE TYP FUNCTION
3-26	THE LEN FUNCTION
3-27	THE TIM FUNCTION
3-28	CHAIN
3-29	COM
4-1	SECTION IV FILES
4-2	TERM: FILE
4-3	SERIAL FILE ACCESS
4-5	OPEN-
4-7	KILL-
4-8	FILES
4-10	SERIAL FILE PRINT
4-12	SERIAL FILE READ
4-14	RESETTING
4-15	LISTING CONTENTS OF A FILE
4-16	THE TYP FUNCTION
4-17	TERM: END-OF-FILE
4-18	IF END#...THEN
4-19	PRINT#...END
4-20	MODIFYING A SERIAL FILE
4-21	EXAMPLE OF SERIAL FILE MODIFICATION
4-23	STRUCTURE OF SERIAL FILES
4-26	TERM: RECORD
4-27	STORAGE REQUIREMENTS
4-28	MOVING THE POINTER
4-29	SAMPLE USE OF READ#M,N
4-30	SUBDIVIDING SERIAL FILES
4-31	USING THE TYP FUNCTION WITH RECORDS
4-32	SAMPLE OF READ#M,N AND TYP(-M)
4-33	HOW TO COPY A FILE
4-34	TERM: RANDOM FILE ACCESS
4-35	SAMPLE OF RANDOM FILE ACCESS

# CONTENTS CONTINUED

4-36	PRINTING A RECORD
4-38	READING A RECORD
4-39	MODIFYING CONTENTS OF A RECORD
4-40	ERASING A RECORD
4-42	UPDATING A RECORD
4-43	AN ALPHABETICALLY ORGANIZED FILE

## 5-1 SECTION V MATRICES

5-1	MATRIX (ARRAY)
5-2	DIM
5-3	MAT...ZER
5-4	MAT...CON
5-5	INPUT
5-6	MAT INPUT
5-7	PRINT MATRICES
5-8	MAT PRINT
5-9	READ
5-10	MAT READ
5-11	MATRIX ADDITION
5-12	MATRIX SUBTRACTION
5-13	MATRIX MULTIPLICATION
5-14	SCALAR MULTIPLICATION
5-15	COPYING A MATRIX
5-16	IDENTITY MATRIX
5-17	MATRIX TRANSPOSITION
5-18	MATRIX INVERSION
5-19	MAT PRINT#
5-20	MAT READ#

## 6-1 SECTION VI STRINGS

6-2	STRING
6-3	STRING VARIABLE

# CONTENTS CONTINUED

- 6-4 SUBSTRING
- 6-6 STRINGS AND SUBSTRINGS
- 6-8 THE STRING DIM STATEMENT
- 6-9 THE STRING ASSIGNMENT STATEMENT
- 6-10 THE STRING INPUT STATEMENT
- 6-11 PRINTING STRINGS
- 6-12 READING STRINGS
- 6-13 STRING IF
- 6-14 THE LEN FUNCTION
- 6-15 STRING IN DATA STATEMENTS
- 6-16 PRINTING STRINGS ON FILES
- 6-17 READING STRINGS FROM FILES
  
- 7-1 SECTION VII  
LOGICAL OPERATIONS
  
- 7-1 LOGICAL VALUES AND NUMERIC VALUES
- 7-2 RELATIONAL OPERATORS
- 7-4 BOOLEAN OPERATORS
- 7-5 SOME EXAMPLES
  
- 8-1 SECTION VIII  
FOR THE PROFESSIONAL
  
- 8-2 SYNTAX REQUIREMENTS OF TSB
- 8-8 STRING EVALUATION BY ASCII CODES
- 8-9 MEMORY ALLOCATION BY A USER
  
- A-1 APPENDIX A  
HOW TO PREPARE A PAPER TAPE OFF-LINE
  
- B-1 APPENDIX B  
THE X-ON, X-OFF FEATURE

# CONTENTS CONTINUED

C-1 APPENDIX C  
DIAGNOSTIC MESSAGES

INDEX

# SECTION I

## AN INTRODUCTION TO TIME SHARED BASIC

This section is for novices and programmers in need of a "brush-up" on mechanical skills. The information presented here is arranged in a tutorial sequence. It is assumed that the reader has access to a Time Shared BASIC terminal, and will use some or all of the examples as practice exercises, depending on his own personal requirements.

If you are familiar with the following procedures, skip this section, and begin at Section II:

- Log in and log out
- Correcting mistakes and changing lines
- Obtaining a diagnostic message
- Running and terminating a program.

---

### WHAT IS TIME SHARING?

Time sharing is a method of computer programming which enables many persons (users) to have access to a single computer simultaneously.

The computer processes the requests of the users so rapidly that it seems to each individual that he is the only one using the machine.

Even if every user required large amounts of computer time, the longest delay possible for any one user is a few seconds.

# COMMUNICATING WITH A COMPUTER

## THE BASIC LANGUAGE

There are many types of languages. English is a natural language used to communicate with people. To communicate with the computer we use a formal language, that is, a combination of simple English and algebra.

BASIC is a formal language used to communicate with the computer during time-sharing.

Like natural languages BASIC has grammatical rules, but they are much simpler. For example, this series of BASIC statements (which calculates the average of five numbers given by you, the user) shows the fundamental rules:

```
10 INPUT A,B,C,D,E
20 LET S = (A+B+C+D+E)/5
30 PRINT S
40 GO TO 10
50 END
```

The frames on the following pages show how to interpret these rules. Notice how the statements are written. What they do is explained later.

# EXAMPLES OF BASIC STATEMENTS

This is a BASIC statement:

```
10 INPUT A,B,C,D,E
```

## COMMENTS

A statement contains a maximum of 72 characters (one teletypewriter line).

A statement may also be called a line.

# STATEMENT NUMBERS

Each BASIC statement begins with a statement number  
(in this example, 20):

```
20 LET S=(A+B+C+D+E)/5
```

## COMMENTS

The number is called a statement number or a line number.

The statement number is chosen by you, the programmer. It may be any integer from 1 to 9999 inclusive.

Each statement has a unique statement number. The computer uses the numbers to keep the statements in order.

Statements may be entered in any order; they are usually numbered by fives or tens so that additional statements can be easily inserted. The computer keeps them in numerical order no matter how they are entered. For example, statements are input in the sequence 30,10,20; the computer arranges them in the order: 10,20,30.

# INSTRUCTIONS (STATEMENT TYPES)

The statement then gives an instruction to the computer (in this example, PRINT):

```
30 PRINT S
```

## COMMENTS

Instructions are sometimes called statement types because they identify a type of statement. For example, the statement above is a "print" statement.

# OPERANDS

If the instruction requires further details, operands (numeric details) are supplied (in this example, 10; on the previous page, "S"):

40 GO TO 10

## COMMENTS

The operands specify what the instruction acts upon; for example, what is PRINTed, or where to GO.

# A PROGRAM

The sequence of BASIC statements given on the previous pages is called a program.

The last statement in a program, as shown here, is and END statement.

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 GO TO 10
50 END
```

## COMMENTS

The last (highest numbered) statement in a program must be an END statement.

The END statement informs the computer that the program is finished.

# THE FORMAT OF STATEMENTS

BASIC is a "free format" language--the computer ignores extra blank spaces in a statement. For example, these three statements are equivalent:

```
3Ø PRINT S
3Ø PRINT  S
3ØPRINTS
```

## COMMENTS

When possible, leave a space between words and numbers in a statement. This makes a program easier to read.



(Spot check)

Be sure you are familiar with these terms before continuing:

statement

instruction (statement type)

statement type

statement number (line number)

operand

program

All of these terms are defined in the context of this section.

# BEFORE GOING ON-LINE

The following pages explain the mechanics of entering, correcting, and checking statements.

Since you will probably have to make several corrections in your first attempts to use the computer, these features should be learned before beginning.

## PRESS RETURN AFTER EACH STATEMENT

The return key must be pressed after each statement.

Examples:

```
10 INPUT A,B,C,D,E return  
20 LET S=(A+B+C+D+E)/5 return  
30 PRINT S return  
40 GO TO 10 return  
50 END return
```

### COMMENTS

Pressing return informs the computer that the statement is complete. The computer then checks the statement for mistakes. (The checking process is explained later.)

# BACKSPACE

The reverse arrow (←) key acts as a backspace, deleting the immediately preceding character.

Typing:	20 LR←ET S=10 <u>return</u>
is equivalent to typing:	20 LET S=10 <u>return</u>
And typing:	30 LET← ← ← PRINT S <u>return</u>
is equivalent to typing:	30 PRINT S <u>return</u>

## COMMENTS

The ← character is a "shift" 0 on most terminals.

## DELETING OR CHANGING A STATEMENT

To delete the statement being typed, press and hold down the ctrl key, strike the x key, and release the ctrl key. (From this point on, this operation is indicated by the characters x<sup>c</sup>.) This causes a \ to be printed, and deletes the entire line being typed.

To delete a previously typed statement, type the statement number followed by a return.

To change a previously typed statement, retype it with the desired changes. The new statement replaces the old one.

Pressing the x<sup>c</sup> keys deletes

the statement being typed: 20 LET S = x<sup>c</sup>

*NOTE: The computer responds with a \ when x<sup>c</sup> is typed, like this:*

20 LET S = \

To delete statement 5 in the

sequence: 5 LET S = 0  
10 INPUT A,B,C,D,E,  
20 LET S = (A+B+C+D+E)/5

*NOTE: \ and / are different, and have very different functions.*

type: 5 return

Or, to change statement 5 in

the above sequence, type: 5 LET S = 5 return

The old statement is re-

placed by the new one.

Typing an x<sup>c</sup>

before a return prevents replacement of a previously typed statement.

For example, typing: 5 LET x<sup>c</sup>

or: 5 x<sup>c</sup>

has no effect on the original statement 5.

## LISTING A PROGRAM

After you have made several corrections you may wish to inspect the entire program. Typing LIST return produces a listing of all lines accepted by the computer.

*NOTE: The program has already been entered.*

The computer skips three lines, separating the listing from previously printed information.

linefeed indicates that the listing is complete.

LIST return

linefeed

linefeed

linefeed

1Ø INPUT A,B,C,D,E

2Ø LET S = (A+B+C+D+E)/5

3Ø PRINT S

4Ø GO TO 1Ø

5Ø END

linefeed

The LIST command followed by a dash and statement number causes the listing to begin at the statement specified.

A list of the same sample program produces these lines:

LIST-3Ø return

linefeed

linefeed

linefeed

3Ø PRINT S

4Ø GO TO 1Ø

5Ø END

linefeed



1. Be sure you understand the use of these features work before using the computer:

- return to end statements
- How to backspace
- How to delete a statement
- How to change a statement
- How to list statements

The following pages explain how to make the connection with the computer and log-in.

# CONNECTION TO THE COMPUTER

To enter a program into the computer, first make a connection between the teleprinter and the computer. There are several ways of doing this, depending on the terminal equipment used. The input-output device, such as teleprinter or optical mark reader, on your end of the line is called terminal equipment. Not all users have the same type of equipment.

IF YOUR TERMINAL EQUIPMENT IS A TELEPRINTER WITH

## ACOUSTIC COUPLER AND TELEPHONE:

1. Turn teleprinter control knob to LINE.
2. Turn on coupler power.
3. If coupler has a duplex switch, set to FULL or FULL/UP.
4. If coupler has a line switch set it to ON-LINE.
5. Call the computer number.
6. When the computer answers with a high pitched tone, place the handset in the coupler (Be sure to check that the handset is inserted in the correct position; the connection will not be made if it is reversed. (The correct position should be marked on the coupler.)

## HALF DUPLEX COUPLER AND TELEPHONE

1. Follow instructions 1,2,4,5,6 given above.
2. Log in. (See Log In and Log Out in this section.)
3. Type ECHO-OFF return

## DATA SET:

1. Turn teleprinter control knob to line.
2. Press TALK button on the Data Set.
3. Call the computer number.
4. When the computer answers with a high pitched tone, press the DATA button until the DATA light is on, and replace the handset.

## DIRECT CONNECTION TO THE COMPUTER:

Turn the teleprinter control knob to the LINE position.

# CHECKING THE CONNECTION

The computer does not respond when the connection is established. If you wish to make sure that the connection has been made, type x<sup>c</sup>. When the connection is made, the computer responds with a "\". You can also verify connection by typing in any letter or digit; if the connection is made, the message PLEASE LOG IN appears.

EXAMPLE:

3 return

The computer then responds with the message:

PLEASE LOG IN return linefeed

*NOTE: linefeed causes the teleprinter to advance to the next line.  
return causes the teleprinter typeface to return to the first print position.*

This step is optional

---

## YOUR IDCODE, PASSWORD AND TERMINAL SUBTYPE

You need your identification code, password, and terminal subtype to log in. The ID code and password are assigned by the system operator. The terminal subtype is obtained from the table below. The ID code is a single letter followed by a three digit number. The password consists of one to six regular or control characters. The terminal subtype code tells the system what kind of terminal is being used. It is an integer from 0 to 5. The following table indicates the meaning of each.

<u>Terminal Subtype Code</u>	<u>Terminal</u>
0	HP 2600A, HP 2749A, ASR-33, ASR-35.
1	Execuport 300
2	ASR-37
3	Terminet-300
4	Memorex 1240
5	Univac DCT 500

The message ILLEGAL FORMAT is printed if the type code is less than 0, or greater than 5. If terminal type code is omitted, 0 is assumed by default.

# CONTROL CHARACTERS

Control characters are non-printing. They are represented with a superscript "C" to indicate that they are control characters; an example is x<sup>C</sup>. By using these non-printing characters, you may keep your password a secret. For example, on the teleprinter the password SE<sup>C</sup>C<sup>C</sup>R<sup>C</sup>E<sup>C</sup>T prints as:

ST

Control letters are input by pressing the letter and ctrl keys simultaneously.

# SAMPLE LOG IN AND LOG OUT

H200 is used as a sample identification code.

User H200 for example, logs in on an ASR-33 by typing:

*HELLO- is a command, not a statement. Commands are orders to the computer which are acted upon (executed) immediately. Unlike statements, commands do not have line numbers.*

The computer acknowledges that the user has correctly logged in, by outputting three linefeeds:

If the operator has put a message into the system for users it is printed when the user logs in:

*NOTE: This message can be terminated by hitting break.*

If there is no message, the computer responds with three linefeeds, then READY, indicating it is awaiting input.

To LOG OUT, type:

The elapsed time since log in is then printed.

HELLO-H200,password return  
or

HEL-H200,password, 0 return

linefeed

linefeed

linefeed

-----  
MESSAGE TO USERS FROM OPERATOR

-----  
linefeed

linefeed

linefeed

READY

linefeed

BYE return

001 MINUTES OF TERMINAL TIME

# MISTAKES DURING LOG IN

If you make a mistake while logging in, the computer responds with a message informing you that something is wrong. For example, if user H200 forgets the hyphen while entering the HELLO command or types an incorrect subtype:

HELLO H200,password return

or

HEL-H200,password, 6 return

the computer responds with the message:

ILLEGAL FORMAT return linefeed

and the user then enters the command in the correct form.

---

If user H200 enters his password incorrectly:

HELLO-H200,password return

the response is:

ILLEGAL ACCESS return linefeed

and the user tries again.

*NOTE: The messages ILLEGAL ACCESS and ILLEGAL FORMAT indicate that some or all of the input is not acceptable (not legal) to the Time Shared BASIC system.*

# ENTERING THE SAMPLE PROGRAM

The frame below shows how to enter a program. If you are not sure how the computer responds when a line is entered, use it as a practice exercise.

*NOTE: Connection to the computer is made.*

Log in: HELLO-H200, password return  
OPERATOR'S MESSAGE TO USER

*or*

READY return linefeed

*NOTE: The computer responds with a linefeed after each line is entered. This indicates that the line has been checked and accepted as a legal BASIC statement. It informs the user that the computer is waiting for further input.*

10 INPUT A,B,C,D,E return

linefeed

20 LET S = (A+B+C+D+E)/5 return

linefeed

30 PRINT S return

linefeed

40 GO TO 10 return

linefeed

50 END return

linefeed

Now the program is ready to run.

# HOW TO OBTAIN A DIAGNOSTIC MESSAGE

If you make a mistake while entering a program, the computer responds with an ERROR message. This indicates that the previous line has not been accepted. There are two possible responses to the ERROR message. The frame below shows how to obtain a diagnostic for the probable cause of the error and how to avoid printing the diagnostic if you recognize the mistake.

If the user types:

```
30 PRINT S return
```

*NOTE: PRINT has been misspelled.*

The computer responds:

```
ERROR
```

The user then types in a colon (or any other character) followed by a *return*. This causes the diagnostic to be printed on the same line. The resulting output looks like this:

```
ERROR: return
```

```
ERROR: NO STATEMENT TYPE FOUND
```

*NOTE: PRIMT has not been recognized as a legal statement type, and the line was not accepted.*

To correct the statement, retype it in the proper form:

```
30 PRINT S return
```

---

If you know the cause of the ERROR message and do not wish to see the diagnostic, type a return after the ERROR message is output, then retype the line:

```
30 PRINT S return
```

```
ERROR return
```

```
30 PRINT S
```

Appendix C contains a list of TSB diagnostic messages and probable causes.

# RUNNING THE SAMPLE PROGRAM

This frame shows what happens when the sample program is run. The program does not begin execution (does not run) until the command RUN followed by a return is input.

*NOTE: The program (averaging 5 numbers) has been entered.*

The computer responds with four linefeed's indicating that the command is being executed.

RUN return linefeed  
linefeed  
linefeed  
linefeed

The question mark indicates that input is expected. The five numbers being averaged should be typed in, SEPARATED BY COMMAS, and followed by a return.

? 95.6,87.3,80.5,90,82.8 return

The answer is printed:

87.24 return linefeed

*NOTE: This program continues executing indefinitely, unless terminated by the user. To stop the program, type a C<sup>C</sup> return (control "C") when more input is requested:*

?-12.5,-50.6,-32,45.6,60 return

2.1 return linefeed

? C<sup>C</sup> return

The program is finished:

DONE

Log off:

BYE return

Time used is printed:

003 MINUTES OF TERMINAL TIME

# STOPPING A PROGRAM: THE break KEY

When the commands RUN or LIST are typed, TSB "takes over" the user's terminal until the program or listing is complete.

To terminate a program or listing, press, then release, the break key:

break

When a program is running or being listed, TSB responds with the message:  
after break is pressed.

STOP

Remember that:

and not break is used to terminate input loops (when the computer is expecting a number to be typed in).

C<sup>c</sup> return

## COMMENTS

break must be held down for at least 1/10 second.

# HOW THE PROGRAM WORKS

Line 10 tells the computer that five numbers will be input, and that they should be given the labels A,B,C,D,E in sequence. The first number input is labeled "A" by the computer, the second "B", etc. A,B,C,D, and E are called variables.

