

## R.A.I.R., Inc.

|                      |                   |        |
|----------------------|-------------------|--------|
| Computer Numbers     | <u>969-2800</u>   | 10 CPS |
|                      | <u>969-2918</u>   | 30 CPS |
| Logon Message        | HELLO-            |        |
| System Backup Time   | 5:30 TO 6:00 P.M. |        |
| For Assistance, Call | <u>964-0413</u>   | (days) |
|                      | (eves.)           |        |

# 2000C:

## A GUIDE TO TIME-SHARED BASIC

FOOG PETE

2000C:  
**A GUIDE TO TIME-SHARED BASIC**

For Reference and Self-Instruction



Software Publications  
Cupertino, California  
95014

© Copyright, 1971, by  
HEWLETT-PACKARD COMPANY  
Cupertino, California  
Printed in the U.S.A.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system (e.g., in memory, disc or core) or transmitted by any means, electronic, mechanical, photocopy, recording or otherwise, without prior written permission from the publisher.

Printed in the U.S.A.

# PREFACE

This publication is designed to meet two requirements:

1. To serve as a clear and concise reference text for Time-Shared BASIC, and
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 Naur Form. Each element of a statement is underlined.

The 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...   |
| 20 PRINT X,Y<br>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 linefeed</u><br><u>esc ctrl</u><br><u>alt-mode break</u> | Represents the terminal keys:<br>Return, Linefeed, Escape, Control,<br>Alt-Mode, and Break.                       |
| Note: Both X and...  | 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.   |

# CONTENTS

|      |   |
|------|---|
| iii  | PREFACE   |
| iv   | CONVENTIONS USED IN THIS TEXT                     |
| 1-1  | SECTION I<br>AN INTRODUCTION TO TIME SHARED BASIC |
| 1-4  | SPECIAL KEYS                                      |
| 1-5  | USING THE TELEPRINTER TERMINAL                    |
| 1-6  | COMMUNICATING WITH THE TSB SYSTEM                 |
| 1-7  | EXAMPLES OF BASIC STATEMENTS                      |
| 1-8  | STATEMENT NUMBERS                                 |
| 1-9  | INSTRUCTIONS (STATEMENT TYPES)                    |
| 1-10 | OPERANDS  |
| 1-11 | A PROGRAM   |
| 1-12 | THE FORMAT OF STATEMENTS                          |
| 1-14 | BEFORE GOING ON-LINE                              |
| 1-15 | PRESS RETURN AFTER EACH STATEMENT                 |
| 1-16 | DELETING OR CHANGING CHARACTERS                   |
| 1-17 | DELETING OR CHANGING A STATEMENT                  |
| 1-18 | LISTING A PROGRAM                                 |
| 1-21 | CHECKING THE CONNECTION                           |
| 1-21 | YOUR ID CODE and PASSWORD                         |
| 1-21 | CONTROL CHARACTERS                                |
| 1-22 | SAMPLE LOG IN AND LOG OUT                         |
| 1-23 | MISTAKES DURING LOG IN                            |
| 1-24 | ENTERING THE SAMPLE PROGRAM                       |
| 1-25 | HOW TO OBTAIN A DIAGNOSTIC MESSAGE                |
| 1-26 | RUNNING THE SAMPLE PROGRAM                        |
| 1-27 | STOPPING A PROGRAM: THE <u>break</u> KEY          |
| 1-28 | 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: EXPRESSION
  - 2-4 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
  - 2-30 END AND STOP
  - 2-31 SAMPLE PROGRAM
  - 2-34 RUNNING THE SAMPLE PROGRAM
  - 2-35 COMMANDS
  - 2-36 HELLO
  - 2-37 BYE
  - 2-38 ECHO-
  - 2-39 RUN
  - 2-40 LIST

## CONTENTS CONTINUED

2-41 SCRATCH  
2-42 RENUMBER  
2-44 BREAK  
2-45 PUNCH AND XPUNCH  
2-47 TAPE  
2-48 KEY  
2-49 TIME  
2-50 MESSAGE

### 3-1 SECTION III

#### ADVANCED BASIC

3-2 ROUTINE  
3-3 TERM: ARRAY  
3-5 TERM: STRING AND STRING VARIABLE  
3-5 TERM: FUNCTION  
3-6 TERM: WORD  
3-7 STORING AND DELETING PROGRAMS  
3-8 LENGTH  
3-9 NAME-  
3-10 SAVE- AND CSAVE-  
3-11 GET- GET-\$, AND GET-\*  
3-12 KILL-  
3-13 APPEND-  
3-14 DELETE-  
3-15 LIBRARY-GROUP-CATALOG  
3-18 SUBROUTINES AND FUNCTIONS  
3-19 GOSUB...RETURN  
3-21 MULTIBRANCH GOSUB  
3-22 NESTING GOSUB'S  
3-23 FOR...NEXT WITH STEP  
3-24 DEF FN  
3-26 GENERAL MATHEMATICAL FUNCTIONS  
3-27 TRIGONOMETRIC FUNCTIONS