```
10 INPUT A,B,C,D,E
```

After line 10 is executed, the variables and their assigned values, typed in by the user, are stored. For example, using the values entered by the user in the previous example, this information is stored: A = -12.5; B = -50.6; C = -32; D = 45.6; E = 60

Line 20 declares that a variable called S exists, and is assigned the value of the sum of the variables A,B,C,D,E divided by 5:

```
20 LET S = (A+B+C+D+E)/5
```

Line 30 instructs the computer to output the value of S to user's terminal:

```
30 PRINT S
```

*NOTE: If the PRINT statement were not given, the value of S would be calculated and stored, but not printed. The computer must be given explicit instruction for each operation to be performed.*

Line 40 tells the computer to go to line 10 and execute whatever instruction is there:

```
40 GO TO 10
```

*NOTE: A "loop" is formed by lines 10 to 40. The sequence of statements in this loop execute until the user breaks the loop. This particular kind of loop is called an input loop (because the user must repeatedly input data). INPUTTING A C WHEN INPUT IS REQUESTED BY A "?" IS THE ONLY WAY TO BREAK AN INPUT LOOP WITHOUT DISCONNECTING THE TERMINAL DEVICE. Other, more controlled loops are explained later. Line 50 is not executed until the loop is broken by entering a C when input is requested.*

Line 50 informs the computer that the program is finished:

```
50 END
```

# SECTION II

## THE ESSENTIALS OF BASIC

### HOW TO READ THIS SECTION

This section contains enough information to allow you to use BASIC in simple applications, without using the capability of storing programs.

Proceed at your own pace. The information in the vocabulary and operators subsections is included for completeness; experienced programmers may skip these. Programmers with some knowledge of BASIC may also concentrate on capabilities of the TSB system presented in the commands subsection.

The "Operators" subsections contain brief descriptions, rather than explanations, of the logical operators. The novice should not expect to gain a clear understanding of logical operators from this presentation. Section VII presents more details and examples of TSB logical operations. Readers wishing to make best use of TSB logical capabilities should consult this section. Those unfamiliar with logical operations should also refer to an elementary logic text.

A simple program is included at the end of this section for reference; it contains a running commentary on the uses of many of the BASIC statements presented in the section.

## TERM: NUMBER

DEFINED IN TSB AS: A positive or negative Decimal number whose magnitude is between an approximate minimum of  $10^{-38}$  (or  $2^{-129}$ ) and an approximate maximum of  $10^{38}$  (or  $2^{127}$ ). Zero is included in this range.

## TERM: E NOTATION

DEFINED IN TSB AS: A means of expressing numbers having more than six decimal digits, in the form of a decimal number raised to some power of 10.

EXAMPLES:  $1.00000E+06$  is equal to 1,000,000 and is read "1 times 10 to the sixth power ( $1 \times 10^6$ ).

$1.02000E+04$  is equal to 10,200

$1.02000E-04$  is equal to .000102

### COMMENTS

"E" notation is used to print numbers greater than six digits. It may also be used to input any number.

When entering numbers in "E" notation, leading and trailing zeroes may be omitted from the number; the + sign and leading zeroes may be omitted from the exponent.

The precision of numbers is 6 to 7 decimal digits (23 binary digits).

## TERM: SIMPLE VARIABLE

DEFINED IN TSB AS: A letter (from A to Z); or a letter immediately followed by a number (from 0 to 9).

EXAMPLES:           A0       B  
                      M5       C2  
                      Z9       D

### COMMENTS

Variables are used to represent numeric values.

For instance, in the statement:

```
10 LET M5 = 96.7
```

M5 is a variable; 96.7 becomes the value of the variable M5.

There are two other types of variables in TSB, string and array variables; their use is explained in Sections V and VI respectively.

## TERM: EXPRESSION

DEFINED IN TSB AS: A combination of variables, constants and operators which has a numeric value.

EXAMPLES:  $(P + 5)/27$   
(where P has previously been assigned a numeric value.)  
 $Q - (N + 4)$   
(where Q and N have previously been assigned numeric values.)

---

## TERM: ARITHMETIC EVALUATION

DEFINED IN TSB AS: The process of calculating the value of an expression.

# THE ASSIGNMENT OPERATOR

SYMBOL:	=
EXAMPLES:	1Ø LET A = B2 = C = Ø 2Ø LET A9 = C5 3Ø Y = (N-(R+5))/T 4Ø N5 = A + B2 5Ø P5 = P6 = P7 = A = B = 98.6
GENERAL FORM:	LET <u>variable</u> = <u>expression</u> <u>variable</u> = <u>expression</u>

## PURPOSE

Assigns an arithmetic or logical value to a variable.

## COMMENTS

When used as an assignment operator, = is read "takes the value of," rather than "equals". It is, therefore, possible to use assignment statements such as:

1ØØ LET X = X+2

This is interpreted by TSB as: "LET X take the value of (the present value of) X, plus two."

Several assignments may be made in the same statement, as in statements 1Ø and 5Ø above.

See Section VII, "LOGICAL OPERATIONS" for a description of logical assignments.

# ARITHMETIC OPERATORS

SYMBOLS:	$\uparrow$ * / + -
EXAMPLES:	4Ø LET N1 = X-5
	5Ø LET C2 = N+3
	6Ø LET A = (B-C)/4
	7Ø LET X = ((P+2)-(Y*X))/N+Q

## PURPOSE

Represents an arithmetic operation, as:

exponentiate:	$\uparrow$
multiply:	*
divide:	/
add:	+
subtract:	-

## COMMENTS

The "-" symbol is also used as a sign for negative numbers.

It is good practice to separate arithmetic operations with parentheses when unsure of the exact order of precedence.

The order of precedence (hierarchy) is:

$\uparrow$
* /
+ -

with  $\uparrow$  having the highest priority. Operators on the same level of priority are acted upon from left to right in a statement. See "Order of Precedence" in this Section for examples.

# RELATIONAL OPERATORS

SYMBOLS:	=	#	<>	>	<	>=	<=
EXAMPLES:	100	IF A=B	THEN	900			
	110	IF A+B >C	THEN	910			
	120	IF A+B < C+E	THEN	920			
	130	IF C>= D*E	THEN	930			
	140	IF C9<= G*H	THEN	940			
	150	IF P2#C9	THEN	950			
	160	IF J <> K	THEN	950			

## PURPOSE

Determines the logical relationship between two expressions, as

equality: =

inequality: # or: <>

greater than: >

less than: <

greater than or equal to: >=

less than or equal to: <=

## COMMENTS

*NOTE: It is not necessary for the novice to understand the nature of logical evaluation of relational operators, at this point. The comments below are for the experienced programmer.*

Expressions using relational operators are logically evaluated, and assigned a value of "true" or "false" (the numeric value is 1 for "true", and 0 for false).

When the = symbol is used in such a way that it might have either an assignment or a relational function, TSB assumes it is an assignment operator. For a description of the assignment statement using logical operators, see Section VII, "Logical Operations".

## MIN AND MAX OPERATORS

```
EXAMPLES:  10 LET A=A9=P2=P5=C2=X=7.5
           20 LET B5=D8=Q1=Q4=Y=B=12.0
           ⋮
           80 PRINT (A MIN 10)
           90 LET B=(A MIN 10)+100
          100 IF (A MIN B5) > (C2 MIN D8) THEN 10
          110 PRINT (X MAX Y)
          120 IF (A9 MAX B) <= 5 THEN 150
```

### PURPOSE

Selects the larger or smaller value of two expressions.

### COMMENTS

In the examples above, statement 110 selects and prints the larger value: since  $X = 7.5$  and  $Y = 12.0$ , the value of  $Y$  is printed. The evaluation is made first, then the statement type (PRINT) is executed.

## THE AND OPERATOR

```
SYMBOL:      AND
EXAMPLES:    6Ø IF A9<B1 AND C#5 THEN 1ØØ
              7Ø IF T7#T AND J=27 THEN 15Ø
              8Ø IF P1 AND R>1 AND N AND V2 THEN 1Ø
              9Ø PRINT X AND Y
```

### PURPOSE

Forms a logical conjunction between two expressions. If both are "true", the conjunction is "true"; if one or both are "false", the conjunction is "false".

*NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.*

### COMMENTS

The numeric value of "true" is 1, of "false" is 0.

All non-zero values are "true". For example, statement 9Ø would print either a 0 or a 1 (the logical value of the expression X AND Y) rather than the actual numeric values of X and Y.

Control is transferred in an IF statement using AND, only when all parts of the AND conjunction are "true". For instance, example statement 8Ø requires four "true" conditions before control is transferred to statement 1Ø.

See Section VII, "Logical Operations" for a more complete description of logical evaluation.

## THE OR OPERATOR

SYMBOL: OR

EXAMPLES: 100 IF A>1 OR B<5 THEN 500  
110 PRINT C OR D  
120 LET D = X OR Y  
130 IF (X AND Y) OR (P AND Q) THEN 600

### PURPOSE

Forms the logical disjunction of two expressions. If either or both of the expressions is true, the OR disjunction is "true"; if both expressions are "false" the OR disjunction is "false".

*NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.*

### COMMENTS

The numeric values are: "true" = 1, "false" = 0.

All non-zero values are true; all zero values are false.

Control is transferred in an IF statement using OR, when either or both of the two expressions evaluate to "true".

See Section VII, "Logical Operations" for a more complete description of logical evaluation.

# THE NOT OPERATOR

SYMBOL:	NOT
EXAMPLES:	30 LET X = Y = 0 35 IF NOT A THEN 300 45 IF (NOT C) AND A THEN 400 55 LET B5 = NOT P 65 PRINT NOT (X AND Y) 70 IF NOT (A=B) THEN 500

## PURPOSE

Logically evaluates the complement of a given expression.

*NOTE: It is not necessary for the novice to understand how this operator works. The comments below are intended for experienced programmers.*

## COMMENTS

If  $A = 0$ , then  $\text{NOT } A = 1$ ; if  $A$  has a non-zero value,  $\text{NOT } A = 0$ .

The numeric values are: "true" = 1, "false" = 0; for example, statement 65 above would print "1", since the expression  $\text{NOT } (X \text{ AND } Y)$  is true.

Note that the logical specifications of an expression may be changed by evaluating the complement. In statement 35 above, if  $A$  equals zero, the evaluation would be "true" (1); since  $A$  has a numeric value of 0, it has a logical value of "false", making  $\text{NOT } A$  "true".

See Section VII, "Logical Operations" for a more complete description of logical evaluation.

# ORDER OF PRECEDENCE OF EXECUTION

The order of performing operations is:

↑ *highest precedence*

NOT

\* /

+ -

MIN MAX

*Relational Operators*

AND

OR *lowest precedence*

If two operators are on the same level,  
the order of execution is left to right,  
for example:

$5 + 6 * 7$  is evaluated as:  $5 + (6 * 7)$

$7 / 14 * 2 / 5$  is evaluated as:  $\frac{(7 / 14) * 2}{5}$

A MIN B MAX C MIN D is evaluated as:  
 $((A \text{ MIN } B) \text{ MAX } C) \text{ MIN } D$

Parentheses override the order of precedence  
in all cases.

# STATEMENTS

Be sure you know the difference between statements and commands.

Statements are instructions to the computer. They are contained in numbered lines within a program, and execute in the order of their line numbers. Statements cannot be executed without running a program. They tell the computer what to do while a program is running.

Commands are also instructions. They are executed immediately, do not have line numbers, and may not be used in a program. They are used to manipulate programs, and for utility purposes, such as logging on and off.

Here are some examples mentioned in Section I:

<u>Statements</u>	<u>Commands</u>
LET	HELLO
PRINT	BYE
INPUT	LIST

Do not attempt to memorize every detail in the "Statements" subsection; there is too much material to master in a single session. By experimenting with the sample programs, and attempting to write your own programs, you will learn more quickly than by memorizing.

# THE ASSIGNMENT STATEMENT

EXAMPLES:

1Ø LET A = 5.Ø2

2Ø X = Y7 = Z = Ø

3Ø B9 = 5\* (X+2)

4Ø LET D = (3\*C2+N)/(A\*(N/2))

GENERAL FORM:

statement number LET variable = number or expression or string or variable...

or

statement number variable = number or expression or string or variable...

## PURPOSE

Used to assign or specify the value of a variable.  
The value may be an expression, a number, string  
or a variable of the same type.

## COMMENTS

Note that LET is an optional part of the assignment  
statement.

The assignment statement must contain:

1. The variable to be assigned a value.
2. The assignment operator, an = sign.
3. The number, expression or variable to be  
assigned to the variable.

Statement 2Ø in the example above shows the use of  
an assignment to give the same value (Ø) to several  
variables. This is a valuable feature for initial-  
izing variables in the beginning of a program.

# REM

EXAMPLES:                   1Ø REM--THIS IS AN EXAMPLE  
                              2Ø REM: OF REM STATEMENTS  
                              3Ø REM-----/////**\*\*\*\*\*!!!!**  
                              4Ø REM. STATEMENTS ARE NOT EXECUTED BY TSB

GENERAL FORM: statement number REM any remark or series of characters

## PURPOSE

Allows insertion of a line of remarks or comment in the listing of a program.

## COMMENTS

Must be preceded by a line number. Any series of characters may follow REM.

REM lines are saved as part of a BASIC program, and printed when the program is listed or punched; however, they are ignored when the program is executing.

Remarks are easier to read if REM is followed by a punctuation mark, as in the example statements.

# GO TO AND MULTIBRANCH GO TO

EXAMPLES:            10 LET X = 20  
                          :  
                          40 GO TO X+Y OF 410,420,430  
                          50 GOTO 100  
                          80 GOTO 10  
                          90 GO TO N OF 100,150,180,190

## GENERAL FORM:

statement number GO TO statement number

statement number GO TO expression OF sequence of statement numbers

## PURPOSE

GO TO transfers control to the statement specified.

GO TO expression...rounds the expression to an integer n and transfers control to the nth statement number following OF.

## COMMENTS

GO TO may be written: GOTO or GO TO.

Must be followed by the statement number to which control is transferred, or expression OF, and a sequence of statement numbers.

GO TO overrides the normal execution sequence of statements in a program.

If there is no statement number corresponding to the value of the expression, the GO TO is ignored.

Useful for repeating a task infinitely, or "jumping" (GOing TO) another part of a program if certain conditions are present.

GO TO should not be used to enter FOR-NEXT loops; doing so may produce unpredictable results or fatal errors.

# IF...THEN

```
SAMPLE PROGRAM:      10 LET N = 10
                     20 READ X
                     30 IF X <=N THEN 60
                     40 PRINT "X IS 10 OR OVER"
                     50 GO TO 80
                     60 PRINT "X IS LESS THAN 10"
                     70 GO TO 20
                     80 END
                       :
```

GENERAL FORM: statement number IF expression THEN statement number

## PURPOSE

Transfers control to a specified statement if a specified condition is true.

## COMMENTS

Sometimes described as a conditional transfer; "GO TO" is implied by IF...THEN, if the condition is true. In the example above, if  $X \leq 10$ , the message in statement 60 is printed.

Since numbers are not always represented exactly in the computer, the = operator should be used carefully in IF...THEN statements.  $\leq$ ,  $\geq$ , etc. should be used in the IF expression, rather than =, whenever possible.

If the specified condition for transfer is not true, then the program will continue executing in sequence. In the example above, if  $X > 10$ , the message in statement 40 will be printed.

See "Logical Operations", Section VII for a more complete description of logical evaluation.

# FOR...NEXT

EXAMPLES:           100 FOR P1 = 1 TO 5  
                      110 FOR Q1 = N TO X  
                      120 FOR R2 = N TO X STEP 1  
                      130 FOR S = 1 TO X STEP Y  
                      140 NEXT S  
                      150 NEXT R2  
                      160 NEXT Q1  
                      170 NEXT P1

---

## *Sample Program - Variable Number Of Loops*

```
40 PRINT "HOW MANY TIMES DO YOU WANT TO LOOP";  
50 INPUT A  
60 FOR J = 1 TO A  
70 PRINT "THIS IS LOOP"; J  
80 READ N1, N2, N3  
90 PRINT "THESE DATA ITEMS WERE READ:" N1; N2; N3  
100 PRINT "SUM ="; (N1+N2+N3)  
110 NEXT J  
120 DATA 5, 6, 7, 8, 9, 10, 11, 12  
130 DATA 13, 14, 15, 16, 17, 18, 19, 20, 21  
140 DATA 22, 23, 24, 25, 26, 27, 28, 29, 30  
150 DATA 31, 32, 33, 34  
160 END
```

## GENERAL FORM:

statement number FOR simple variable = initial value TO final value

or

statement number FOR simple variable = initial value TO final value STEP step value

(Statements to be repeated)

⋮

statement number NEXT simple variable

**NOTE:** The same simple variable must be used in both the FOR and NEXT statements of a loop.

# FOR...NEXT, CONTINUED

## PURPOSE

Allows repetition of a group of statements within a program.

## COMMENTS

Initial value, final value and step value may be any expression.

How the loop works:

The simple variable is assigned the value of the initial value; the value of the simple variable is increased by 1 (or by the optional step value) each time the loop executes.

When the value of the simple variable passes the final value, control is transferred to the statement following the "NEXT" statement.

STEP and step value are optional.

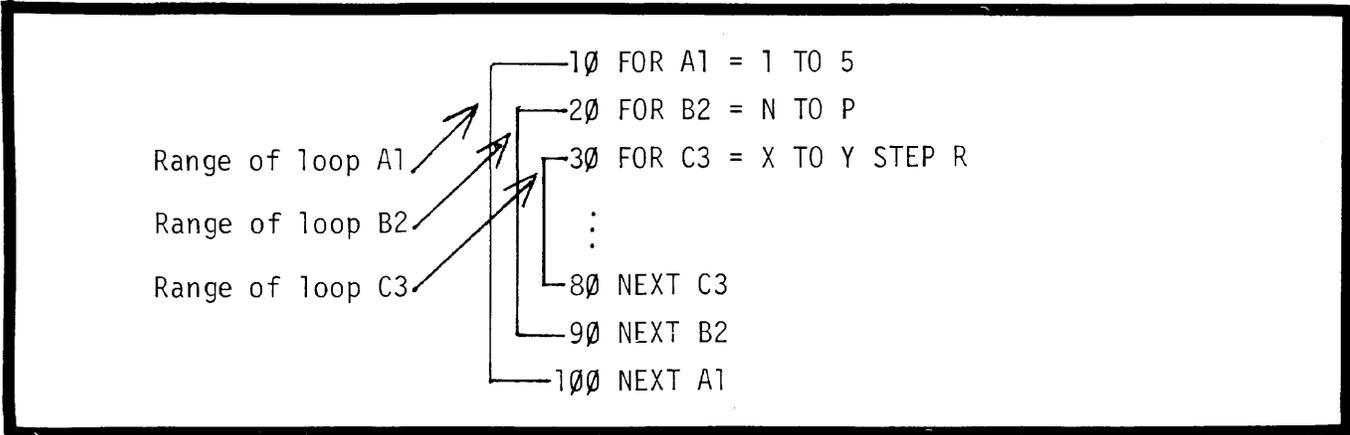
For further details on the STEP feature, see "FOR...NEXT with STEP" in Section III.

Try running the sample program if you are not sure what happens when FOR...NEXT loops are used in a program.

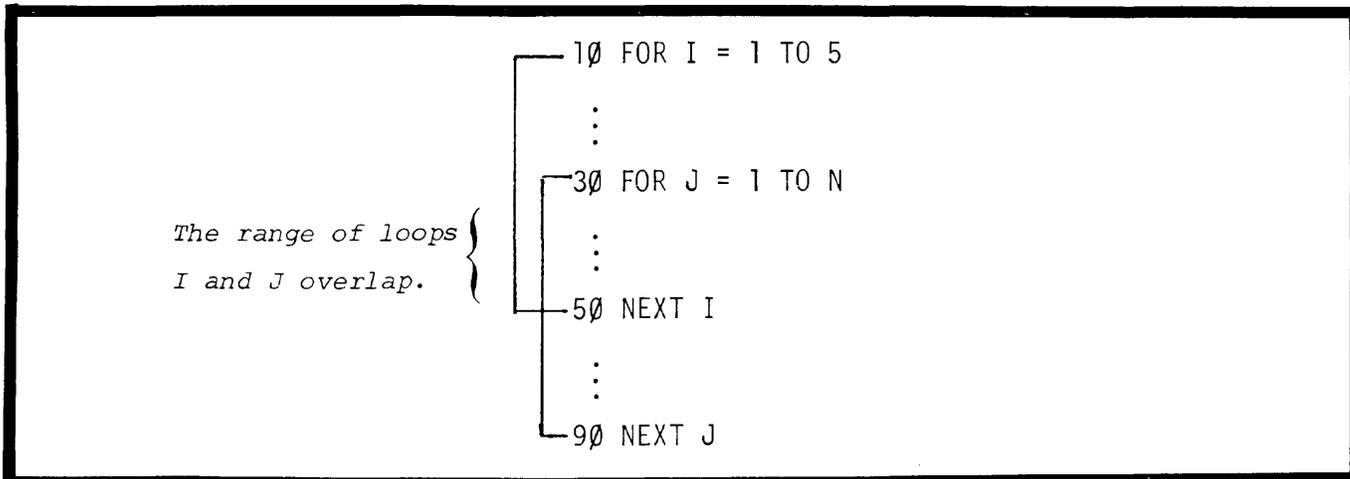
# NESTING FOR...NEXT LOOPS

Multiple FOR...NEXT loops may be used in the same program; they may also be nested (placed inside one another). There are two important features of FOR...NEXT loops:

1. FOR...NEXT loops may be nested.



2. The range of FOR...NEXT loops may not overlap. The loops in the example above are nested correctly. This example shows improper nesting.



# READ, DATA AND RESTORE

Sample Program using READ and DATA

```
15 FOR I=1 TO 5
20 READ A
40 LET X=A^2
45 PRINT A;" SQUARED =" ;X
50 NEXT I
55 DATA 5.24,6.75,30.8,72.65,89.72
60 END
```

Each data item may be read only once in this program. TSB keeps track of data with a "pointer". When the first READ statement is encountered, the "pointer" indicates that the first item in the first DATA statement is to be read; the pointer is then moved to the second item of data, and so on.

In this example, after the loop has executed five times, the pointer remains at the end of the data list. To reread the data, it is necessary to reset the pointer. A RESTORE statement moves the pointer back to the first data item.

# READ, DATA AND RESTORE, CONTINUED

## Sample Program Using READ, DATA and RESTORE

```
20 FOR I=1 TO 5
30 READ A
40 LET X=A+2
50 PRINT A; "SQUARED =";X
60 NEXT I
80 RESTORE
100 FOR J=1 TO 5
110 READ B
120 LET Y=B+4
130 PRINT B; "TO THE FOURTH POWER =";Y
140 NEXT J
150 DATA 5.24,6.75,30.8,72.65,89.72
160 END
```

### GENERAL FORM:

statement number READ variable , variable ,...

statement number DATA number or string , number or string ,...

statement number RESTORE

statement number RESTORE statement number

### PURPOSE

The READ statement instructs TSB to read an item from a DATA statement.

The DATA statement is used for specifying data in a program. The data is read in sequence from first to last DATA statements, and from left to right within the DATA statement.

The RESTORE statement resets the pointer to the first data item, allowing data to be re-read.

RESTORE followed by a statement number resets the pointer to the first data item, beginning at the specified statement.

# READ, DATA AND RESTORE, CONTINUED

## COMMENTS

READ statements require at least one DATA statement in the same program.

Items in a DATA statement must be separated by commas. String and numeric data may be mixed.

DATA statements may be placed anywhere in a program. The data items will be read in sequence as required.

DATA statements do not execute; they merely specify data.

The RUN command automatically sets the pointer to the first data item.

If you are not sure of the effects of READ, DATA, and RESTORE, try running the sample programs.

Programmers mixing string and numeric data may find the TYP function useful. See "The TYP Function", Section III.

# INPUT

This program shows several variations of the INPUT statement and their effects.

## Sample Program Using INPUT

```
5 FOR M=1 TO 2
10 INPUT A
20 INPUT A1,B2,C3,Z0,Z9,E5
30 PRINT "WHAT VALUE SHOULD BE ASSIGNED TO R";
40 INPUT R
50 PRINT A;A1;B2;C3;Z0;Z9;E5;"R=";R
60 NEXT M
70 END
```

---

## RESULTS

RUN

?1 return

?2,3,4,5,6,7 return

WHAT VALUE SHOULD BE ASSIGNED TO R?27 return

1 2 3 4 5 6 7 R=27

?1.5 return

?2.5,3.5,4.5,6.,7.2 return

??8.1 return ?? indicates that more input is expected

WHAT VALUE SHOULD BE ASSIGNED TO R?-99

1.5 2.5 3.5 4.5 6 7.2

8.1 R=-99

DONE

GENERAL FORM:

statement number INPUT variable , variable ,...

## PURPOSE

Assigns a value input from the teleprinter to a variable.

# INPUT CONTINUED

## COMMENTS

The program comes to a halt, and a question mark is printed when the INPUT statement is used. The program does not continue execution until the input requirements are satisfied.

Only one question mark is printed for each INPUT statement. The statements:

1Ø INPUT A, B2, C5, D, E, F, G

*and*

2Ø INPUT X

each cause a single "?" to be printed. Note that the "?" generated by statement 1Ø requires seven input items, separated by commas, while the "?" generated by statement 2Ø requires only a single input item.

The only way to stop a program when input is required is entering: C<sup>C</sup> return. Note that the C<sup>C</sup> aborts the program; it must be restarted with the RUN command.

Relevant Diagnostics:

? indicates that input is required.

?? indicates that more input is needed to satisfy an INPUT statement.

??? indicates that TSB cannot decipher your input.

ENTRA INPUT-WARNING ONLY indicates that a) extra input was entered; b) it has been disregarded; and c) the program is continuing execution.

See the description of the "PRINT" format in this section for variations on output formats.

# PRINT

This sample program gives a variety of examples of the PRINT statement.  
The results are shown below.