## CONTENTS CONTINUED

- 3-28 THE LEN FUNCTION
- 3-29 THE TIM FUNCTION
- 3-30 CHAIN
- 3-32 COM
- 3-34 ENTER
  
- 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 ASSIGN
- 4-12 SERIAL FILE PRINT
- 4-14 SERIAL FILE READ
- 4-16 RESETTING
- 4-17 THE TYP FUNCTION
- 4-18 LISTING CONTENTS OF A FILE
- 4-19 TERM: END-OF-FILE
- 4-20 IF END#...THEN
- 4-21 PRINT#...END
- 4-22 STRUCTURE OF SERIAL FILES
- 4-23 EXAMPLE OF SERIAL FILE MODIFICATION
- 4-27 TERM: RECORD
- 4-28 STORAGE REQUIREMENTS
- 4-29 MOVING THE POINTER
- 4-30 SAMPLE USE OF READ#M,N
- 4-31 SUBDIVIDING SERIAL FILES
- 4-32 USING THE TYP FUNCTION WITH RECORDS
- 4-33 SAMPLE OF READ#M,N AND TYP(-M)
- 4-34 HOW TO COPY A FILE
- 4-35 TERM: RANDOM FILE ACCESS
- 4-36 SAMPLE OF RANDOM FILE ACCESS

## CONTENTS CONTINUED

- 4-37 PRINTING A RECORD
- 4-38 READING A RECORD
- 4-40 MODIFYING CONTENTS OF A RECORD
- 4-41 ERASING A RECORD
- 4-43 UPDATING A RECORD
- 4-44 AN ALPHABETICALLY ORGANIZED FILE

### 5-1 SECTION V MATRICES

- 5-2 DIM
- 5-3 MAT...ZER
- 5-4 MAT...CON
- 5-5 INPUT
- 5-6 MAT INPUT
- 5-7 PRINTING MATRICES
- 5-8 MAT PRINT
- 5-10 READ
- 5-11 MAT READ
- 5-12 MATRIX ADDITION
- 5-13 MATRIX SUBTRACTION
- 5-14 MATRIX MULTIPLICATION
- 5-15 SCALAR MULTIPLICATION
- 5-16 COPYING A MATRIX
- 5-17 IDENTITY MATRIX
- 5-18 MATRIX TRANSPONSTION
- 5-19 MATRIX INVERSION
- 5-20 MAT PRINT#
- 5-21 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  
7-6 SOME EXAMPLES

8-1 SECTION VIII  
FORMATTED OUTPUT  
8-2 DEFINITIONS  
8-5 SUMMARY  
8-6 STRING FORMAT SPECIFICATIONS  
8-8 INTEGER FORMAT SPECIFICATIONS  
8-10 FIXED-POINT FORMAT SPECIFICATIONS  
8-12 FLOATING-POINT FORMAT SPECIFICATIONS  
8-14 POSITION OF THE SIGN  
8-15 GROUP FORMAT SPECIFICATIONS  
8-15 TERM: FORMAT STRINGS  
8-16 TERM: EXPRESSION LIST  
8-17 PRINT USING  
8-19 MAT PRINT USING