```
10 LET A=B=C=10
20 LET D1=E9=20
30 PRINT A,B,C,D1,E9
40 PRINT A/B,B/C/D1+E9
50 PRINT "NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE"
60 PRINT "VALUE IN THE SAME STATEMENT."
70 PRINT
80 PRINT
90 REM* "PRINT" WITH NO OPERAND CAUSES THE TELEPRINTER TO SKIP A LINE.
100 PRINT "'A' DIVIDED BY 'E9' =" ; A/E9
110 PRINT "11111","22222","33333","44444","55555","66666"
120 PRINT "11111";"22222";"33333";"44444";"55555";"66666"
130 END
```

----- RESULTS -----

RUN return

```
10          10          10          20          20
1          20.05
NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE
VALUE IN THE SAME STATEMENT.

'A' DIVIDED BY 'E9' = .5
11111          22222          33333          44444          55555
66666
111112222233333444445555566666
DONE
```

*NOTE: The "," and ";" used in statements 110 and 120 have very different effects on the format.*

# PRINT, CONTINUED

## GENERAL FORM:

statement number PRINT expression , expression , ...

or

statement number PRINT "any text" ; expression ; ...

or

statement number PRINT "text" ; expression ; "text" , "text" , ...

or

statement number PRINT any combination of text and/or expressions

or

statement number PRINT

## PURPOSE

Causes the operand(s) to be output to the teleprinter or terminal device.

Causes the teleprinter to skip a line when used without an operand.

## COMMENTS

Note the effects of , and ; on the output of the sample program. If a comma is used to separate PRINT operands, five fields will be printed per teleprinter line. If semicolon is used, up to twelve "packed" numeric fields will be output per teleprinter line, or 72 characters.

# END AND STOP

## EXAMPLES:

```
200 IF A # 27.5 THEN 350
:
300 STOP
:
350 LET A = 27.5
:
500 IF B # A THEN 9999
:
550 PRINT "B = A"
600 END
9999 END
```

## GENERAL FORM:

```
any statement number STOP
any statement number END
Highest statement number in program END
```

## PURPOSE

Terminates execution of the program and returns control to TSB.

## COMMENTS

The highest numbered statement in the program must be an END statement.

END and STOP statements may be used in any portion of the program to terminate execution.

END and STOP have identical effects; the only difference is that the highest numbered statement in a program must be an END statement.

# SAMPLE PROGRAM

If you understand the effects of the statement types presented up to this point, skip to the "COMMANDS" section.

The sample program on the next two pages uses several BASIC statement types.

Running the program gives a good idea of the various effects of the PRINT statement on teleprinter output. If you choose to run the program, you may save time by omitting the REM statements.

After running the program, compare your output with that shown under "RUNNING THE SAMPLE PROGRAM". If there is a difference, LIST your version and compare it with the one presented on the next two pages. Check your PRINT statements for commas and semicolons; they must be used carefully.

# SAMPLE PROGRAM

```
10 REMARK: "REMARK" OR "REM" IS USED TO INDICATE REMARKS OR COMMENTS
20 REMARK: THE USER WANTS TO INCLUDE IN THE TEXT OF HIS PROGRAM.
30 REM: THE COMPUTER LISTS AND PUNCHES THE "REM" LINE, BUT DOES NOT
40 REM: EXECUTE IT.
50 REM: "PRINT" USED ALONE GENERATES A "RETURN" "LINEFEED"
60 PRINT
70 PRINT "THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY."
80 PRINT
90 PRINT "IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS."
100 PRINT
110 PRINT "PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY."
120 PRINT
130 PRINT
140 REM: FIRST, ALL VARIABLES USED IN THE PROGRAM ARE INITIALIZED
150 REM: TO ZERO (THEIR VALUE IS SET AT ZERO.)
160 LET A=N=R1=S=0
180 REM: NOW THE USER WILL BE GIVEN A CHANCE TO SPECIFY HOW MANY
190 REM: NUMBERS HE WANTS TO AVERAGE.
200 PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE";
210 INPUT N
220 PRINT
230 PRINT "O.K., TYPE IN ONE OF THE ";N;"NUMBERS AFTER EACH QUES. MARK."
240 PRINT "DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER."
250 PRINT
260 PRINT "NOW, LET'S BEGIN"
270 PRINT
280 PRINT
300 REM: "N" IS NOW USED TO SET UP A "FOR-NEXT" LOOP WHICH WILL READ
310 REM: 1 TO "N" NUMBERS AND KEEP A RUNNING TOTAL.
320 FOR I=1 TO N
330 INPUT A
340 LET S=S+A
350 NEXT I
360 REM: "I" IS A VARIABLE USED AS A COUNTER FOR THE NUMBER OF TIMES
```

## SAMPLE PROGRAM CONTINUED

```
370 REM: THE TASK SPECIFIED IN THE "FOR-NEXT" LOOP IS PERFORMED.
380 REM: "I" INCREASES BY 1 EACH TIME THE LOOP IS EXECUTED.
390 REM: "A" IS THE VARIABLE USED TO REPRESENT THE NUMBER TO BE
400 REM: AVERAGED. THE VALUE OF "A" CHANGES EACH TIME THE
410 REM: USER INPUTS A NUMBER.
420 REM: "S" WAS CHOSEN AS THE VARIABLE TO REPRESENT THE SUM
430 REM: OF ALL NUMBERS TO BE AVERAGED.
440 REM: AFTER THE LOOP IS EXECUTED "N" TIMES, THE PROGRAM CONTINUES.
460 REM: A SUMMARY IS PRINTED FOR THE USER.
470 PRINT
480 PRINT
490 PRINT N; "NUMBERS WERE INPUT."
500 PRINT
510 PRINT "THEIR SUM IS: ";S
520 PRINT
530 PRINT "THEIR AVERAGE IS: ";S/N
540 PRINT
550 PRINT
570 REM: NOW THE USER WILL BE GIVEN THE OPTION OF QUITTING OR
580 REM: RESTARTING THE PROGRAM.
590 PRINT "DO YOU WANT TO AVERAGE ANOTHER GROUP OF NUMBERS?"
600 PRINT
610 PRINT "TYPE 1 IF YES, 0 IF NO"
620 PRINT "BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER."
630 PRINT
640 PRINT "YOUR REPLY";
650 INPUT R1
660 IF R1=1 THEN 120
670 REM: THE FOLLOWING LINES ANTICIPATE A MISTAKE IN THE REPLY.
680 IF R1#0 THEN 700
690 GO TO 720
700 PRINT "TO REITERATE, YOU SHOULD TYPE 1 IF YES, 0 IF NO."
710 GO TO 640
720 END
```

# RUNNING THE SAMPLE PROGRAM

RUN return

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY.

IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS.

PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.

HOW MANY NUMBERS DO YOU WANT TO AVERAGE? 5 return

O.K.,TYPE IN ONE OF THE 5 NUMBERS AFTER EACH QUES. MARK.

DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER.

NOW, LET'S BEGIN

? 99 return

? 87.6 return

? 92.7 return

? 79.5 return

? 84 return

5 NUMBERS WERE INPUT.

THEIR SUM IS: 442.8

THEIR AVERAGE IS: 88.56

DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?

TYPE 1 IF YES, 0 IF NO

BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.

YOUR REPLY? 2 return

TO REITERATE, YOU SHOULD TYPE 1 IF YES, 0 IF NO.

YOUR REPLY? 1 return

HOW MANY NUMBERS DO YOU WISH TO AVERAGE? C<sup>c</sup> return

DONE

# COMMANDS

Remember the difference between commands and statements (See "Statements" in this section).

Commands are direct instructions to the computer, and are executed immediately. They are used to manipulate programs, and for utility purposes.

Note that all TSB commands may be abbreviated to their first three letters. If information is required after a command, a hyphen "-" must be included. For example, when logging in:

```
HEL-H200,SECCCRCECT return
```

Do not try to memorize all of the details in the COMMANDS subsection. The various commands and their functions will become clear to you as you begin writing programs.

# HELLO -

EXAMPLE: HELLO-D007,POS<sup>C</sup>T return  
or  
HEL-C321,PASS,4 return

GENERAL FORM: HELLO-IDcode , password , terminal type code  
or  
HEL-IDcode , password , terminal type code  
or  
HEL-IDcode , password

## PURPOSE

The command used to log in to the TSB system.

## COMMENTS

ID codes and passwords are assigned by the system master or operator.

Several users with the same I.D. code may be logged on to the computer simultaneously, using different terminals.

The terminal subtype code tells the system what kind of terminal is being used. It is an integer from 0 to 5. The following table indicates the meaning of each.

<u>Terminal Subtype Code</u>	<u>Terminal</u>
0	HP 2600A, HP 2749A, ASR-33, ASR-35.
1	Execuport 300
2	ASR-37 <i>ASR-37</i>
3	Terminet-300
4	Memorex 1240 <i>Memorex 1240</i>
5	Univac DCT 500

The message ILLEGAL FORMAT is printed if the type code is less than 0, or greater than 5.

If terminal type code is omitted, 0 is assumed by default.

# BYE

EXAMPLE:	BYE <u>return</u> 009 MINUTES OF TERMINAL TIME
GENERAL FORM:	<u>BYE</u>

## PURPOSE

The command used to log out of the TSB system.

## COMMENTS

Causes the amount of terminal time used to be printed.

Breaks a telephone connection to the computer.

# ECHO-

EXAMPLES:	ECHO-OFF <u>return</u>
	ECHO-ON <u>return</u>
GENERAL FORM:	ECHO-ON
	or
	ECHO-OFF

## PURPOSE

Allows use of half duplex terminal.

## COMMENTS

Users with half duplex terminal equipment must first log on, then type the ECHO-OFF command; then input and output becomes legible.

ECHO-ON returns a user to the full-duplex mode.

May be abbreviated to its first three letters.

# RUN

EXAMPLE:	RUN <u>return</u>
	or
	RUN- 300 <u>return</u>
GENERAL FORM:	<u>RUN</u>
	<u>RUN- statement number</u>

## PURPOSE

Starts execution of a program at the lowest numbered statement when used without specifying a statement number.

Starts execution of a program at the specified statement when a statement number is used.

## COMMENTS

Note that when RUN- statement number is used, all statements before the specified statement will be skipped. Variables defined in statements which have been skipped are therefore considered to be undefined by TSB, and may not be used until they are defined in an assignment, READ, or LET statement.

A running program may be terminated by pressing the break key; or, to terminate a running program at some point when input is required, type:

C<sup>c</sup> return

# LIST

EXAMPLE:	<u>LIST</u> <u>return</u>
	<u>LIST</u> -100 <u>return</u>
	<u>LIST</u> -100, 200 <u>return</u>
GENERAL FORM:	<u>LIST</u>
	<u>LIST-</u> <u>statement number</u>
	<u>LIST-</u> <u>statement number</u> , <u>statement number</u>
	<u>LIST-</u> , <u>statement number</u>

## PURPOSE

Produces a listing of all statements in a program (in statement number sequence) when no statement number is specified.

When a statement number is specified, the listing begins at that statement.

When a second statement number is specified, listing ends with that statement.

When a "," and a statement number appear, listing starts at the beginning and ends with the specified statement.

## COMMENTS

A listing may be stopped by pressing the break key.

System library programs designated "RUN ONLY" by the operator cannot be listed.

May be abbreviated to its first three letters.

# SCRATCH

EXAMPLE:

SCRATCH return

*or*

SCR return

GENERAL FORM:

SCRATCH

*or*

SCR

## PURPOSE

Deletes (from memory) the program currently being accessed from the teleprinter.

## COMMENTS

Scratched programs are not recoverable. For information about saving programs on paper tape or in your personal library, see the NAME and SAVE commands in the next section, and PUNCH in this section.

# RENUMBER

EXAMPLES:

RENUMBER return

REN return

REN-100

REN-10, 1 return

REN-20, 50 return

GENERAL FORM:

REN

or

REN-number assigned to first statement

or

REN-number assigned to first statement , interval between new statement numbers

## PURPOSE

Renumbers statements in a Program.

## COMMENTS

GO TO's, GO SUB's, IF...THEN's, and RESTORE's are automatically reassigned the appropriate new numbers.

If no first statement number is specified, renumbering begins at statement 10, in intervals of 10.

If no interval is specified, the new numbers are spaced at intervals of 10, from the beginning statement.

Remember that numbers or text contained in REM and PRINT statements are not revised by RENUMBER.

# BREAK

EXAMPLES: break (Press the break key.)

## PURPOSE

Terminates a program being run.

Terminates the execution of LIST, PUNCH, CATALOG, and LIBRARY commands.

## COMMENTS

Pressing the break key signals the computer to terminate a program, producing the message: STOP.

When break is pressed during a listing, the message STOP is output.

Pressing break will not terminate the program if it is awaiting input (data to be typed in from the teleprinter). In this case the only means of ending the program is typing:

<sup>C</sup> return

which produces the DONE message.

break will not delete a program; however, the RUN command must be used to restart it.

# PUNCH

EXAMPLES:	PUNCH <u>return</u>
	PUN- 100, 200 <u>return</u>
	PUN-65 <u>return</u>
	PUN-, 300 <u>return</u>
GENERAL FORM:	<u>PUN</u>
	<u>PUN- statement number</u>
	<u>PUN- statement number , statement number</u>
	<u>PUN- , statement number</u>

## PURPOSE

Punches a program onto paper tape; also punches the program name, and leading and trailing feed holes on the tape; lists the program as it is punched. Can begin and/or end at specified statements (see LIST).

## COMMENTS

If the teleprinter is not equipped with a paper tape reader/punch, only a listing is produced.

Remember to press the paper tape punch "ON" button before pressing the return after PUNCH.

# XPUNCH

EXAMPLES:       XPUNCH return  
                  XPU- 65, P return  
                  XPU- P return

GENERAL FORM:  XPU  
                  XPU- statement number  
                  XPU- statement number , statement number  
                  XPU- statement number , statement number , P  
                  XPU- , statement number  
                  XPU- P

## PURPOSE

Punches a program onto paper tape; also punches the program name, and leading and trailing feed holes on the tape; lists the program as it is punched. Punching can begin and/or end at specified statements; "P" provides the pagination option (see LIST).

## COMMENTS

If the teleprinter is not equipped with a paper tape reader/punch, only a listing is produced.

Remember to press the paper tape punch "ON" button before pressing the return after PUNCH.

XPUNCH produces the same results as punch, but adds an X-OFF character at the end of each line (before return linefeed) to enable other BASIC programs to read the paper tape as data. (See Appendix B.)

# TAPE

EXAMPLES:

TAPE return

TAP return

GENERAL FORM:

TAPE

*or*

TAP

## PURPOSE

Informs the computer that following input is from paper tape.

## COMMENTS

TAPE suppresses any diagnostic messages which are generated by input errors, as well as the automatic linefeed after return. The KEY command (KEY return) or any other command, causes the diagnostic messages to be output to the teleprinter, ending the TAPE mode.

TSB responds to the TAPE command with a linefeed.

# KEY

EXAMPLES:	KEY <u>return</u>
GENERAL FORM:	<u>KEY</u>

## PURPOSE

Informs the computer that following input will be from the teleprinter keyboard; used only after a TAPE (paper tape input) sequence is complete; causes error messages suppressed by TAPE to be output to the teleprinter.

## COMMENTS

Any command followed by a return has the same effect as KEY. Commands substituted for KEY in this manner are not executed if diagnostic messages were generated during tape input.

# TIME

EXAMPLE:           TIME return  
                  CONSOLE TIME = 0 MINUTES. TOTAL TIME = 00 MINUTES

GENERAL FORM:    TIME

## PURPOSE

Produces listings of terminal time used since log on, and total time used for the account.

## COMMENTS

Time used by each ID code is recorded automatically by TSB. The system operator controls the accounting system. Consult your system operator for information about your system's accounting methods.

# DISC

EXAMPLE:       DISC return  
                  DISC ALLOWED = 100 SECTORS.   DISC USED = 0 SECTORS

GENERAL FORM: DISC

## PURPOSE

Produces listings of the total disc space allowed, and the disc space already used for the account.

## COMMENTS

The disc space used by each ID is recorded automatically each time a SAVE, OPEN, or KILL command is used. The total disc space allowed is controlled by the system operator. Consult your system operator for information about your system's accounting methods.

# MESSAGE

EXAMPLE:           MES-PLEASE MOVE PROGRAM "DUMMY" TO SYSTEM DISC, USER  
                          J122

GENERAL FORM:    MESSAGE-character string return

                          or

MES-character string return

## PURPOSE

Sends a character string to the system operator, preceded by the user's port (terminal) number.

## COMMENTS

Can be used to request information from the system operator, or to have programs moved to a particular disc.

If the system operator's message storage area is full, the message:

                  CONSOLE BUSY

will be printed on the user's terminal, indicating that the message has not been sent and should be entered again.

# SECTION III

## ADVANCED BASIC

This section describes more sophisticated capabilities of BASIC.

The experienced programmer has the option of skipping the "Vocabulary" subsection, and briefly reviewing the commands and functions presented here. The most important features of the TSB system--files, matrices, and strings are explained in the next three sections.

The inexperienced programmer need not spend a great deal of time on programmer-defined and standard functions. They are shortcuts, and some programming experience is necessary before their specifications become apparent.

## TERM: ROUTINE

DEFINED IN TSB AS: A sequence of program statements which produces a certain result.

### PURPOSE

Routines are used for frequently performed operations. Using routines saves the programmer the work of defining an operation each time he uses it, and saves computer memory space.

### COMMENTS

A routine may also be called a program, subroutine, or subprogram.

The task performed by a routine is defined by the programmer.

Examples of routines and subroutines are given in this section.

## TERM: ARRAY ( OR MATRIX )

DEFINED IN TSB AS: An ordered collection of numeric data. A single program can have up to 2000 total array elements (numeric values).

### COMMENTS

Arrays are referenced by columns (vertical) and rows (horizontal).

Arrays may have one or two dimensions. For example,

1.0  
2.1  
3.2  
4.3

is a one dimensional array, while

6 , 5 , 4  
3 , 2 , 1  
0 , 9 , 8

is a two dimensional array.

Array elements are referenced by their row and column position. For instance, if the examples above were arrays A and Z respectively, 2.1 would be A(2); similarly, 0 would be Z(3,1). The references to array elements are called subscripts, and set apart with parentheses. For example P(1,5) references the fifth element of the first row of array P; 1 and 5 are the subscripts. In X(M,N), M and N are the subscripts.

*NOTE: A2 is a simple variable; A(2) is an array element.*

## TERM: STRING

DEFINED IN TSB AS:           Ø to 72 teleprinter characters enclosed  
by quotation marks.

### COMMENTS

Sample strings: "ANY CHARACTERS!?\* / ---"  
"TEXT 1234567..."

Quotation marks may not be used within a  
string.

---

## TERM: FUNCTION

DEFINED IN TSB AS:           The mathematical relationship between two  
variables (X and Y for example) such that  
for each value of X there is one and only  
one value of Y.

### COMMENTS

The independent variable is called an argument;  
the dependent variable is the function value.  
For instance in

1ØØ LET Y = SQR(X)

X is the argument; the function value is the  
square root of X; and Y takes the value of the  
function.

## TERM: WORD

DEFINED IN TSB AS:       The amount of computer storage space occupied by two tele-printer characters.

### COMMENTS

Numeric characters contained in strings require the same amount of storage space as other characters.

Numbers (stored in binary format) require two words of storage each.

---

## TERM: RECORD

DEFINED IN TSB AS:       A storage unit containing 128 2-character words.

### COMMENTS

Further details on file storage are given in Section IV, "FILES".

# STORING AND DELETING PROGRAMS

Up to this point manipulation of programs has been limited to the "current" program, that is, the program being written or run at the moment. The only means of saving a program introduced thus far is the PUNCH command.

The commands on the following pages allow the user to create his own library of programs on the Time Shared BASIC system. Library programs are easily accessed, modified, and run.

The experienced programmer need only review the commands briefly -- they do what their names imply: NAME, SAVE, etc.

A word of caution for the inexperienced programmer: it is wise to make a "hard" copy (on paper tape) of programs you wish to use frequently. Although it is easy and convenient to store programs "on-system", you will make mistakes as you learn, and may accidentally delete programs. It is much less time consuming to enter a program from paper tape than rewrite it!

# LENGTH

EXAMPLES:	LENGTH <u>return</u>
	LEN <u>return</u>
	0000 WORDS
GENERAL FORM:	<u>LEN</u> <u>return</u>

## PURPOSE

Prints the number of two-character words in the program currently being accessed from the terminal. This is the amount of "storage space" needed to SAVE the program.

## COMMENTS

Each user has a working "space" of approximately 4000 two-character words. LEN is a useful check on total program length when writing long programs.



# SAVE

EXAMPLES:	SAVE <u>return</u>
	SAV <u>return</u>
GENERAL FORM:	<u>SAVE</u> or <u>SAV</u>

## PURPOSE

Saves a copy of the current program in the user's private library.

## COMMENTS

A program must be named before it can be saved.  
(See NAME, this section.)

No program or file in a user's library may have the same name as another program or file in that library. The procedure for saving a changed version of a program is as follows (the program name is SAMPLE):

KILL-SAMPLE <u>return</u>	(Deletes the stored version)
<u>linefeed</u>	
NAME-SAMPLE <u>return</u>	(Names the current program)
<u>linefeed</u>	
SAVE <u>return</u>	(Saves the current program, named SAMPLE)
<u>linefeed</u>	

For instructions on opening a file, see Section IV, "FILES."

# GET- AND GET- \$

EXAMPLES:	GET-PROGRAM <u>return</u>
	GET-MYPROG <u>return</u>
	GET-\$PUBLIC <u>return</u>
	GET-\$NAMES <u>return</u>
GENERAL FORM:	GET- <u>name of a program in user's library</u>
	GET-\$ <u>name of system library program</u>

## PURPOSE

GET- retrieves the specified program, making it the program currently accessed from the teleprinter.

GET-\$ retrieves the specified program from the system library, making it the program currently accessed from the teleprinter.

## COMMENTS

The program being accessed previous to using GET- is not recoverable unless it has been previously SAVED or PUNCHED (GET- performs an implicit SCRATCH).

For more information on public library programs, see "LIBRARY" in this section.

# KILL-

EXAMPLE:	KILL-PROG12 <i>return</i>
	KIL-EXMPLE <i>return</i>
	KIL-FILE1Ø <i>return</i>
GENERAL FORM:	<u>KILL- program or file to be deleted</u>
	or
	<u>KIL- program or file to be deleted</u>

## PURPOSE

Deletes the specified program or file from the user's library. (Does not delete the program currently being accessed from the teleprinter, even if it has the same name.)

## COMMENTS

CAUTION: Files have only one version, the stored one. A KILLED file is not recoverable.

A file may not be KILLED while it is being accessed by another user.

KILL-should be used carefully, as the KILLED program is not recoverable unless:

- a) A paper tape was previously PUNCHED, or
- b) The KILLED program was also the current program.

SCRATCH deletes the program currently being accessed from the teleprinter, while KILL deletes a program or file stored on-system. The stored and current versions of a program occupy separate places in the system. They may differ in content, even though they have the same name.

The sequence of commands for changing and storing a program named PROG\*\* is:

GET-PROG\*\* (Retrieves the program.)

(*make changes*)

KILL-PROG\*\* (Deletes the stored version.)

SAVE (Saves the current version.)

# APPEND-

EXAMPLES:	APPEND-MYPROG <u>return</u>
	APP-MYPROG <u>return</u>
	APPEND-\$PUBLIC <u>return</u>
	APP-\$SYSLIB <u>return</u>
GENERAL FORM:	<u>APPEND-program name</u>
	or
	<u>APP-program name</u>
	or
	<u>APP-\$system library program name</u>

## PURPOSE

Retrieves the named program from the user's or public library and appends it (attaches it) to the program currently being accessed from the teleprinter.

## COMMENTS

The lowest statement number of the APPENDED program must be greater than the highest statement number of the current program.

CAUTION: If an APPENDED public library program is "run-only", the entire program to which it is APPENDED becomes "run-only". ("Run-only" programs may not be listed, punched, or saved.)

The \$ in system library program names is needed to APPEND them. For details, see "LIBRARY" in this section.



# LIBRARY

EXAMPLES:

LIBRARY return

LIB return

BINOPO	Ø594	CDETER	Ø7Ø6	CSHFLO	1598	CURFIT	1618	DIFFEO	4466
QFILE	F544Ø	QFILE2	F544Ø	ROMINT	Ø299	SQE	Ø2Ø9		

GENERAL FORM:

LIBRARY

or

LIB

## PURPOSE

Produces an alphabetical listing of TSB system library program and file names, followed by the size of each, in two-character words. For files, "F" precedes the size. The absence of a letter indicates a saved program.

## COMMENTS

Public library programs are available to users; typing:

GET-\$ program name return

retrieves the specified program.

Public files are accessed with the FILES statement. (See Section IV, "FILES" for details.)

Programs designated "run-only" by User AØØØ may be RUN but not listed, punched, xpunched, or saved.

LIBRARY listings may be terminated with the break key.

# CATALOG

EXAMPLES: CAT return

CATALOG return

PROG1 0024 PROG2 2348 PROG3 1489

GENERAL FORM: CATALOG

*or*

CAT

## PURPOSE

Produces an alphabetical listing of the names of the programs and files stored on-system, under the user's account name and size of each in two-character words. "F" precedes the size for files. The absence of a letter indicates a saved program.

## COMMENTS

May be terminated with the break key

Programs are accessed with the GET command.

Files are accessed with a FILES statement. See Section IV, "Files" for details.

# SUBROUTINES AND FUNCTIONS

The following pages show TSB features useful for repetitive operations -- subroutines, programmer-defined and standard functions.

The programmer-controlled features, such as multibranch GOSUB's, FOR...NEXT with STEP, and DEF FN become more useful as the user gains experience, and learns to use them as shortcuts.

Standard mathematical and trigonometric functions are convenient timesavers for programmers at any level. They are treated as numeric expressions by TSB.

The utility functions TAB, SGN, TYP, and LEN also become more valuable with experience. They are used to control or monitor the handling of data by TSB, rather than for performing mathematical chores.

## GOSUB...RETURN

```
EXAMPLE:      50 READ A2
              60 IF A2<100 THEN 80
              70 GOSUB 400
              :
              380 STOP (STOP frequently precedes
                        the first statement of a subroutine, to
                        prevent accidental entry.)
              390 REM--THIS SUBROUTINE ASKS FOR A 1 OR 0 REPLY.
              400 PRINT "A2 IS>100"
              410 PRINT "DO YOU WANT TO CONTINUE";
              420 INPUT N
              430 IF N #0 THEN 450
              440 LET A2 = 0
              450 RETURN
              :
              600 END
```

```
GENERAL FORM: statement number GOSUB statement number starting subroutine
              :
              statement number RETURN
```

### PURPOSE

GOSUB transfers control to the specified statement number.

RETURN transfers control to the statement following the GOSUB statement which transferred control.

GOSUB...RETURN eliminates the need to repeat frequently used groups of statements in a program.

### COMMENTS

The portion of the program to which control is transferred must end with a RETURN statement.

RETURN statements may be used at any desired exit point in a subroutine.

GOSUB...RETURN's may be nested to a level of 9 (see "Nesting GOSUBs").

# MULTIBRANCH GOSUB

EXAMPLES:            20 GOSUB 3 OF 100,200,300,400,500  
                      60 GOSUB N+1 OF 200,210,220  
                      70 GOSUB N OF 80,180,280,380,480,580

GENERAL FORM:

statement number GOSUB expression OF sequence of statement numbers ...

## PURPOSE

GOSUB expression rounds the expression to an integer n and transfers control to the nth statement number following OF.

## COMMENTS

Subroutines should be exited only with a RETURN statement.

The expression indicates which of the specified subroutines will be executed. For example, statement 20, above transfers control to the subroutine beginning with statement 300. The expression specifies which statement in the sequence of five statements is used as the starting one in the subroutine.

The expression is evaluated as an integer. Non-integer values are rounded to the nearest integer.

If the expression evaluates to a number greater than the number of statements specified, or less than 1, the GOSUB is ignored.

Statement numbers in the sequence following OF must be separated by commas.

## NESTING GOSUB S

```
EXAMPLES:      100 GOSUB 200
                :
                200 LET A = R2/7
                210 IF A THEN 230
                220 GOSUB 250
                :
                250 IF A>B THEN 270
                260 RETURN
                270 GOSUB 600
                :
```

### PURPOSE

Allows selective use of subroutines within subroutines.

### COMMENTS

GOSUB's may be nested to a level of nine.

RETURN statements may be used at any desired exit point in a subroutine. Note, however, that nested subroutines are exited in the order in which they were entered. For example, if subroutine 250 (above) is entered from subroutine 200, 250 is exited before subroutine 200.

# FOR...NEXT WITH STEP

EXAMPLES:            20 FOR I5 = 1 TO 20 STEP 2  
                      40 FOR N2 = 0 TO -10 STEP -2  
                      80 FOR P = 1 TO N STEP R  
                      90 FOR X = N TO W STEP (N+2-V)  
                      ⋮

GENERAL FORM:

statement number FOR simple variable = expression TO expression STEP expression

## PURPOSE

Allows the user to specify the size of the increment of the FOR variable.

## COMMENTS

The step size need not be an integer. For instance,

100 FOR N = 1 TO 2 STEP .01

is a valid statement which produces approximately 100 loop executions, incrementing N by .01 each time. Since no binary computer represents all decimal numbers exactly, round-off errors may increase or decrease the number of steps when a non-integer step size is used.

A step size of 1 is assumed if STEP is omitted from a FOR statement.

A negative step size may be used, as shown in statement 40 above.

## DEF FN

EXAMPLE:           6Ø DEF FNA (B2) = A+2 + (B2/C)  
                      7Ø DEF FNB (B3) = 7\*B3+2  
                      8Ø DEF FNZ (X) = X/5

GENERAL FORM:

statement number DEF FN single letter A to Z ( simple variable ) = expression

### PURPOSE

Allows the programmer to define functions.

### COMMENTS

A maximum of 26 programmer-defined functions are possible in a program (FNA to FNZ).

Any operand in the program may be used in the defining expression; however such circular definitions as:

1Ø DEF FNA (Y) = FNB (X)  
2Ø DEF FNB (X) = FNA (Y)

causes infinite looping.

See the vocabulary at the beginning of this section for a definition of "function".

# GENERAL MATHEMATICAL FUNCTIONS

EXAMPLES:	642 PRINT EXP(N); ABS(N)
	652 IF RND (Ø) >=.5 THEN 9ØØ
	662 IF INT (R) # 5 THEN 91Ø
	672 PRINT SQR (X); LOG (X)
GENERAL FORM:	The general mathematical functions may be used as expressions, or as parts of an expression.

## PURPOSE

Facilitates the use of common mathematical functions by pre-defining them, as:

ABS ( <u>expression</u> )	the absolute value of the expression;
EXP ( <u>expression</u> )	the constant $e$ raised to the power of the expression value (in statement 642 above, $e^{\uparrow N}$ )
INT ( <u>expression</u> )	the largest integer $\leq$ the expression;
LOG ( <u>expression</u> )	the logarithm of the positively valued expression to the base $e$ ;
RND ( <u>expression</u> )	a random number between 1 and Ø.
SQR ( <u>expression</u> )	the square root of the positively valued expression.

## COMMENTS

The RND function can be restarted by having an initial call to RND with a negative argument. In this way a sequence of random numbers can be repeated.

# TRIGONOMETRIC FUNCTIONS

```
EXAMPLES:      500 PRINT SIN(X); COS(Y)
                510 PRINT 3*SIN(B); TAN (C2)
                520 PRINT ATN (22.3)
                530 IF SIN (A2) <1 THEN 800
                540 IF SIN (B3) = 1 AND SIN(X) <1 THEN 90
```

## PURPOSE

Facilitates the use of common trigonometric functions by pre-defining them, as:

SIN (expression) the sine of the expression (in radians);  
COS (expression) the cosine of the expression (in radians);  
TAN (expression) the tangent of the expression (in radians);  
ATN (expression) the arctangent of the expression (in radians).

## COMMENTS

The function is of the value of the expression (the value in parentheses, or argument).

The trigonometric functions may be used as expressions, or parts of an expression.

ATN returns the angle in radians.

See the next three pages for other standard functions.

## THE TAB AND SGN FUNCTIONS

EXAMPLES:           500 IF SGN (X) > -1 THEN 800  
                      510 LET Y = SGN(X)  
                      520 PRINT TAB (5); A2; TAB (20) "TEXT"  
                      530 PRINT TAB (N), X, Y, Z2  
                      540 PRINT TAB (X+2) "HEADING"; R5

GENERAL FORM:       The TAB and SGN functions may be used as  
                      expressions, or parts of an expression.

The function forms are:

TAB ( expression indicating column number )

SGN ( expression )

### PURPOSE

TAB (expression), when used in a PRINT statement, causes the teleprinter to move to the column number specified by the expression (0 to 71).

SGN (expression), returns a 1 if the expression is greater than 0, returns a 0 if the expression equals 0, returns a -1 if the expression is less than 0.

## THE TYP FUNCTION

EXAMPLES:	800 IF TYP (3) = 2 THEN 1000 850 PRINT TYP (N) 900 IF TYP (R) # X THEN 1200
GENERAL FORM:	TYP may be used as an expression or as part of an expression; the function form is: <u>TYP ( <i>file number formula</i> )</u>

### PURPOSE

If the file number formula is positive, TYP returns these values indicating the type of the next data item in a file: 1 = number; 2 = string; 3 = "end of file".

If the file number formula is zero, TYP returns these values for the next data item in a DATA statement: 1 = number; 2 = string; 3 for an "out of data" condition.

If the file number formula is negative, TYP returns these values for the next data item in a file: 1 = number; 2 = string; 3 = "end of file"; 4 = "end of record".

### COMMENTS

When using files as random storage devices, the file number formula should be negative, enabling TYP to return an "end of record" value. (See Section IV for details of file structure.)

## THE LEN FUNCTION

EXAMPLES:           580 IF LEN (B\$) >= 21 THEN 9999  
                      800 IF LEN (C\$) = R THEN 1000  
                      850 PRINT LEN (N\$)  
                      880 LET P5 = LEN (N\$)

GENERAL FORM:       The LEN function may be used as an expression, or  
                      part of an expression. The function form is  
                      LEN ( *string variable* )

### PURPOSE

Returns the length (number of characters)  
currently assigned to a string variable.

### COMMENTS

Note the difference between the LEN function  
and the LENGTH command. The command is used  
outside a program, and returns the working  
length of the current program in two-character  
words. The LEN function may be used only in  
a program statement.

# THE TIM FUNCTION

## EXAMPLES:

```
580 IF TIM (0) - A > 15 THEN 9000
```

```
700 LET A3 = TIM (B)
```

```
800 PRINT TIM (0) "MINUTES" TIM (1) "HOURS" TIM (2) "DAYS" TIM (3) "YEARS"
```

GENERAL FORM: TIM (X)

where if X = 0, TIM (X) = current minutes (0 to 59)

X = 1, TIM (X) = current hour (0 to 23)

X = 2, TIM (X) = current day (1 to 366)

X = 3, TIM (X) = current year (0 to 99)

## PURPOSE

Returns the current minute, hour, day or year.

## COMMENTS

Note the difference between the TIM function and the TIME command. The TIME command is used outside a program and gives the console time and total time used. The TIM function can only be used within a program statement.

# CHAIN

EXAMPLES:           100 CHAIN "PROG2"  
                      175 CHAIN \$PROG1  
                      5320 CHAIN \*\*\*

GENERAL FORM:

statement number CHAIN name of a program in a private library

or

statement number CHAIN \$name of a public library program

## PURPOSE

Retrieves the names program from the user's private library or the public library and executes it.

## COMMENTS

Communication between chained programs can be done using COMMON.

DEBUGGING HINT: Output to the teleprinter need not be from the program that is currently in memory. This is because output is buffered; the program that contained the PRINT statement may have CHAINED to another program before the output was actually printed. Therefore, if the programmer hits break, he cannot be sure that the program in memory was responsible in the printing.

# COM

EXAMPLES:           10 COM A,B,D\$(53),E(3,4),F2  
                      15 COM H2,K8, C\$(14)

GENERAL FORM:

statement number COM list of variables, dimensioned arrays and strings

## PURPOSE

Defines variables that can be accessed by more than one program.

## COMMENTS

The equivalence of COMmon variables in different programs is determined by their relative order in the COM statements. Thus, if one program contains the statement

10 COM A,B1,C\$(10)

and a second program contains the statements

1 COM X

2 COM Y,Z\$(10)

and the two programs are run in order, identifiers A and X refer to the same variable, as do identifiers B1 and Y, C\$ and Z\$.

There are certain restrictions on the use of COM:

1. COM statements must be the lowest numbered statements in the program.
2. A variable that is declared COMmon in one program can be accessed by another program only if all preceding COMmon variables in both programs are of the same type and size.

## COM, CONTINUED

3. Arrays and strings which are to be in common must be dimensioned in the COM statement and they must not also appear in DIM statements.

Variables in COM should be initialized by the first program that is run. After that, other programs containing equivalent COM definitions can be executed by GET and RUN or CHAIN. The COM variables will still have the same values. These values are destroyed, however, when a line of syntax is entered or a program is called that does not have equivalent COM.

### EXAMPLES

1Ø COM A,B,C,Q\$(63),F(3,6),S1	(In program A)	All variables in common
1Ø COM J,K,L,C\$(63),C(3,6),V	(In program B)	
1Ø COM A,B,C,Q\$(63),F(3,6),S1	(In program A)	Three variables in common
1Ø COM H,N,M,O	(In program B)	
1Ø COM A,B,C	(In program A)	No variables in common
1Ø COM S\$(45),A,B,C	(In program B)	
1Ø COM A,B,C	(In program A)	All variables in common.
1Ø COM V	(In program B)	
3Ø COM N,M		

# SECTION IV

## FILES

For those problems that require permanent data storage external to a particular program, the TSB system provides a data file capability. This allows flexible, direct manipulation of large volumes of data stored within the system itself. Special versions of the READ, PRINT, MAT READ, MAT PRINT, and IF statements allow you to read from and write onto mass storage files.

File programming offers two levels of complexity. Many problems can be solved using files treated simply as serial access storage devices. In this case, the program reads or writes a serial list of data items (either numbers or strings of characters) without regard to the underlying structure of the file. However, with additional programming effort, files can be used as random access storage devices. In this case, the program breaks the file into a series of logical subfiles that can be modified independently.

This section deals with the serial use of files, then internal file structure and random access use. Explanatory programming samples follow each series of frames in this section.

## TERM: FILE

DEFINED IN TSB AS: An area of memory external to the program where numbers and strings of characters can be stored and retrieved. Files are created by, and belong to, a particular user.

### COMMENTS

The user determines the name and size of a file. Files vary in size from 1 to 48 records. A record contains 128 16-bit words -- space for 64 numbers or 248 characters.

When a program stores some information in a file, the information remains there until it is changed or the file is eliminated. All the programs of a particular user can access this information.

Each program must declare its files with a FILES statement before it can access them. Each program can access up to 4 different files.

For each file declared in the program, there is a file pointer that keeps track of that program's current position within the file. The RUN command causes all these pointers to be reset to the beginning of the file. As the program reads or writes on a file, the pointer for the file is moved through the file.

# SERIAL FILE ACCESS

The program writes all the data items out into the file in serial order. Each write operation begins where the previous one left off. Then, to retrieve one of these items, the program must reset the pointer to the beginning of the file and read through the items until it comes to the desired item.

## SAMPLE SERIAL FILE ACCESS

OPEN-GHIJK,24

GHIJK is the name of the file.  
The file is 24 records long.

NAM-PROG1

100 FILES GHIJK

The FILES statement links the file into the program. From now on, the file is referenced by number; GHIJK is file #1. This allows programs to use different files by changing only the FILES statement.

200 INPUT A,B,C,D

300 PRINT #1;A,B,C,D

This is a serial file PRINT statement. It is identical to the normal PRINT statement except that a file number appears and the values of the variables are written onto the file, not the terminal.

## SERIAL FILE ACCESS, CONTINUED

```
400 INPUT A,B,C,D
```

```
500 PRINT #1;A,B,C,D
```

This PRINT stores the new values of the variables immediately following the previous values.

```
600 READ #1,1
```

This is a reset operation; it resets the pointer for file #1 to the beginning of the file.

```
700 READ #1; H1,H2,H3
```

This is a serial file READ statement. It assigns the first three values in the file to the three variables specified.

```
800 PRINT H1,H2,H3
```

```
900 READ #1; H1,H2,H3,H4,H5
```

This READs the remaining five values in the file into the five variables given. The values in the file are not disturbed.

```
1000 PRINT H1,H2,H3,H4,H5
```

```
2000 END
```

Try this example. It should print out the same numbers you type in.

Now, since you are done with the file, remove it from your library by using the KILL command:

```
KIL-GHIJK
```

# OPEN-

EXAMPLES            OPEN-FILE27, 20 return  
                      OPEN-SAMPLE, 48 return  
                      OPEN- \*\*\*\*, 10 return

## GENERAL FORM:

OPEN- 1 to 6 character file name , number of 128-word records in file

OPE- 1 to 6 character file name , number of 128-word records in file

## PURPOSE

Creates a file with a specified number of 128-word records and assigns it a name.

## COMMENTS

The file that is open is accessible only by the user I.D. number that OPENED it. (NOTE: Unprotected public library files can be read by all users.)  
The file remains OPEN until the same user KILLS it.

File names must conform to the same rules as program names.

The size of the file may vary from a minimum of 1 record to a maximum of 48 records.

If the system does not have enough storage space for the new file, the OPEN command is rejected and an error message is printed:

SYSTEM OVERLOAD

## OPEN-, CONTINUED

If the user does not have enough space left for the new file in the amount set for him by the system operator, the OPEN command is rejected and an error message is printed:

FILE SPACE FULL

If the name given in the OPEN command equals the name of an existing file or program, the command is rejected and an error message is printed:

DUPLICATE ENTRY

# KILL-

EXAMPLE:           KILL-NAMEXX return  
                  KIL-EXMPLE return  
                  KIL-FILE1Ø return

GENERAL FORM:     KILL-file to be deleted  
                  KIL-file to be deleted

## PURPOSE

Removes the named file from the user's library and releases the space it occupied for further storage. Users can only KILL their own files.

## COMMENTS

Files have only one version, the stored one. When a file is KILLED, all the information in it is lost.

If the file named is currently being accessed by a user on another terminal, the KILL command is rejected and an error message is printed:

FILE IN USE

# FILES

EXAMPLES:            10 FILES MATH, SCORE, SQRT

GENERAL FORM:

statement number FILES up to 16 file names separated by commas

## PURPOSE

Declares which files will be used in a program; assumes that the files will be OPENED before the program is RUN.

## COMMENTS

One FILES statement can appear in a program, with up to 4 files declared (duplicate entries are legal). The files are assigned numbers (from 1 to 4) in the order they are declared in the program. In the EXAMPLES above, MATH is file #1, SCORE is file #2, and SQRT is file #3.

These numbers are used in the program to reference the files. For instance, in the same example,

```
100 PRINT #2; A
```

would print the value of A into the file named SCORE. This feature allows most programming to be done independently of the files to be used. The FILES statements are then added just before running the program.

## FILES, CONTINUED

Public library files to be read (they cannot be written on) must also be declared in a FILES statement but with a \$ preceding the file name.

Users with the same I.D. number can share files, but only one user can write on a file at a time. I.D. codes beginning with an "A" (e.g., A067) are an exception to the rule; they may read or write on files at the same time.

# SERIAL FILE PRINT

EXAMPLES:           125 PRINT #4; A1,B2,C\$  
                      130 PRINT #4; D,E,F, "B,C,D,E"  
                      140 PRINT #M+N; B

GENERAL FORM:

statement number PRINT #file number formula ;  
list of data items separated by commas

## PURPOSE

Prints variables, numbers, or strings of characters consecutively on the specified file, starting after the last item previously read or printed.

## COMMENTS

The file number formula may be any expression; it is rounded to the nearest integer (from 1 through 4). If the value is n, then the nth file declared in the FILES statement is used.

The serial file PRINT always writes the indicated data items into the next available space in the file. However, since character strings may vary in length and each string must be wholly contained within a record, some space in each record may be left unused. You can calculate the number of words occupied by any string with a formula described under "Storage Requirements" in this section.

After a serial file PRINT, the file pointer is updated so that it points to the next available space.

# SERIAL FILE PRINT

The information written in a file remains there even when the program terminates. Therefore, the user can return a day or week later and access the data at that time. If a program terminates because of an error or the user typing break, the files may not have been completely updated.

*NOTE: Matrices can also be written on files using a  
MAT PRINT # statement described in Section V.*

# SERIAL FILE READ

EXAMPLES:           65 READ #4; A,B,C  
                      70 READ #3; B\$  
                      80 READ #N; A,B\$, C(5,6)  
                      90 READ #(N+1); A,B\$,C

GENERAL FORM:

statement number READ #file number formula ;  
list of data items separated by commas

## PURPOSE

Reads numbers and strings into variables consecutively from the specified file, starting after the last item read.

## COMMENTS

The file number formula is evaluated as in the serial file PRINT.

Both strings and numbers can be read, but the order of variable types must match the order of data item types exactly. The TYP Function provides a means of determining the type of the next item.

The serial file READ moves from record to record within a file automatically, as necessary to find the next data item. After a READ, the file pointer is updated, and a subsequent READ will start with the next consecutive data item. Record boundaries and unused portions of records are ignored.

## SERIAL FILE READ, CONTINUED

Matrices can also be read from files using a MAT READ # statement described in Section V.

*NOTE: Following a serial file PRINT, the file pointer should be reset to the beginning of the file before the data can be read. This is done using the reset operation described on the next page. A serial READ should not directly follow a serial PRINT.*

# RESETTING

EXAMPLE:           100 READ #1,1  
                  200 READ #2,1  
                  300 READ #M+N,1

GENERAL FORM:

statement number READ #file number formula , 1

## PURPOSE

Resets the file pointer to the beginning of the file specified by the file number formula.

## COMMENTS

READ #N,1 is used after a serial PRINT to prepare for a serial READ.

*NOTE: Do not use PRINT #1,1 to reset, as this erases the first record of the file.*

## LISTING CONTENTS OF A FILE

Here is a sample program that lists a file of unknown contents. It assumes that the file (DATUMS) has been previously filled serially by some other program.

NAM - LIST

100 FILES DATUMS

200 DIM A\$[72]

300 IF END #1 THEN 1000

The IF END statement tells the program where to go if it comes to the end of file #1. Without this statement, the program would quit at the end of the file and give an error message.

500 IF TYP(1)=1 THEN 600 }

550 IF TYP(1)=2 THEN 700 }

TYP checks whether the next data item is a number (1) or a string (2).

600 READ #1;A

Reads a number from file #1 into variable A.

650 PRINT A

675 GOTO 500

700 READ #1;A\$

Reads a string from file #1 into variable A\$.

750 PRINT A\$

775 GOTO 500

1000 PRINT "FILE LIST COMPLETED"

The program comes here when it reaches the end of file #1.

2000 END

# THE TYP FUNCTION

EXAMPLES:           100 IF TYP(1)=2 THEN 1000  
                      250 IF TYP (4)=3 THEN 500  
                      300 GO TO TYP(B) of 400,600,800

GENERAL FORM:       TYP may be used as an expression or as  
                      part of an expression; the function form is:

TYP (*file number formula*)

## PURPOSE

Determines the type of the next data item in the specified file so that the program can avoid a type mismatch on a file READ.

There are three possible responses:

- 1 = next item is number
- 2 = next item is character string
- 3 = next item is "end of file."

## COMMENTS

If the file number formula is negated (<0), the TYP function also detects "end of record" conditions (explained later under "Random Access") and returns a value of 4 for them.

If the file number formula equals zero, the TYP function references the DATA statement. (See "the TYP Function" in Section III.)

## TERM: END-OF-FILE

If a program attempts to PRINT beyond the physical end of a file or attempts to READ more values than are present in the file, the TSB system detects an end-of-file condition and terminates the program.

### COMMENTS

The OPEN command causes end-of-file marks to be written at the start of every record in the file. End-of-file marks can also be written by the user (as explained later under "END").

*NOTE: If the user or an error (such as end-of-file) stops a program abnormally, it is not possible to know which file PRINTs of the program were in fact performed.*

To avoid termination of a program because of end-of-file, use the IF END statement on the next page. If this is done, all of the values preceding the end-of-file are transferred successfully.

# IF END#...THEN

EXAMPLES:            300 IF END #N THEN 800  
                      310 IF END #2 THEN 830  
                      320 IF END #3 THEN 9999

GENERAL FORM:

statement number IF END #file number formula THEN statement number

## PURPOSE

Defines a statement to be branched to if an "end-of-file" occurs on a specified file.

## COMMENTS

The IF END statement defines an exit procedure which remains in effect until another IF END for the same file changes it.

A different exit procedure can be defined for each file.

IF END is also used with random access to provide exit procedures when an "end-of-record" occurs. (See "Random Access.")

If a program does not contain an IF END statement for a file and an "end-of-file" occurs on that file, the program is terminated and an error message is printed:

END OF FILE/END OF RECORD IN STATEMENT xxx

# PRINT #...END

EXAMPLES:            95 PRINT #N: A,B2,END  
                      100 PRINT #(X+1); R3,S1,N\$, "TEXT" , END  
                      110 PRINT #2; G5,H\$,P, END

GENERAL FORM:

statement number PRINT #file number formula ; data item list , END

## PURPOSE

Places a logical "end-of-file" marker after the last value written on the file; END is ignored if it is not the last item in the statement.

## COMMENTS

The "end-of-file" marker written by this statement is a logical marker; each file also has a physical end-of-file which marks the physical boundary of the file.

The "end-of-file" mark is overlaid by the first item in the next serial PRINT statement. An "end-of-file" condition that aborts the program or triggers an IF END statement occurs only on an attempted READ beyond the available data or an attempted PRINT beyond the physical end-of-file.

END and IF END can be used to modify a serial file.

## MODIFYING A SERIAL FILE

PRINCIPLE: Serial files can only be lengthened (added to) or shortened.

Items within a serial file cannot be written over without destroying the succeeding items in the file.

RESULT: An item within a serial file can be updated only if you read it and all succeeding items into the program, then correct the one item and write them all back out again.

### ADDING TO A SERIAL FILE

You may often want to keep a serial file for data collection. The file would contain a list of data items that would be added to continuously over a period of time until the file is full.

IF END and PRINT ... END are used to write a program that creates and lengthens such a file.

## EXAMPLE OF SERIAL FILE MODIFICATION

OPEN-DATUMS, 48

When the file is opened, "end-of-file" markers are written into every record.

NAM-ADDIT

```
100  FILES DATUMS
200  DIM A$[72]
300  IF END #1 THEN 1500
400  REM THIS PROGRAM FIRST FINDS THE END OF THE FILE.  IT ASKS THE
410  REM USER FOR A STRING AND A NUMBER.  IF THIS IS NOT THE PHYSICAL
420  REM END OF THE FILE, IT ADDS THEM TO THE END OF THE FILE.
430  REM THEN THE PROGRAM ASKS THE USER IF HE WANTS TO ADD ANY MORE ITEMS.
440  REM IF THE USER ANSWERS YES, THE PROGRAM REPEATS THE INPUT AND
450  REM WRITE LOOP.
800  READ #1;A$,A
850  GOTO 800
1500 IF END #1 THEN 2000
1600 PRINT "STRING";
1650 INPUT A$
1700 PRINT "NUMBER";
1750 INPUT A
1800 PRINT #1;A$,A, END
1900 PRINT "MORE";
1950 INPUT A$
1960 IF A$="YES" THEN 1600
1970 STOP
2000 PRINT "PHYSICAL END OF THIS FILE"
5000 END
```

*NOTE: If the file is empty, the first thing the program finds is an end-of-file. Therefore, it begins filling the file from the first location.*

## EXAMPLE, CONTINUED

The IF END statement (line 300) is changed once the end-of-file marker is found. The program is then looking for the physical end-of-file.

You can use the listing sample program to check the contents of the file.



# STRUCTURE OF SERIAL FILES, CONTINUED

↓

To read this data, the pointer must be reset.

INFO = 

A	B	C	F\$	Q1	G1	G2	G\$	H\$	Z	EOF		PEOF
---	---	---	-----	----	----	----	-----	-----	---	-----	--	------

↑

400 READ #1,1

↓

Now the data can be read.

INFO = 

A	B	C	F\$	Q1	G1	G2	G\$	H\$	Z	EOF		PEOF
---	---	---	-----	----	----	----	-----	-----	---	-----	--	------

↑

500 READ #1; M1,M2

M1 now contains the value of A

M2 now contains the value of B

↓

At this point, the program continues to read the data.

INFO = 

A	B	C	F\$	Q1	G1	G2	G\$	H\$	Z	EOF		PEOF
---	---	---	-----	----	----	----	-----	-----	---	-----	--	------

↑

600 READ #1; D1

D1 now contains the value of C

# STRUCTURE OF SERIAL FILES, CONTINUED

However, if you PRINT anything in the file at this point, the rest of the file is effectively lost as far as serial access is concerned.

INFO = 

A	B	C	D2	EOF		PEOF
---	---	---	----	-----	--	------

↑

700 PRINT #1; D2,END

↓

The correct way to modify an item in the middle of serial file is to READ all the succeeding items, then PRINT them and the new value out again.

INFO = 

A	B	C	F\$	Q1	G1	G2	G\$	H\$	Z	EOF		PEOF
---	---	---	-----	----	----	----	-----	-----	---	-----	--	------

↑

700 READ #1; M\$, P1, P2, P3, P\$, R\$, P4  
(READ the values)

750 READ #1,1 (reset the pointer)

800 READ #1; A, B, C  
(move the pointer out to the correct item)

900 PRINT #1; D2 (PRINT the new item)

1000 PRINT #1; P1, P2, P3, P\$, R\$, P4, END  
(PRINT the old values out)

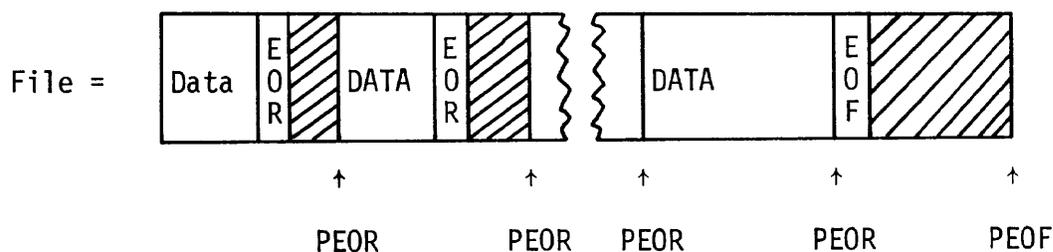
A	B	C	D2	Q1	G1	G2	G\$	H\$	Z	EOF		PEOF
---	---	---	----	----	----	----	-----	-----	---	-----	--	------

↑

# TERM: RECORD

DEFINED IN TSB AS: A physical division of a file; consists of 128 memory words that can be used to store numbers or strings of characters. Every file consists of from 1 to 48 records.

## COMMENTS



where PEOR = the physical end of the record.

EOR = the end-of-record marker written by the system.

EOF = the end-of-file marker written by the system.

PEOF = the physical end of the file.

Following the data in a record, there is always an end-of-record marker. Every record also has a physical end. (When the record is completely full, this also acts as the logical end-of-record marker.)

During serial access the end-of-record markers act as skip markers that say to look in the next record for the data item, but during random access they cause an end-of-file condition. This will be explained later.

## STORAGE REQUIREMENTS

Numerical data items require two words of storage space per item. Therefore, if a record is filled completely with numbers, it contains 64 items.

Strings can be of varying sizes: they require about 1/2 word of storage per character in the string. The exact formula for the number of words needed to store a string is:

If the number of characters is odd, then

$$1 + \left( \frac{\text{number of characters in the string} + 1}{2} \right)$$

If the number of characters is even, then

$$1 + \left( \frac{\text{number of characters in the string}}{2} \right)$$

Therefore, four 62-character strings would completely fill a record.

Strings and numbers can be mixed within a record, but each item must fit completely within the bounds of the record. For example, a record could contain a string of 72 characters (using 37 words) and a maximum of 45 numbers (leaving one word of the record unused).

# MOVING THE POINTER

EXAMPLES:           200 READ #1,N  
                      300 READ #M,N  
                      400 READ #3\*J,9

GENERAL FORM:

statement number READ #file number formula , record number formula

## PURPOSE

Moves the pointer to the beginning of a specified record within a file; rounds the file number formula and the record number formula to integers.

## COMMENTS

The READ #M,N statement only generates an end-of-file condition at the physical end of the file, not for end-of-file markers.

After moving the pointer to the start of a record, you can use the serial READ and PRINT statements normally.

# SAMPLE USE OF READ# M,N

## DETERMINE LENGTH OF A FILE

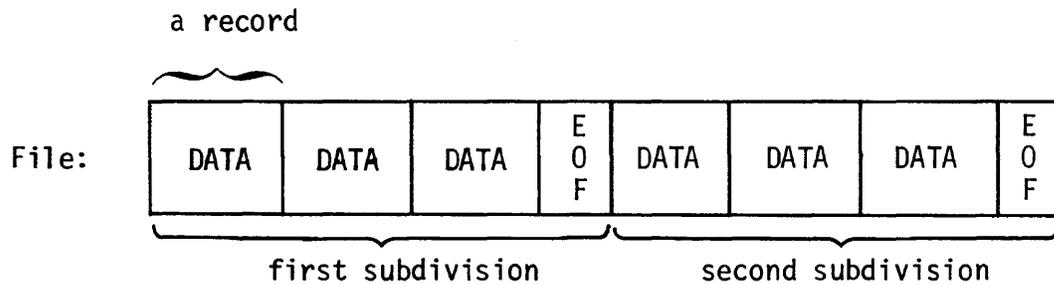
Here is a sample program that determines the number of records in a file. It uses the READ #M,N statement through the records until it comes to the physical end of the file.

NAM-LENGTH

```
10 REM THIS PROGRAM PRINTS OUT THE LENGTH IN RECORDS OF ANY FILE.  
20 FILES M  
30 REM M IS THE FILE WHOSE LENGTH IS SOUGHT  
40 IF END #1 THEN 80  
50 FOR R=1 TO 128  
60 READ #1,R  
70 NEXT R  
80 PRINT "LENGTH IN RECORDS: ";R-1  
90 END
```

# SUBDIVIDING SERIAL FILES

Serial files can be divided into smaller serial files by moving the pointer and using the PRINT END statement. For example, a file of six records could be treated as two files of three records.



To switch from the first subdivision to the second, use this statement

```
1000 READ #1, 4
```

since the fourth record is the start of the second subdivision.

When using this technique, you must be careful that you do not PRINT more data into the subdivision than it will hold. If you PRINT too much, the data will overflow into the next subdivision and destroy its contents.

A logical extension of this concept is to make each subdivision equal to a single record. The TYP function detects end-of-record markers. The random access versions of PRINT# and READ# (described later) allow you to access random records within a file without overflowing the bounds of the record.

# USING THE TYP FUNCTION WITH RECORDS

EXAMPLES:           1000 GO TO TYP(-1) OF 2000, 2500, 3000, 4000  
                  20000 A=TYP(-4) + B\*2

GENERAL FORM:       TYP is a function and can be used as an  
                  expression or a part of an expression.

TYP (-file number formula)

## PURPOSE

Returns a code telling the type of the next item in a specified file.

TYP(- X) = 1 for a number  
          2 for a string  
          3 for an end-of-file  
          4 for an end-of-record

## COMMENTS

The file number formula must be negated to detect the end of record. If it is positive or zero, different results are returned. See TYP Function in this section and Section III.

# SAMPLE OF READ # M,N AND TYP(-M)

## LIST CONTENTS OF A RECORD

Here is a sample program that lists the exact contents of any record in a file.

NAM-RLIST

```
1   REM THIS PROGRAM LISTS THE CONTENTS OF ANY RECORD OF THE FILE.
5   DIM A$[72]
10  FILES PETER
20  IF END #1 THEN 60
30  PRINT "RECORD NUMBER";
40  INPUT R
50  IF R>0 AND R=INT(R) THEN 80
60  PRINT "INVALID RECORD NUMBER."
70  GOTO 30
80  READ #1,R
100 GOTO TYP(-1) OF 110,150,220,200
110 PRINT "NUMBER:";
120 READ #1;X
130 PRINT X
140 GOTO 100
150 PRINT "STRING:";
160 READ #1;A$
170 PRINT A$
180 GOTO 100
200 PRINT "END OF RECORD MARK."
210 STOP
220 PRINT "END OF FILE MARK."
230 END
```

## HOW TO COPY A FILE

Here is a sample program that copies one file into another using only the statements and functions covered so far: IF END, TYP, FILES, READ #M,N, serial READ, and serial PRINT.

### NAM-COPY

```
1   REM THIS PROGRAM COPIES FILE #1 INTO FILE #2
10  FILES SAM1, SAM2
20  DIM A$(72)
30  IF END #1 THEN 170
40  IF END #2 THEN 180
50  FOR I=1 TO 128
60  READ #1,I
70  PRINT #2,I
80  GOTO TYP(-1) OF 90,120,150,160
90  READ #1;X
100 PRINT #2;X
110 GOTO 80
120 READ #1;A$
130 PRINT #2;A$
140 GOTO 80
150 PRINT #2; END
160 NEXT I
170 STOP
180 PRINT "SECOND FILE TOO SMALL"
190 END
```

## TERM: RANDOM FILE ACCESS

DEFINED IN TSB AS: A READ or PRINT access of a file is "random" if it specifies a particular record within the file.

Serial Access: 100 READ #1;A,B,C  
(Reads from the file pointer)

Random Access: 100 READ #1,5;A,B,C  
(Moves to record 5 before reading)

### COMMENTS

When files are accessed serially, the record structure of files is ignored. Serial READs skip over end-of-record markers to the next record and act as if all data were in a continuous list.

The TSB System does, however, provide statements that take advantage of this record structure. The file pointer can be moved to the beginning of any record. Also, any record can be READ or PRINTed independently of the rest of the file using random access versions of READ# and PRINT#. The TYP function and IF END statement can detect end-of-record conditions. These extensions to BASIC constitute a random access file capability.

# SAMPLE OF RANDOM FILE ACCESS

This sample program fills each record with two strings of up to 30 characters each and five numbers. Then it lists the contents of any record.

```
OPEN-RNDFL,20
```

```
NAM-PROG2
```

```
100  FILES RNDFL
150  DIM A$(30),B$(30)
200  IF END #1 THEN 1000
300  FOR J=1 TO 20
400  INPUT A$,B$,A,B,C,D,E
500  PRINT #1,J; A$,B$,A,B,C,D,E
600  NEXT J
700  PRINT "WHICH RECORD WOULD YOU LIKE TO SEE";
750  INPUT J
760  READ #1, J; A$,B$,A,B,C,D,E
770  PRINT A$
780  PRINT B$
790  PRINT A,B,C,D,E
800  GO TO 700
1000 END
```

This loop reads in two strings and five numbers from the user, then it writes the Jth record of the file.

This section will read and list the contents of record N.

# PRINTING A RECORD

EXAMPLES:           165 PRINT #N,X;G2,H,I,"TEXT"  
                      170 PRINT #1,3;X,Y4,Z,6127,B  
                      175 PRINT #(N+2),(X+2);F,P5  
                      180 PRINT #2,5;A,B,C,D,END

## GENERAL FORM:

statement number PRINT #file number formula ,  
record number formula ; list of data items

## PURPOSE

Prints a specified list of data items into a specific record of a file, starting at the beginning of the record. (The record number formula is rounded to the nearest integer.)

## COMMENTS

The previous contents of the record are destroyed. An end-of-record marker is written after the data. If an END occurs in the data list, it acts as an end-of-record marker too. The random PRINT cannot change the contents of any record except the one specified. The entire list of data items must fit within the 128-word record. Otherwise, an end-of-file condition occurs which terminates the program and prints an error message:

END OF FILE/END OF RECORD

An IF END statement establishes an exit procedure. See "IF END" in this section.

## PRINTING A RECORD, CONTINUED

Matrices are PRINTed using the random version of MAT PRINT# described in Section V. Note, however, that the matrix must fit within a single record. Therefore, a maximum of 64 numerical items can be PRINTed. If this rule is violated, an end-of-file occurs.

# READING A RECORD

EXAMPLES:           100 READ #2,3;A,B,C3,X\$  
                      110 READ #N,2;N1,N2,N3  
                      120 READ #M,N;R2,P7,A\$,T(35)  
                      130 READ #(M+1),(N+1);X,Y,Z

GENERAL FORM:

statement number READ #file number formula ,  
record number formula ; list of data items

## PURPOSE

Reads data from a specified record of a file, starting at the beginning of the record. (The file number formula and record number formula are rounded to integers.)

## COMMENTS

The contents of the file are not changed.

If the READ encounters an end-of-record marker before filling all the data items, an end-of-file occurs. The program is terminated unless an IF END statement has been defined previously. (See IF END in this section.)

Matrices are READ from records using a random version of MAT READ# described in Section V. Note however, that only 64 numbers can be stored in a record. If the READ requests more than 64 numbers (or more items than the record contains), an end-of-file condition occurs.

# MODIFYING CONTENTS OF A RECORD

PRINCIPLE: The contents of a record can be changed only by READING the entire record into the program, modifying the items desired, then PRINTing it back on the file again.

EXAMPLE: 100 READ #1,5;A,B,C,Z\$  
200 LET A = Q\*2+(M/5)  
300 LET Z\$ = M\$  
500 PRINT #1,5;A,B,C,Z\$

A,B,C, and Z\$ are the entire contents of record 5,

DANGER: When the strings are replaced by longer strings, the result may no longer fit within a record. If this happens, an end-of-file condition occurs.

# ERASING A RECORD

EXAMPLES:        32Ø PRINT #M+N, R+S  
                  33Ø PRINT #1,2  
                  34Ø PRINT #4,Q1

GENERAL FORM:

statement number PRINT #file number formula , record number formula

## PURPOSE

Erases the contents of a specified record in a file by PRINTing an end-of-record marker at the beginning of the record.

Moves the file pointer to the start of the specified record.

## COMMENTS

Only the contents of the specified record are erased; the rest of the file is unchanged. The erased record still exists, however, and can be filled with new data.

Do not confuse this erase operation with the KILL command which permanently eliminates the entire file.

Here is a sample program that uses the erase operation to erase an entire file, record by record.

## ERASING A RECORD, CONTINUED

NAM-ERASE

```
1  REM THIS PROGRAM ERASES A FILE BY ERASING EVERY RECORD
5  DIM A$(72)
10 FILES X
20 IF END #1 THEN 60
30 FOR I=1 TO 128
40 PRINT #1,I
50 NEXT I
60 END
```

## UPDATING A RECORD IN A FILE

File programming is simplified if every record of a file has the same data structure. For example, each record might contain a string (e.g., a person's name) and a number (e.g., the amount of money he owes). Here is a sample program that manipulates such a file. The program searches through the file until it finds a specified string; then it updates the number in the record to a new value.

```
NAM-UPDATE  
10  FILES DATA  
20  DIM A$(72) , B$(72)  
30  IF END #1 THEN 160  
40  PRINT "NAME";  
50  INPUT A$  
60  FOR I= 1 to 128  
70  READ #1, I  
80  IF TYP (-1) #2 THEN 150  
90  READ #1; B$  
100 IF B$#A$ THEN 150  
110 PRINT "NEW NUMBER";  
120 INPUT N  
130 PRINT #1; N  
140 STOP  
150 NEXT I  
160 PRINT "NAME NOT ON FILE."  
170 END
```

## AN ALPHABETICALLY ORGANIZED FILE

If the first item of every record in a file is a string, the records can be ordered alphabetically. Here is a program that inserts a new record where it alphabetically belongs. The rest of the file must be moved up one record. In this example, record 1 contains the record number of the last item.

### NAM-INSERT

```
10  FILES DATA
20  DIM G$(72), H$(72)
30  IF END #1 THEN 290
40  READ #1,1;N
50  READ #1, N+1
60  PRINT "STRING";
70  INPUT G$
72  IF N#0 THEN 80
74  R=2
76  GO TO 180
80  F= 1
90  L= N+1
100 R= INT ((F+L)/2)
110 READ #1,R; H$
120 IF G$<H$ THEN 210
130 IF G$>H$ THEN 230
140 FOR I= N TO R STEP -1
150 READ #1,I; H$
160 PRINT #1,I+1; H$
170 NEXT I
180 PRINT #1,R; G$
190 PRINT #1,1; N+1
200 STOP
210 L=R
220 IF F#L THEN 100
225 GO TO 140
230 F= R
240 IF L-F > 1 THEN 100
250 R=R+1
260 GO TO 140
270 PRINT "FILE FULL."
280 STOP
290 N=0
300 GO TO 50
310 END
```

## SECTION V MATRICES

This section explains matrix manipulation. In TSB, you can treat a matrix simply as an array of data organized into rows and columns or as a mathematical entity. Using data arrays requires no special background; the necessary statements are covered in this section. Using the matrix as a mathematical entity, as in matrix inversion, transposition, etc., requires a background in matrix theory.

### TERM: MATRIX (ARRAY)

DEFINED IN TSB AS:	An ordered collection of numeric data containing not more than 2000 elements (numeric values).
--------------------	--

Matrix elements are referenced by subscripts following the matrix variable, indicating the row and column of the element. For example, if matrix A is:

1	2	3
4	5	6
7	8	9

the element 5 is referenced by A(2,2); likewise 9 is A(3,3).

See Section III, "Vocabulary" for a more complete description of matrices.

# DIM

EXAMPLES:                   110 DIM A (50), B(20,20)  
                              120 DIM Z (5,20)  
                              130 DIM S (5,25)  
                              140 DIM R (4,4)

GENERAL FORM:

statement number DIM matrix variable ( integer ) ...

or

statement number DIM matrix variable ( integer , integer ) ...

## PURPOSE

Sets upper limits on the amount of working space used by a matrix in the TBS system.

## COMMENTS

The integers refer to the number of matrix elements if only one dimension is supplied, or to the number of column and row elements respectively, if two dimensions are given.

A matrix (array) variable is any single letter from A to Z.

Arrays not mentioned in a DIM statement are assumed to have 10 elements if one-dimensional, or 10 rows and columns if two-dimensional.

The working size of a matrix may be smaller than its physical size. For example, an array declared 9 x 9 in a DIM statement may be used to store fewer than 81 elements; the DIM statement supplies only an upper bound on the number of elements.

The absolute maximum matrix size is 2000 elements; a matrix of this size is practical only in conjunction with a very small program.

# MAT...ZER

## EXAMPLES:

305 MAT A = ZER

310 MAT Z = ZER (N)

315 MAT X = ZER (30, 10)

320 MAT R = ZER (N, P)

## GENERAL FORM:

statement number MAT matrix variable = ZER

or

statement number MAT matrix variable = ZER ( expression )

or

statement number MAT matrix variable = ZER ( expression , expression )

## PURPOSE

Sets all elements of the specified matrix equal to 0; a new working size may be established.

## COMMENTS

The new working size in a MAT...ZER is an implicit DIM statement within the limits set by the DIM statement on the total number of elements.

Since 0 has a logical value of "false", MAT...ZER is useful in logical initialization.

The expressions in new size specifications should evaluate to integers. Non-integers are rounded to the nearest integer value.



# INPUT

EXAMPLES:           600 INPUT A(5)  
                      610 INPUT B(5,8)  
                      620 INPUT R(X), N\$, A(3,3)  
                      630 INPUT Z(X,Y), P3, W\$  
                      640 INPUT Z(X,Y), Z(X+1, Y+1), Z(X+R3, Y+S2)

GENERAL FORM:

statement number INPUT matrix variable ( expression ) ...

or

statement number INPUT matrix variable ( expression , expression ) ...

## PURPOSE

Allows input of a specified matrix element(s)  
from the teleprinter.

## COMMENTS

Expression should evaluate to integers. Non-integers are rounded to the nearest integer value.

The subscripts (expressions) used after the matrix variable designate the row and column of the matrix element. Do not confuse these expressions with working size specifications, such as those following a MAT INPUT statement.

See MAT INPUT and DIM in this section for further details on matrix input.

# MAT INPUT

## EXAMPLES:

```
355 MAT INPUT A
360 MAT INPUT B(5)
365 MAT INPUT Z(5,5)
370 MAT INPUT A(N)
375 MAT INPUT B(N,M)
```

## GENERAL FORM:

statement number MAT INPUT matrix variable

or

statement number MAT INPUT matrix variable ( expression )...

or

statement number MAT INPUT matrix variable ( expression , expression )...

## PURPOSE

Allows input of an entire matrix from the teleprinter; a new working size may be specified, within the limits of the DIM statement on total number of elements.

## COMMENTS

Do not confuse the size specifications following MAT INPUT with element specifications. For example, INPUT X(5,5) causes the fifth element of the fifth row of matrix X to be input, while MAT INPUT X(5,5) requires input of the entire matrix called X, and sets the working size at 5 rows of 5 columns.

Example statements 360 through 375 require input of the specified number of matrix elements, because they specify a new size.

Elements being input must be separated by commas.

A "??" response to an input item means that more input is required.

Only one ? is generated by a MAT INPUT statement, regardless of the number of elements.

MAT INPUT causes the entire matrix to be filled from teleprinter input in the (row, col.) order: 1,1;1,2;1,3; etc.

# PRINTING MATRICES

EXAMPLES:           800 PRINT A(3)  
                      810 PRINT A(3,3);  
                      820 PRINT F(X);E\$; C5;R(N)  
                      830 PRINT G(X,Y)  
                      840 PRINT Z(X,Y), Z(1,5), Z(X+N, Y+M)

GENERAL FORM:

statement number PRINT matrix variable ( expression ) ...

or

statement number PRINT matrix variable ( expression , expression ) ...

## PURPOSE

Causes the specified matrix element(s) to be printed.

## COMMENTS

Expressions (subscripts) should evaluate to integers. Non-integers are rounded to the nearest integer value.

A trailing semicolon packs output into twelve elements per teleprinter line, if possible. A trailing comma prints five elements per line.

Expressions (subscripts) following the matrix variable designate the row and column of the matrix element. Do not confuse these with new working size specifications, such as those following a MAT INPUT statement.

This statement prints a single matrix element. MAT PRINT is used to print an entire matrix.

# MAT PRINT

EXAMPLES:           500 MAT PRINT A  
                      505 MAT PRINT A;  
                      515 MAT PRINT A,B,C  
                      520 MAT PRINT A,B,C;

GENERAL FORM:

statement number MAT PRINT matrix variable  
                                  or  
statement number MAT PRINT matrix variable , matrix variable ...

## PURPOSE

Causes an entire matrix to be printed, row by row, with double spacing between rows.

## COMMENTS

Matrices may be printed in "packed" rows up to 12 elements wide by using the ";" separator, as in example statement 505. Normal separation (",") prints 5 elements per row.

# READ

EXAMPLES:           900 READ A(6)  
                      910 READ A(9,9)  
                      920 READ C(X); P\$; R7  
                      930 READ C(X,Y)  
                      940 READ Z(X,Y), P(R2, S5), X(4)

GENERAL FORM:

statement number READ matrix variable ( expression )

or

statement number READ matrix variable ( expression , expression ) ...

## PURPOSE

Causes the specified matrix element to be read from the current DATA statement.

## COMMENTS

Expressions (subscripts) should evaluate to integers. Non-integers are rounded to the nearest integer.

Expressions following the matrix variable designate the row and column of the matrix element. Do not confuse these with working size specifications, such as those following MAT INPUT statement.

The MAT READ statement is used to read an entire matrix from DATA statements. See details in this section.

# MAT READ

EXAMPLES:            35Ø MAT READ A  
                      37Ø MAT READ B(5),C,D  
                      38Ø MAT READ Z (5,8)  
                      39Ø MAT READ N (P3,Q7)

GENERAL FORM:

statement number MAT READ matrix variable

or

statement number MAT READ matrix variable ( expression ) ...

or

statement number MAT READ matrix variable ( expression , expression )

## PURPOSE

Reads an entire matrix from DATA statements.  
A new working size may be specified, within  
the limits of the original DIM statement.

## COMMENTS

MAT READ causes the entire matrix to be filled  
from the current DATA statement in the (row, col.)  
order: 1,1; 1,2; 1,3; etc. In this case the  
DIM statement controls the number of elements  
read.

# MATRIX ADDITION

EXAMPLES:

31Ø MAT C = B + A

32Ø MAT X = X + Y

33Ø MAT P = N + M

GENERAL FORM:

statement number MAT matrix variable = matrix variable + matrix variable

## PURPOSE

Establishes a matrix equal to the sum of two matrices of identical dimensions; addition is element-by-element.

## COMMENTS

The resulting matrix must be previously mentioned in a DIM statement, if it has more than 10 elements, or 10 x 10 elements if two dimensional. Dimensions must be the same as the component matrices.

The same matrix may appear on both sides of the = sign, as in example statement 32Ø.

# MATRIX SUBTRACTION

EXAMPLES:

55Ø MAT C = A - B

56Ø MAT B = B - Z

57Ø MAT X = X - A

GENERAL FORM:

statement number MAT matrix variable = matrix variable - matrix variable

## PURPOSE

Establishes a matrix equal to the difference of two matrices of identical dimensions; subtraction is element-by-element.

## COMMENTS

The resulting matrix must be previously mentioned in a DIM statement if it has more than 10 elements, or 10 x 10 elements if two dimensional. Its dimension must be the same as the component matrices.

The same matrix may appear on both sides of the = sign, as in example statement 56Ø.

# MATRIX MULTIPLICATION

EXAMPLES:

93Ø MAT Z = B \* C

94Ø MAT X = A \* A

95Ø MAT C = Z \* B

GENERAL FORM:

statement number MAT matrix variable = matrix variable \* matrix variable

## PURPOSE

Establishes a matrix equal to the product of the two specified matrices.

## COMMENTS

Following the rules of matrix multiplication, if the dimensions of matrix B = (P,N) and matrix C = (N,Q), multiplying B\*C results in a matrix of dimensions (P,Q).

Note that the resulting matrix must have an appropriate working size.

The same matrix variable may not appear on both sides of the = sign.

# SCALAR MULTIPLICATION

EXAMPLES:

110 MAT A = (5) \* B

115 MAT C = (10) \* C

120 MAT C = (N/3) \* X

130 MAT P = (Q7\*N5) \* R

GENERAL FORM:

statement number MAT matrix variable = ( expression ) \* matrix variable

## PURPOSE

Establishes a matrix equal to the product of a matrix multiplied by a specified number, that is, each element of the original matrix is multiplied by the number.

## COMMENTS

The resulting matrix must be previously mentioned in a DIM statement, if it contains more than 10 elements (10x10 if two dimensional).

The same matrix variable may appear on both sides of the = sign.

Both matrices must have the same working size.

# COPYING A MATRIX

EXAMPLES:            405 MAT B = A  
                      410 MAT X = Y  
                      420 MAT Z = B

GENERAL FORM:

statement number MAT matrix variable = matrix variable

## PURPOSE

Copies a specified matrix into a matrix of the same dimensions; copying is element-by-element.

## COMMENTS

The resulting matrix must be previously mentioned in a DIM statement if it has more than 10 elements, or 10x10 if two dimensional. It must have the same dimensions as the copied matrix.

# IDENTITY MATRIX

## EXAMPLES:

205 MAT A = IDN

210 MAT B = IDN (3,3)

215 MAT Z = IDN (Q5, Q5)

220 MAT S = IDN (6, 6)

## GENERAL FORM:

statement number MAT array variable = IDN

or

statement number MAT array variable = IDN ( expression , expression )

## PURPOSE

Establishes an identity matrix (all 0's, with a diagonal of all 1's): a new working size may be specified.

## COMMENTS

The IDN matrix must be two dimensional and square.

Specifying a new working size has the effect of a DIM statement.

Sample identity matrix:

1	0	0
0	1	0
0	0	1



# MATRIX INVERSION

EXAMPLES:            380 MAT A = INV(B)  
                      390 MAT C = INV(A)  
                      400 MAT Z = INV(Z)

GENERAL FORM:

statement number MAT matrix variable = INV ( matrix variable )

## PURPOSE

Establishes a square matrix as the inverse of the specified square matrix of the same dimensions.

## COMMENTS

A matrix may be inverted into itself, as in example statement 400, above.

Number representation in TSB is accurate to 6-7 decimal digits; matrix elements are rounded accordingly.

# MAT PRINT #

EXAMPLES:                   52Ø MAT PRINT #4; A  
                              53Ø MAT PRINT #3, 3; B  
                              54Ø MAT PRINT #4,M; A  
                              55Ø MAT PRINT #N,M; A

## GENERAL FORM:

statement number MAT PRINT# file number formula ; matrix variable ...

or

stat. no. MAT PRINT# file no. form. , record no. form.; matrix var. ...

## PURPOSE

Prints an entire matrix on a file, or on a specified record within a file.

## COMMENTS

A random matrix file print (i.e., with a record number specified) cannot transfer more than 64 numeric values because that is the maximum a record can hold. This generates an end-of-file condition.

A serial matrix file print, however, can transfer as many elements as will fit in the entire file.

# MAT READ #

## EXAMPLES:

72Ø MAT READ #2;A  
73Ø MAT READ #2,3;B  
74Ø MAT READ #M,N;B(5)  
75Ø MAT READ #M,N;B(P7,R5)

## GENERAL FORM:

statement number MAT READ# file formula number ; matrix variable ...

or

statement no. MAT READ# file formula no. , record no. formula ; matrix variable...

or

statement no. MAT READ# file form. no. , record no. form. ; matrix var. ( expression )...

or

stmt. no. MAT READ# file form. no. , record no. form. ; matrix var. ( expr. , expr. )...

## PURPOSE

Reads a matrix from a file, or specified record within a file. A new working size may be specified.

## COMMENTS

MAT READ# fills the entire matrix in a row-by-row sequence of elements as: 1,1; 1,2; 1,3; 1,4 ...

Remember that a maximum of 64 numbers may be transferred on a random read.

## SECTION VI

### STRINGS

A string is a set of characters such as "DDDDDE" or "45T,#". BASIC contains special variables and language elements for manipulating string quantities. This section explains how to use the string features of BASIC. There is little difference in the form of statements referencing numeric quantities and those referencing strings. One important difference, however, is the use of subscripts which is explained later.

The examples and comments in this section emphasize modifications in statement form or other special considerations in handling strings.

If you are familiar with the concepts "string," "string variable," and "substring," skip directly to "The String DIM Statement."

# TERM: STRING

DEFINED IN TSB AS:

A set of 1 to 72 characters enclosed by quotation marks or the null string (no characters).

## COMMENTS

Typical  
Strings: "ABCDEFGHIJKLMNOP"  
          "12345"  
          "BOB AND TOM"  
          "MARCH 13, 1970"

Special purpose characters such as  $x^c$  and quotation marks cannot be used within a string.

Apostrophes and most control characters are legal as string characters.

Strings are manipulated in string variables.  
For example:

100 A\$ = "THIS IS A STRING"  
          ↑                  ↑  
          string              string  
          variable

200 B\$ = A\$(1,10)  
          ↑          ↑  
          string      substring  
          variable   (defined later)

## TERM: STRING VARIABLE

DEFINED IN TSB AS: A variable used to store strings; consists of a single letter (A to Z) followed by a \$.

For example: A\$, Z\$, M\$

### COMMENTS

String variables must be declared before being used if they contain strings longer than one character. See "The String DIM Statement."

When a string variable is declared, its "physical" length is set. The "physical" length is the maximum size string that the variable can accommodate. For example:

```
71Ø DIM A$(72),B$(2Ø),C$(5)
```

During execution of a program, the "logical" length of a string variable varies. The "logical" length of the variable is the actual number of characters that the string variable contains at any point. For example:

1ØØ DIM A\$(72)	(Sets physical length of A\$)
2ØØ A\$ = "SAMPLE STRING"	(Logical length of A\$ is 13)
3ØØ A\$="LONGER SAMPLE STRING"	(Logical length of A\$ is now 20)

# TERM: SUBSTRING

DEFINED IN TSB AS: A single character or a set of contiguous characters within a string. The substring is defined by a subscripted string variable.

## COMMENTS

A substring is defined by subscripts placed after the string variable. Characters within a string are numbered from the left starting with one.

Two subscripts specify the first and last characters of the substring. For example:

```
100 Z$ = "ABCDEFGH"
```

```
200 PRINT Z$(2,6)
```

prints the substring

```
BCDEF
```

A single subscript specifies the first character of the substring and implies that all characters following are part of the substring. For example:

```
300 PRINT Z$(3)
```

prints the substring

```
CDEFGH
```

## TERM: SUBSTRING

Two equal subscripts specify a single character substring. For example:

```
400 PRINT Z$(2,2)
```

prints the substring

B

# STRINGS AND SUBSTRINGS

A string variable can be assigned the null string (no value, as distinguished from a blank space which has a value). This can be done by assigning it the value of a substring whose second subscript is one less than its first. For example:

```
100 A$= B$(6,5)  (A$ now contains a
                  null string)
```

This is the only case in which a smaller second subscript is acceptable in a substring.

Substrings can become strings. For example:

```
100 A$ = "ABCDEFGH"
```

```
200 B$ = A$(3,5)
```

```
300 PRINT B$
```

prints the string

```
CDE
```

because the substring CDE of A\$ is now the string of B\$.

Substrings can be assigned to subscripted string variables to change characters within a larger string. For example:

```
100 A$ = "ABCDEFGH"
```

```
200 A$(3,5) = "123"
```

```
300 PRINT A$
```

## STRINGS AND SUBSTRINGS, CONTINUED

prints the string

AB123FGH

Strings, substrings, and string variables can be used with relational operators. They are compared and ordered as entries are in a dictionary. For example:

100 IF A\$ = B\$ THEN 2000

200 IF A\$ ≤ "TEST" THEN 3000

3000 IF A\$(5,6) ≥ B\$(7,8) THEN 4000

# THE STRING DIM STATEMENT

EXAMPLES:                    35 DIM A\$ (72), B\$(60)  
                              40 DIM Z\$ (10)  
                              45 DIM N\$ (2), R(5,5), P\$(8)

GENERAL FORM :

statement number DIM string variable ( number of characters in string )

## PURPOSE

Reserves storage space for strings longer than 1 character; also for matrices (arrays).

## COMMENTS

The number of characters specified for a string in its DIM statement must be expressed as an integer from 1 to 72.

Each string having more than 1 character must be mentioned in a DIM statement before it is used in the program.

Strings not mentioned in a DIM statement are assumed to have a length of 1 character.

The length mentioned in the DIM statement specifies the maximum number of characters which may be assigned; the actual number of characters assigned may be smaller than this number. See "The LEN Function" in this section for further details.

Matrix dimension specifications may be used in the same DIM statement as string dimensions (example statement 45 above).

# THE STRING ASSIGNMENT STATEMENT

*NOTE: These strings have been mentioned in a DIM statement*

EXAMPLES:                   200 LET A\$ = "TEXT OF STRING"  
                              210 B\$ = "\*\*\* TEXT !!!"  
                              220 LET C\$ = A\$(1,4)  
                              230 D\$ = B\$(4)  
                              240 F\$(3,8)=N\$

GENERAL FORM:

statement number LET string variable = "string value"

or

statement number LET string variable = string or substring variable

or

statement number string variable = "string value"

or

statement number string variable = string or substring variable

## PURPOSE

Establishes a value for a string; the value may be a literal value in quotation marks, or a string or substring value.

## COMMENTS

Strings contain a maximum of 72 characters, enclosed by quotation marks. Strings having more than 1 character must be mentioned in a DIM statement.

Special purpose characters, such as ← or  $x^c$  may not be string characters.

If the source string is longer than the destination string, the source string is truncated at the appropriate point.

# THE STRING INPUT STATEMENT

*NOTE: These string variables have been mentioned in a DIM statement.*

EXAMPLES:                   50 INPUT R\$  
                              55 INPUT A\$,B\$, C9, D10  
                              60 INPUT A\$(1,5)  
                              65 INPUT B\$(3)

GENERAL FORM:

statement number **INPUT** string or substring variable...

## PURPOSE

Allows string values to be entered from the teleprinter.

## COMMENTS

Placing a single string variable in an INPUT statement allows the string value to be entered without enclosing it in quotation marks.

If multiple string variables are used in an INPUT statement, each string value must be enclosed in quotation marks, and the values separated by commas. The same convention is true for substring values. Mixed string and numeric values must also be separated by commas.

If a substring subscript extends beyond the boundaries of the input string, the appropriate number of blanks are appended.

Numeric variables may be used in the same INPUT statement as string variables (example statement 55 above).

# PRINTING STRINGS

## EXAMPLES:

```
105 PRINT A$
110 PRINT A$, B$, Z$
115 PRINT C$(8) "END OF STRING" B3
120 PRINT C$(1,7)
130 PRINT "THE TOTAL IS: ";X5
```

## GENERAL FORM:

statement number PRINT string or substring variable , string or substring variable...

## PURPOSE

Causes the current value of the specified string or substring variable to be output to the teleprinter.

## COMMENTS

String and numeric values may be mixed in a PRINT statement (example statements 115 and 130 above).

Specifying only one substring parameter causes the entire substring to be printed. For instance, if C\$ = "WHAT IS YOUR NAME?", example statement 120 prints:

WHAT IS

while statement 115 prints

YOUR NAME?END OF STRING 642

Numeric and string values may be "packed" in PRINT statements without using a ";", as in example statement 115.

0<sup>C</sup> and N<sup>C</sup> generate a return and linefeed respectively when printed as string characters.

# READING STRINGS

## EXAMPLES:

```
300 READ C$  
305 READ X$, Y$, Z$  
310 READ Y$(5), A,B,C5,N$  
315 READ Y$(1,4)
```

## GENERAL FORM:

statement number READ string or substring variable , string or substring variable ,...

## PURPOSE

Causes the value of a specified string or substring variable to be read from a DATA statement.

## COMMENTS

A string variable (to be assigned more than 1 character) must be mentioned in a DIM statement before attempting to READ its value.

String or substring values read from a DATA statement must be enclosed in quotation marks, and separated by commas. See "Strings in DATA Statements" in this section.

Only the number of characters specified in the DIM statement may be assigned to a string. Blanks are appended to substrings extending beyond the string dimensions.

For example, in the following code, the actual contents of B\$ is "ABC "

```
DIM A$(3), B$(72)  
A$ = "ABC"  
B$ (1,5) = A$
```

Mixed string and numeric values may be read (example statement 310 above); see "The TYP Function," Section III for description of a data type check which may be used with DATA statements.

# STRING IF

EXAMPLES:                   340 IF C\$<D\$ THEN 800  
                              350 IF C\$>=D\$ THEN 900  
                              360 IF C\$#D\$ THEN 1000  
                              370 IF N\$(3,5)<R\$(9) THEN 500  
                              380 IF A\$(10)="END" THEN 400

GENERAL FORM:

statement no. IF string variable relational oper. string var. THEN statement no.

## PURPOSE

Compares two strings. If the specified condition is true, control is transferred to the specified statement.

## COMMENTS

Strings are compared one character at a time, from left to right; the first difference determines the relation. If one string ends before a difference is found, the shorter string is considered the smaller one.

Characters are compared by their ASCII representation. See Section VII, "String Evaluation by ASCII Codes" for details.

If substring subscripts extend beyond the length of the string, null characters (rather than blanks) are appended.

## THE LEN FUNCTION

### EXAMPLE:

```
469 PRINT LEN (A$)
479 PRINT LEN (X$)
489 PRINT "TEXT"; LEN(A$); B$, C
499 IF LEN (P$) #5 THEN 600
509 IF LEN (P$) = 5 THEN 609
519 IF LEN (P$) = 5 OR LEN (P$) = 10 THEN 10
529 LET X$(LEN(X$)+1) = "ADDITIONAL SUBSTRING"
:
600 STOP
609 PRINT "STRING LENGTH = "; LEN (P$)
```

### GENERAL FORM:

*statement number statement type* **LEN** ( *string variable* ) ...

### PURPOSE

Supplies the current (logical) length of the specified string, in number of characters.

### COMMENTS

DIM merely specifies a maximum string length. The LEN function allows the user to check the actual number of characters currently assigned to a string variable.

Note that LEN is a directly executable command (See Section III), while LEN (...\$) is a pre-defined function used only as an operand in a statement. The LEN command gives the working program length; the LEN function gives the current length of a string.

## STRINGS IN DATA STATEMENTS

EXAMPLES:                   500 DATA "NOW IS THE TIME."  
                              510 DATA "HOW", "ARE", "YOU,"  
                              520 DATA 5.172, "NAME?", 6.47,5071

GENERAL FORM:

*statement number* DATA " *string text* " , " *string text* " ...

### PURPOSE

Specifies data in a program (numeric values may also be used as data).

### COMMENTS

String values must be enclosed by quotation marks and separated by commas.

String and numeric values may be mixed in a single DATA statement. They must be separated by commas (example statement 520 above).

Strings up to 72 characters long may be stored in a DATA statement.

See "The TYP Function," Section III, for description of a data type (string, numeric) check which may be used with DATA statements.

# PRINTING STRINGS ON FILES

## EXAMPLES:

```
350 PRINT #4; "THIS IS A STRING."  
355 PRINT #2; C$, B$, X$, Y$, D$  
360 PRINT #3,3; X$, P$, "TEXT", 27.5,R7  
365 PRINT #N,R; P$, N, A(5,5), "TEXT"
```

## GENERAL FORM:

statement number PRINT# file number , record number formula ; string variable ...

or

statement number PRINT# file number formula , record number formula ; "string text "

or

statement number PRINT# file number formula ; string variable or substring variable...

## PURPOSE

Prints string or substring variables on a file.

## COMMENTS

String and numeric variables may be mixed in a single file or record within a file (example statement 360 above).

The formula for determining the number of 2-character words required for storage of a string on a file is:

$1 + \frac{\text{number of characters in string}}{2}$  if the number of characters is even;

$1 + \frac{\text{number of characters in string} + 1}{2}$  if the number of characters is odd.

A maximum of 248 string characters may be stored on 1 file record.

See "The TYP Function," Section III for description of a data type check.

# READING STRINGS FROM FILES

## EXAMPLES:

71Ø READ #1, 5; A\$, B\$

715 READ #2; C\$, A1, B2, X

72Ø READ #3,6; C\$(5),X\$(4,7),Y\$

73Ø READ #N,P; C\$, V\$(2,7), R\$(9)

## GENERAL FORM:

statement no. READ# file no. formula , record no. formula ; string or substring variable...

or

statement no. READ# file no. formula ; string or substring variable...

## PURPOSE

Reads string and substring values  
from a file.

## COMMENTS

String and numeric values may be  
mixed in a file and in a READ#  
statement; they must be separated  
by commas.

See "The TYP Function", Section III,  
for description of a data type check.

# SECTION VII

## LOGICAL OPERATIONS

### LOGICAL VALUES AND NUMERIC VALUES

When using the logical capability of Time Shared BASIC, be sure to distinguish between logical values and the numeric values produced by logical evaluation.

The logical value of an expression is determined by definitions established in the user's program.

The numeric values produced by logical evaluation are assigned by Time Shared BASIC. The user may not assign these values.

Logical value is the value of an expression or statement, using the criteria:

any nonzero expression value = "true"  
any expression value of zero = "false"

When an expression or statement is logically evaluated, it is assigned one of two numeric values, either:

1, meaning the expression or statement is "true",  
or  
0, meaning the expression or statement is "false".

# RELATIONAL OPERATORS

There are two ways to use the relational operators in logical evaluations:

1. As a simple check on the numeric value of an expression.

EXAMPLES:

150 IF B=7 THEN 600

200 IF A#27.65 THEN 700

300 IF (Z/10)>=0 THEN 800

When a statement is evaluated, if the "IF" condition is currently true (for example, in statement 150, if B = 7), then control is transferred to the specified statement.

Note that the numeric value produced by the logical evaluation is unimportant when the relational operators are used in this way. The user is concerned only with the presence or absence of the condition indicated in the IF statement.

## RELATIONAL OPERATORS CONTINUED

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = 0.

EXAMPLES:	610 LET X=27
	615 PRINT X=27
	620 PRINT X#27
	630 PRINT X>=27

The example PRINT statements give the numeric values produced by logical evaluation. For instance, statement 615 is interpreted by TSB as "Print 1 if X equals 27, 0 if X does not equal 27." There are only two logical alternatives; 1 is used to represent "true", and 0 "false".

The numeric value of the logical evaluation is dependent on, but distinct from, the value of the expression. In the example above, X equals 27, but the numeric value of the logical expression X=27 is 1, since it describes a "true" condition.

# BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.

1. As logical checks on the value of an expression or expressions.

EXAMPLES:

```
51Ø IF A1 OR B THEN 67Ø
52Ø IF B3 AND C9 THEN 68Ø
53Ø IF NOT C9 THEN 69Ø
54Ø IF X THEN 7ØØ
```

Statement 51Ø is interpreted: "if either A1 is true (has a nonzero value) or B is true (has a nonzero value) then transfer control to statement 67Ø."

Similarly, statement 54Ø is interpreted: "if X is true (has a nonzero value) then transfer control to statement 7ØØ."

The Boolean operators evaluate expressions for their logical values only; these are "true" = any non-zero value, "false" = zero. For example, if B3 = 9 and C9 = -5, statement 52Ø would evaluate to "true", since both B3 and C9 have a nonzero value.

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = Ø.

EXAMPLES:

```
49Ø LET B = C = 7
5ØØ PRINT B AND C
51Ø PRINT C OR B
52Ø PRINT NOT B
```

Statements 5ØØ - 52Ø returns a numeric value of either: 1, indicating that the statement has a logical value of "true", or Ø, indicating a logical value of "false".

Note that the criteria for determining the logical values are:

true = any nonzero expression value  
false = an expression value of Ø.

The numeric value 1 or Ø is assigned accordingly.

## SOME EXAMPLES

These examples show some of the possibilities for combining logical operators in a statement.

It is advisable to use parentheses wherever possible when combining logical operators.

```
EXAMPLES:      310 IF (A9 MIN B7)<0 OR (A9 MAX B7)>100 THEN 900
                310 PRINT (A>B) AND (X<Y)
                320 LET C = NOT D
                330 IF (C7 OR D4) AND (X2 OR Y3) THEN 930
                340 IF (A1 AND B2) AND (X2 AND Y3) THEN 940
```

The numerical value of "true" or "false" may be used in algebraic operations. For example, this sequence counts the number of zero values in a file:

```
90 LET X = 0
100 FOR I = 1 TO N
110 READ #1; A
120 LET X = X+(A=0)
130 NEXT I
140 PRINT N; "VALUES WERE READ."
150 PRINT X; "WERE ZEROES."
160 PRINT (N-X); "WERE NONZERO."
```

Note that X is increased by 1 or 0 each time A is read; when A = 0, the expression A = 0 is true, and X is increased by 1.

## SECTION VIII

### FOR THE PROFESSIONAL

This section contains the most precise reference authority -- the syntax requirements of Time Shared BASIC. The syntax requirements are explicit and unambiguous. They may be used in all cases to clarify descriptions of BASIC language features presented in other sections.

The other subsections give technical information of interest to the sophisticated user.

# SYNTAX REQUIREMENTS OF TSB

## LEGEND

::= "is defined as..."  
| "or"  
< > enclose an element of Time Shared BASIC

## LANGUAGE RULES

1. Exponents have 1 or 2 digit integers only.
2. A <parameter> primary appears only in the defining formula of a <DEF statement>.
3. A <sequence number> must lie between 1 and 9999 inclusive.
4. An array bound must lie between 1 and 9999 inclusive; a string variable bound must lie between 1 and 72 inclusive.
5. The character string for a <REM statement> may include the character " .
6. An array may not be transposed into itself, nor may it be both an operand and the result of a matrix multiplication.

*Note: Parentheses, (), and square brackets, [], are accepted interchangeably by the syntax analyzer.*

## SYNTAX REQUIREMENTS OF TSB, CONTINUED

<constant>	::= <number> +<number> -<number> <literal string>
<number>	::= <decimal number> <decimal number><exponent part>
<decimal number>	::= <integer> <integer>. <integer>.<integer> . <integer>
<integer>	::= <digit> <integer><digit>
<digit>	::= 0 1 2 3 4 5 6 7 8 9
<exponent part>	::= E<integer> E+<integer> E-integer ( <i>see rule 1</i> )
<literal string>	::= "<character string>"
<character string>	::= <character> <character string><character>
<character>	::= any ASCII character except null, line feed, return, x-off, X <sup>C</sup> , ←, " , and rubout
<variable>	::= <simple variable> <subscripted variable>
<simple variable>	::= <letter> <letter><digit>
<letter>	::= A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
<subscripted variable>	::= <letter>(<sublist>)
<sublist>	::= <expression> <expression>,<expression>
<string variable>	::= <string simple variable> <string simple variable>(<sublist>)
<string simple variable>	::= <letter>\$
<expression>	::= <conjunction> <expression>OR<conjunction>
<conjunction>	::= <relation> <conjunction>AND<relation>
<relation>	::= <minmax> <minmax><relational operator><minmax>
<minmax>	::= <sum> <minmax>MIN<sum> <minmax>MAX<sum>
<sum>	::= <term> <sum>+<term> <sum>-<term>
<term>	::= <subterm> <term>*<subterm> <term>/<subterm>
<subterm>	::= <denial> <signed factor>

# SYNTAX REQUIREMENTS OF TSB , CONTINUED

<denial>	::=	<factor> NOT<factor>
<signed factor>	::=	+<factor> -<factor>
<factor>	::=	<primary> <factor>†<primary>
<primary>	::=	<variable> <number> <functional> <parameter> (rule 2)  (<expression>)
<relational operator>	::=	< <=> <#> <>> <=>
<parameter>	::=	<letter> <letter><digit>
<functional>	::=	<function identifier>(<expression>)  <pre-defined function>(<expression>)  LEN (<string simple variable>)
<function identifier>	::=	FN <letter>
<pre-defined function>	::=	SIN COS TAN ATN EXP LOG ABS SQR INT RND SGN TYP TIM
<source string>	::=	<string variable> <literal string>
<destination string>	::=	<string variable>
<file reference>	::=	#<file formula> #<file formula>,<record formula>
<file formula>	::=	<expression>
<record formula>	::=	<expression>
<array identifier>	::=	<letter>
<sequence number>	::=	<integer> (see rule 3)
<program statement>	::=	<sequence number><BASIC statement>carriage return
<BASIC statement>	::=	<LET statement> <IF statement> <GOTO statement>  <GOSUB statement> <RETURN statement> <FOR statement>  <NEXT statement> <STOP statement> <END statement>  <DATA statement> <READ statement> <INPUT statement>  <PRINT statement> <RESTORE statement> <DIM statement>  <COM statement> <DEF statement> <FILES statement>  <REM statement> <CHAIN statement> <MAT statement>
<LET statement>	::=	LET <leftpart><expression>  LET <destination string>=<source string>  <leftpart><expression>  <destination string>=<source string>
<leftpart>	::=	<variable>= <leftpart><variable>=
<IF statement>	::=	IF<decision expression>THEN<sequence number>  IF END #<file formula>THEN<sequence number>
<decision expression>	::=	<expression>  <comparison string 1><relational operator> <comparison string 2>

## SYNTAX REQUIREMENTS OF TSB , CONTINUED

<comparison string 1>	::=	<string variable>
<comparison string 2>	::=	<string variable> <literal string>
<GOTO statement>	::=	GOTO <sequence number>  GOTO <expression>OF<sequence list>
<sequence list>	::=	<sequence number> <sequence list>,<sequence number>
<GOSUB statement>	::=	GOSUB <sequence number>  GOSUB <expression>OF <sequence list>
<RETURN statement>	::=	RETURN
<FOR statement>	::=	FOR <for variable>=<initial value>TO<final value>  FOR <for variable>=<initial value>TO<final value> STEP<step size>
<for variable>	::=	<simple variable>
<initial value>	::=	<expression>
<final value>	::=	<expression>
<step size>	::=	<expression>
<NEXT statement>	::=	NEXT<for variable>
<STOP statement>	::=	STOP
<END statement>	::=	END
<DATA statement>	::=	DATA<constant> <DATA statement>,<constant>
<READ statement>	::=	READ<variable list> READ<file reference>  READ<file reference>;<variable list>
<variable list>	::=	<read variable> <variable list>,<read variable>
<read variable>	::=	<variable> <destination string>
<INPUT statement>	::=	INPUT<variable list>
<PRINT statement>	::=	<type statement> <file write statement>  PRINT<file reference>
<type statement>	::=	<print 1> <print 2>
<print 1>	::=	PRINT <print 2>,<print 2>;<print 3>
<print 2>	::=	<print 1><print expression> <print 3>
<print 3>	::=	<type statement><literal string>
<print expression>	::=	<expression> TAB(<expression>) <source string>
<file write statement>	::=	PRINT<file reference>;<write expression>  <file write statement>,<write expression>  <file write statement>;<write expression>  <file write statement><literal string>  <file write statement><literal string> <write expression>

# SYNTAX REQUIREMENTS OF TSB, CONTINUED

<write expression>	::=	<expression> END <source string>
<RESTORE statement>	::=	RESTORE RESTORE<sequence number>
<DIM statement>	::=	DIM<dimspec> <DIM statement>,<dimspec>
<COM statement>	::=	COM<com list element>  <COM statement>,<com list element>
<com list element>	::=	<simple variable> <string simple variable>  <dimspec>
<dimspec>	::=	<array identifier>(<bound>)  <array identifier>(<bound>,<bound>)  <string simple variable>(<bound>)
<bound>	::=	<integer> (see rule 4)
<DEF statement>	::=	DEF<function identifier>(<parameter>)=<expression>
<FILES statement>	::=	FILES<name> <FILES statement>,<name>
<name>	::=	a string of 1 to 6 printing characters>
<REM statement>	::=	REM<character string> (see rule 5)
<CHAIN statement>	::=	CHAIN<name> <CHAIN \$<name>
<MAT statement>	::=	<MAT READ statement> <MAT INPUT statement>  <MAT PRINT statement> <MAT initialization statement>  <MAT assignment statement>
<MAT READ statement>	::=	MAT READ<actual array>  MAT READ<file reference>;<actual array>  <MAT READ statement>,<actual array>
<actual array>	::=	<array identifier> <array identifier>(<dimensions>)
<dimensions>	::=	<expression> <expression>,<expression>
<MAT INPUT statement>	::=	MAT INPUT<actual array>  <MAT INPUT statement>,<actual array>
<MAT PRINT statement>	::=	<MAT PRINT 1> <MAT PRINT 2>
<MAT PRINT 1>	::=	MAT PRINT<array identifier>  MAT PRINT<file reference>;<array identifier>  <MAT PRINT 2><array identifier>
<MAT PRINT 2>	::=	<MAT PRINT 1>,<MAT PRINT 1>;
<MAT initialization statement>	::=	MAT<array identifier>=<initialization function>  MAT<array identifier>=<initialization function> (<dimensions>)
<initialization function>	::=	ZER CON IDN

## SYNTAX REQUIREMENTS OF TSB CONTINUED

<MAT assignment  
statement> (rule 6) ::= MAT<array identifier>=<array identifier>|  
MAT<array identifier>=<array identifier><mat operator>  
<array identifier>|  
MAT<array identifier>=INV(<array identifier>)  
MAT<array identifier>=TRN(<array identifier>)|  
MAT<array identifier>=(<expression>)\*<array identifier>

<mat operator> ::= +|-|\*

# STRING EVALUATION BY ASCII CODES

Each teleprinter character is represented by an ASCII (American Standard Code for Information Interchange) number.

Strings are compared by their ASCII representation.

The ASCII sequence, from lowest to highest is:

<i>Lowest:</i>	<u>bell</u>		
	space	5	I
	!	6	J
	#	7	K
	\$	8	L
	%	9	M
	&	:	N
	'	;	O
	(	<	P
	)	=	Q
	*	>	R
	+	?	S
	,	@	T
	-	A	U
	.	B	V
	/	C	W
	Ø	D	X
	1	E	Y
	2	F	Z
	3	G	[
	4	H	\
			]
			↑ <i>Highest</i>

Quotation marks are used to delimit strings, and may not be used within a string.

# MEMORY ALLOCATION BY A USER

Approximate number of 2-character words per user: 4,500

System overhead (approx.): 320

Space available for user allocation: 4,180 2-character words

## SOME EXAMPLES OF USER-DETERMINED ALLOCATION\*

- a) Introduction of each simple, string, or matrix variable uses 4 words.
- b) A 9 word stack is reserved for GOSUB's.
- c) 6 X (maximum level of FOR...NEXT loop nesting)
- d) Each file name mentioned in a FILES statement reserves 128 words for buffer space.
- e) An approximate estimate of space required for a program is:
  - 11 words per BASIC statement
  - +2X(number of matrix elements dimensioned)
  - +1/2X(number of string characters used)

\* *This is variable "system overhead"; it is not included in word counts produced by the LEN command.*

# APPENDIX A

## HOW TO PREPARE A PAPER TAPE OFF-LINE

To prepare a paper tape for input:

1. Turn teleprinter control knob to "LOCAL".
2. Press the "ON" button (on tape punch).
3. Press the "HERE IS" key; or press @<sup>C</sup> (control shift "p") several times to put leading holes on the tape.
4. Type program as usual, following each line with return linefeed.
5. Press "HERE IS"; or press @<sup>C</sup> several times to put trailing holes on the tape.
6. Press the "OFF" button on the tape punch.

### COMMENTS

The standard on-line editing features, such as x<sup>C</sup>, ←, and repeating the same line number may be punched on tape; x<sup>C</sup> must be followed by return linefeed.

Pressing the "B.SP." (backspace) button on the tape punch, then the "RUBOUT" key will physically delete the previous character from a paper tape.

# APPENDIX B

## THE X-ON, X-OFF FEATURE

Terminals equipped with the X-ON, X-OFF feature may be used to input data from a paper tape while a program is running.

Data is punched on paper tape in this format:

line of data items separated by commas x-off return linefeed

(x-off, return and linefeed are teleprinter keys.)

### COMMENTS

Remember that each line of data must end with x-off return linefeed.

See Appendix A, "Preparing A Paper Tape Offline," for instructions on editing a paper tape.

# APPENDIX C

## DIAGNOSTIC MESSAGES

### COMMAND ERROR MESSAGES

#### SAVE

NO PROGRAM  
NO PROGRAM NAME  
FILE SPACE FULL  
SYSTEM OVERLOAD  
DUPLICATE ENTRY  
RUN ONLY

#### GET

INVALID NAME  
NO SUCH PROGRAM  
ENTRY IS A FILE  
PROGRAM TOO LARGE

#### APPEND

NO COMMON AREA ALLOWED  
PROGRAM TOO LARGE  
SEQUENCE NUMBER OVERLAP  
INVALID NAME  
NO SUCH PROGRAM  
ENTRY IS A FILE

#### HELLO

ILLEGAL ACCESS  
NO TIME LEFT

#### KILL

ILLEGAL NAME  
NO SUCH PROGRAM  
FILE IN USE

#### RENUMBER

SEQUENCE NUMBER OVERFLOW  
BAD PARAMETER

#### NAME

ONLY 6 CHARACTERS ACCEPTED  
\$ ILLEGAL AS FIRST CHARACTER

#### DELETE

NOTHING DELETED

#### PROTECT

PRIVILEGED COMMAND  
INVALID NAME  
NO SUCH PROGRAM

#### OPEN

FILE SPACE FULL  
SYSTEM OVERLOAD  
DUPLICATE ENTRY

#### LIST

RUN ONLY

#### PUNCH

RUN ONLY

#### UNPROTECT

PRIVILEGED COMMAND  
INVALID NAME  
NO SUCH PROGRAM

#### XPUNCH

RUN ONLY

# DIAGNOSTIC MESSAGES CONTINUED

## LANGUAGE ERROR MESSAGES

### Syntax Errors

OUT OF STORAGE  
ILLEGAL OR MISSING INTEGER  
EXTRANEIOUS LIST DELIMITER  
MISSING ASSIGNMENT OPERATOR  
CHARACTERS AFTER STATEMENT END  
MISSING OR ILLEGAL SUBSCRIPT  
MISSING OR BAD LIST DELIMITER  
MISSING OR BAD FUNCTION NAME  
MISSING OR BAD SIMPLE VARIABLE  
MISSING OR ILLEGAL 'OF'  
MISSING OR ILLEGAL 'THEN'  
MISSING OR ILLEGAL 'TO'  
MISSING OR ILLEGAL 'STEP'  
MISSING OR ILLEGAL DATA ITEM  
ILLEGAL EXPONENT  
SIGN WITHOUT NUMBER  
MISSING RELATIONAL OPERATOR  
ILLEGAL READ VARIABLE  
ILLEGAL SYMBOL FOLLOWS 'MAT'  
MATRIX CANNOT BE ON BOTH SIDES  
NO '\*' AFTER RIGHT PARENTHESIS  
NO LEGAL BINARY OPERATOR FOUND  
MISSING LEFT PARENTHESIS  
MISSING RIGHT PARENTHESIS  
PARAMETER NOT STRING VARIABLE  
UNDECIPHERABLE OPERAND  
MISSING OR BAD ARRAY VARIABLE  
STRING VARIABLE NOT LEGAL HERE  
MISSING OR BAD STRING OPERAND  
NO CLOSING QUOTE  
72 CHARACTERS MAX FOR STRING  
STATEMENT HAS EXCESSIVE LENGTH  
MISSING OR BAD FILE REFERENCE

### Run Time Errors

UNDEFINED STATEMENT REFERENCE  
NEXT WITHOUT MATCHING FOR  
SAME FOR-VARIABLE NESTED  
FUNCTION DEFINED TWICE IN LINE n  
VARIABLE DIMENSIONED TWICE IN LINE n  
LAST STATEMENT NOT 'END' IN LINE n  
UNMATCHED FOR  
UNDEFINED FUNCTION  
ARRAY TOO LARGE  
ARRAY OF UNKNOWN DIMENSIONS  
OUT OF STORAGE  
DIMENSIONS NOT COMPATIBLE IN LINE n  
CHARACTERS AFTER COMMAND END  
BAD FORMAT OR ILLEGAL NAME  
MISSING OR PROTECTED FILE  
GOSUBS NESTED TEN DEEP  
RETURN WITH NO PRIOR GOSUB  
SUBSCRIPT OUT OF BOUNDS  
NEGATIVE STRING LENGTH  
NON-CONTIGUOUS STRING CREATED  
STRING OVERFLOW  
OUT OF DATA  
DATA OF WRONG TYPE  
UNDEFINED VALUE ACCESSED  
MATRIX NOT SQUARE  
REDIMENSIONED ARRAY TOO LARGE  
NEARLY SINGULAR MATRIX  
LOG OF NEGATIVE ARGUMENT  
SQRT OF NEGATIVE ARGUMENT  
ZERO TO ZERO POWER  
NEGATIVE NUMBER TO REAL POWER  
ARGUMENT OF SIN OR TAN TOO BIG  
OVER/UNDERFLOWS - WARNING ONLY  
LAST INPUT IGNORED, RETYPE IT  
TOO MANY FILES STATEMENTS  
NON-EXISTENT FILE REQUESTED  
WRITE TRIED ON READ-ONLY FILE  
END-OF-FILE/END OF RECORD  
INVALID PROGRAM NAME IN CHAIN  
NON-EXISTENT PROGRAM REQUESTED  
CHAIN REQUEST IS A FILE  
PROGRAM CHAINED IS TOO LARGE  
COM STATEMENT OUT OF ORDER  
ARGUMENT OF TIM OUT OF RANGE IN LINE n

# DIAGNOSTIC MESSAGES CONTINUED

## Warnings

BAD INPUT, RETYPE FROM ITEM  
LOG OF ZERO - WARNING ONLY  
ZERO TO NEGATIVE POWER-WARNING  
DIVIDE BY ZERO - WARNING ONLY  
EXP OVERFLOW - WARNING ONLY  
OVERFLOW - WARNING ONLY  
UNDERFLOW - WARNING ONLY  
EXTRA INPUT - WARNING ONLY  
READ-ONLY FILES:

*Diagnostic messages printed while entering a program refer only to the first error found in a line.*

? (Input is required to continue execution.)  
?? (More input is required to continue execution.)  
??? (Input is unintelligible.)

# INDEX

↑.....	2-6	ATN Function.....	3-23
←.....	1-12	Backspace.....	1-12
↘.....	1-13	BASIC.....	1-2,2-1
;	2-27	Before Going On-Line.....	1-10
,	1-23,2-23,2-25,2-27	Boolean Operators.....	7-4
+	2-6,5-11	<i>break</i> .....	1-24,2-41
-	2-6,5-12	BYE Command.....	2-35
/	2-6	CATALOG Command.....	3-15
*	2-6,5-13,5-14	CHAIN Statement.....	3-28
=	2-5,2-7,2-17,5-15,6-9	Changing Statements.....	1-13
#	2-7,6-13	COLUMNS.....	3-3,5-1
<>.....	2-7,6-13	Communication Between Programs...3-29	
<	2-7,6-13	COM Statement.....	3-29
>	2-7,6-13	Commands.....	2-13,2-33
>=.....	2-7,2-17,6-13	Comments.....	2-15
<=.....	2-7,2-17,6-13	Comparing Strings.....	6-13,8-8
ABS Function.....	3-22	Conditionals.....	2-17
Accuracy.....	2-2	Connections.....	1-16,1-17
Acoustic Coupler.....	1-16	CONTENTS.....	vii
Add.....	2-6,5-11	Control C.....	1-24
Adding Matrices.....	5-11	Control Characters.....	1-18
Adding to a Serial File.....	4-20	CONVENTIONS.....	iv
Advanced BASIC.....	3-1	Copying a File.....	4-33
Alphabetical File.....	4-43	Copying a Matrix.....	5-14
AND Operator.....	2-9	COS Function.....	3-23
APPEND Command.....	3-12	Current Program.....	3-6
Arithmetic Evaluation.....	2-4	Data Set.....	1-16
Arithmetic Operators.....	2-6	DATA Statement.....	2-21,6-15
Arguments.....	3-4	Data Types.....	3-25,4-16,4-31
Array.....	3-3,5-1	Declaration of Files.....	4-8
Assignment Operator.....	2-5,6-9	DEF FN Statement.....	3-21
Assignment Statement.....	2-14,6-9	DELETE Command.....	3-13

# INDEX

- Deleting Programs.....3-6,3-11
- Deleting Statements.....1-13
- Determining Length of a File....4-29
- Diagnostic Messages.....1-22,C-1
- DISC Command.....2-47
- Divide.....2-6
- DIM Statement.....5-2,6-8
- ECHO Command.....2-36
- End-of-File Marker....4-17,4-18,4-19  
4-23,4-26,4-30
- End-of-Record Marker.....4-26
- END Statement.....1-7,2-28
- E Notation.....2-2
- Equality.....2-7,6-13
- Erasing a Record.....4-40,4-41
- Error Messages.....1-22,C-1
- Essentials of BASIC.....2-1
- Execution.....1-23
- EXP Function.....3-22
- Exponentiate.....2-6
- Expression.....2-4
- False.....2-7
- File Names.....4-5
- File Numbers.....4-3,4-8,4-10,4-12
- File Pointer.....4-2,4-3,4-14,4-16  
4-23,4-28
- Files.....4-1,4-2,4-5
- FILES Statement.....4-3,4-8
- Format, Page.....v
- FOR..NEXT Statements.....2-18,3-20
- FOR Statement.....2-18
- Functions....3-4,3-16,3-21,3-22,3-23  
3-24,3-25,3-26,4-15,4-16,4-31,6-14
- GET Command.....3-10
- GOSUB...RETURN Statements.....3-17
- GO TO Statement.....2-16
- Greater Than.....2-7,6-13
- Greater Than or Equal To.....2-7,6-13
- Half-Duplex Coupler.....1-16
- HELLO Command.....2-34
- How To Use This Book.....vi
- IDcode.....1-17,1-19,2-34
- Identity Matrix.....5-16
- IDN.....5-16
- IF END Statement.....4-15,4-18
- IF..THEN Statement.....2-17,6-13
- Inequality.....2-7,6-13
- Input Logs.....1-25
- INPUT Statement.....2-24,5-5,6-10
- Inputting a String.....6-10
- Inputting Matrix Elements.....5-5,5-6
- Instructions.....1-5
- INT Function.....3-22
- Introduction.....1-1
- INV.....5-18
- Inverting Matrices.....5-18
- KEY Command.....2-45
- KILL Command.....3-11,4-4,4-7
- LEN Function.....3-26,6-14
- LENGTH Command.....3-7,6-14
- Length of Programs.....3-7,3-26
- Length of Files.....4-29
- Less Than.....2-7,6-13
- Less Than or Equal To.....2-7,6-13
- LET Statement.....2-5,2-14
- LIBRARY Command.....3-14

# INDEX

Line Number.....	1-4	MAX Operator.....	2-8
LINE Setting.....	1-16	Memory Allocation.....	8-9
Linking Programs.....	3-28	MESSAGE Command.....	2-48
LIST Command.....	1-14,2-38	MIN Operator.....	2-8
List Contents of a Record.....	4-32	Minimum Number.....	2-2
Listing a Program.....	1-14,2-38	Modifying a Record.....	4-39,4-42
Listing a File.....	4-15	Modifying a Serial File.....	4-20,4-21
LOG Function.....	3-22	Moving the Pointer.....	4-28
Logging In.....	1-17,1-19,1-20	Multibranch GOSUB.....	3-18
Logging Out.....	1-19	Multibranch GOTO.....	2-16
Logical Length of Strings...6-3,6-14		Multiply.....	2-6,5-13
Logical Operations.....	7-1	Multiplying Matrices.....	5-13
Logical Value.....	7-1	NAME Command.....	3-8
Loop.....	1-25	Nested GOSUBS.....	3-17,3-19
Looping.....	2-18,3-20	Nesting Loops.....	2-20
Mathematical Functions.....	3-22	NEXT Statement.....	2-18
MAT...CON Statement.....	5-4	NOT Operator.....	2-11
MAT INPUT Statement.....	5-6	Numbers.....	2-2
MAT PRINT Statement.....	5-8	OPEN Command.....	4-3,4-5
MAT PRINT#M; Statement.....	5-19	Operands.....	1-6
MAT READ Statement.....	5-10	OR Operator.....	2-10
MAT READ#M; Statement.....	5-20	Paper-Tape.....	A-1
MAT...ZER Statement.....	5-3	Parenthesis.....	2-6,2-12
Matrix.....	3-3,5-1	Password.....	1-17,1-19,2-34
Matrix Addition.....	5-11	Physical End-of-File.....	4-23,4-26
Matrix Element.....	5-1	Physical Length of Strings.....	6-3
Matrix File Print.....	5-19	Pointers.....	2-21
Matrix File Read.....	5-20	Precedence.....	2-6,2-12
Matrix Inversion.....	5-18	Precision.....	2-2
Matrix Multiplication.....	5-13	Printing Matrix Elements.....	5-7,5-8 5-19
Matrix Subtraction.....	5-12	Printing Strings.....	6-11
Matrix Transposition.....	5-17	PRINT Statement.....	2-26,5-7,6-11
Matrix Variable.....	5-2	PRINT #..END Statement.....	4-19
Maximum Number.....	2-2	PRINT#M; Statement.....	6-16

# INDEX

- PRINT# Statement.....4-3,4-10  
PRINT#M,N Statement.....4-40  
PRINT#M,N; Statement.....4-35,4-36  
Program.....1-7,1-25  
Public Files.....4-5,4-9  
Public Library Programs....3-10,3-12  
3-14  
PUNCH Command.....2-42  
Random File Access....4-34,4-35,4-36  
4-38,4-39  
Random Numbers.....3-22  
Reading Matrix Elements.....5-9,5-10  
5-20  
Reading Strings.....6-12  
READ Statement.....2-21,6-12  
READ# Statement.....4-4,4-12  
READ#M; Statement.....6-17  
READ#N,1.....4-14  
READ#M,N Statement....4-28,4-29,4-32  
READ#M,N; Statement.....4-35,4-38  
Records.....3-5,4-2,4-26,4-27,4-31  
4-32,4-34,4-36,4-38,4-39  
Relational Operators....2-7,6-13,7-2  
7-3  
Remark.....2-15  
REM Statement.....2-15  
RENUMBER Command.....2-40  
Resetting the File Pointer.....4-14  
RESTORE Statement.....2-21  
Retrieving Programs.....3-10  
return.....1-11  
RETURN.....3-17,3-19  
RND Function.....3-22  
Routine.....3-2,3-16  
Rows.....3-3,5-1  
RUN Command.....1-23,2-37  
Running a Program.....1-23,2-32,2-37  
Run-Only.....3-12  
Sample Programs...1-21,1-23,1-25,2-29  
4-3,4-15,4-21,4-29,4-32,4-33,4-35  
4-41,4-42,4-43  
SAVE Command.....3-9  
Scalar Multiplication.....5-14  
SCRATCH Command.....2-39  
Sequence of Statements.....1-4  
Serial File Access.....4-3,4-10,4-12  
4-14,4-16,4-20,4-23,4-30  
Serial File Print.....4-10  
Serial File Read.....4-12  
SGN Function.....3-24  
SIN Function.....3-23  
Spaces.....1-8  
Spot Checks.....1-9,1-15  
SQR Function.....3-22  
Statement Format.....1-8  
Statement Numbers.....1-4,2-40  
Statement Types.....1-5  
Statements.....1-3,2-13  
STEP.....3-20  
Stopping a Program.....1-24,2-37  
STOP Statement.....2-28  
Storage Requirements.....4-27  
Storing Programs.....3-6,3-9  
String Assignment Statement.....6-9  
String Evaluation.....8-8  
String File Print.....6-16  
String File Read.....6-17  
String Variable.....6-2,6-3,6-6  
Strings.....3-4,4-27,6-1,6-2,6-6

# INDEX

Strings in DATA Statements.....	6-15	Telephone.....	1-16
Strings in IF Statements.....	6-13	TIME Command.....	2-46
Structure of Serial Files.....	4-23	Time-Out on Input.....	3-31
Subdividing Serial Files.....	4-30	TYP Function.....	3-25,4-15,4-16 4-31,4-32
Subroutines.....	3-16,3-17	Time Sharing.....	1-1
Subscripts.....	5-1,6-4,6-6	TIM Function.....	3-27
Substring.....	6-4,6-6	Transposing Matrices.....	5-17
Subtract.....	2-6,5-12	Trigonometric Functions.....	3-23
Subtracting Matrices.....	5-12	TRN.....	5-17
Summary.....	D-1	True.....	2-7
Syntax Requirements of BASIC	8-2	Variables.....	1-25,2-3
TAB Function.....	3-24	Word.....	3-5
TAN Function.....	3-23	Working Size.....	5-2
TAPE Command.....	2-44	XPUNCH Command.....	2-43
Teleprinter.....	1-16	X-ON, X-OFF.....	B-1
		Zeroing a Matrix.....	5-3