## CONTENTS CONTINUED

8-20 FORMAT IN A STRING VARIABLE

8-21 IMAGE

8-22 USING CARRIAGE CONTROL

8-23 NUMERICAL OUTPUT

8-25 REPORT GENERATION

8-27 FATAL ERRORS

8-28 NON-FATAL ERRORS

9-1 SECTION VIII  
FOR THE PROFESSIONAL

9-2 SYNTAX REQUIREMENTS OF TSB

9-10 STRING EVALUATION BY ASCII CODES

9-11 MEMORY ALLOCATION BY A USER

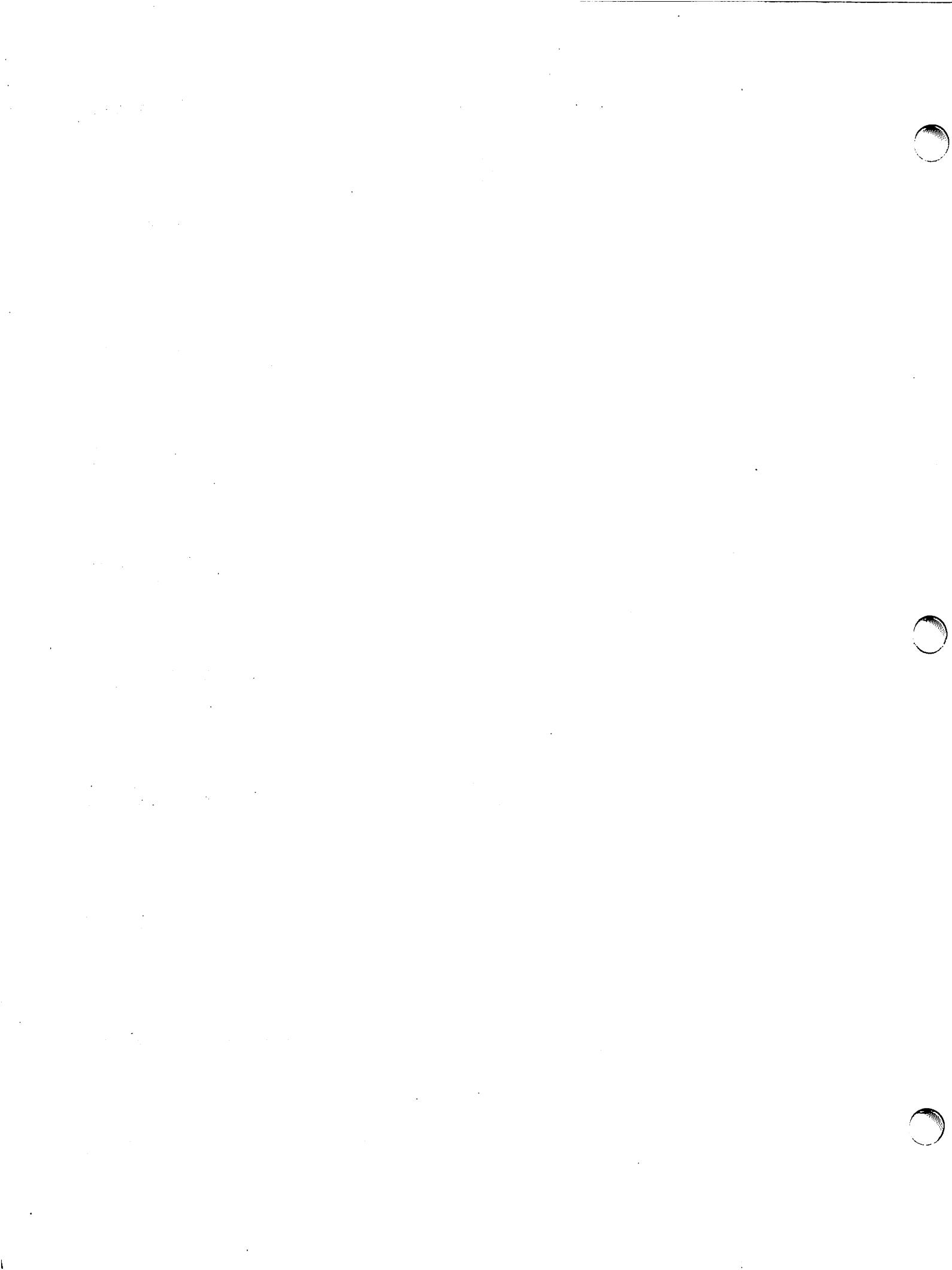
A-1 APPENDIX A  
HOW TO PREPARE A PAPER TAPE OFF-LINE

B-1 APPENDIX B  
THE X-ON, X-OFF FEATURE

C-1 APPENDIX C  
DIAGNOSTIC MESSAGES

D-1 APPENDIX D  
ADDITIONAL LIBRARY FEATURES

INDEX



**SECTION I: AN INTRODUCTION TO TSB**

**SECTION II: THE ESSENTIALS OF BASIC**

**SECTION III: ADVANCED BASIC**

**SECTION IV: FILES**

**SECTION V: MATRICES**

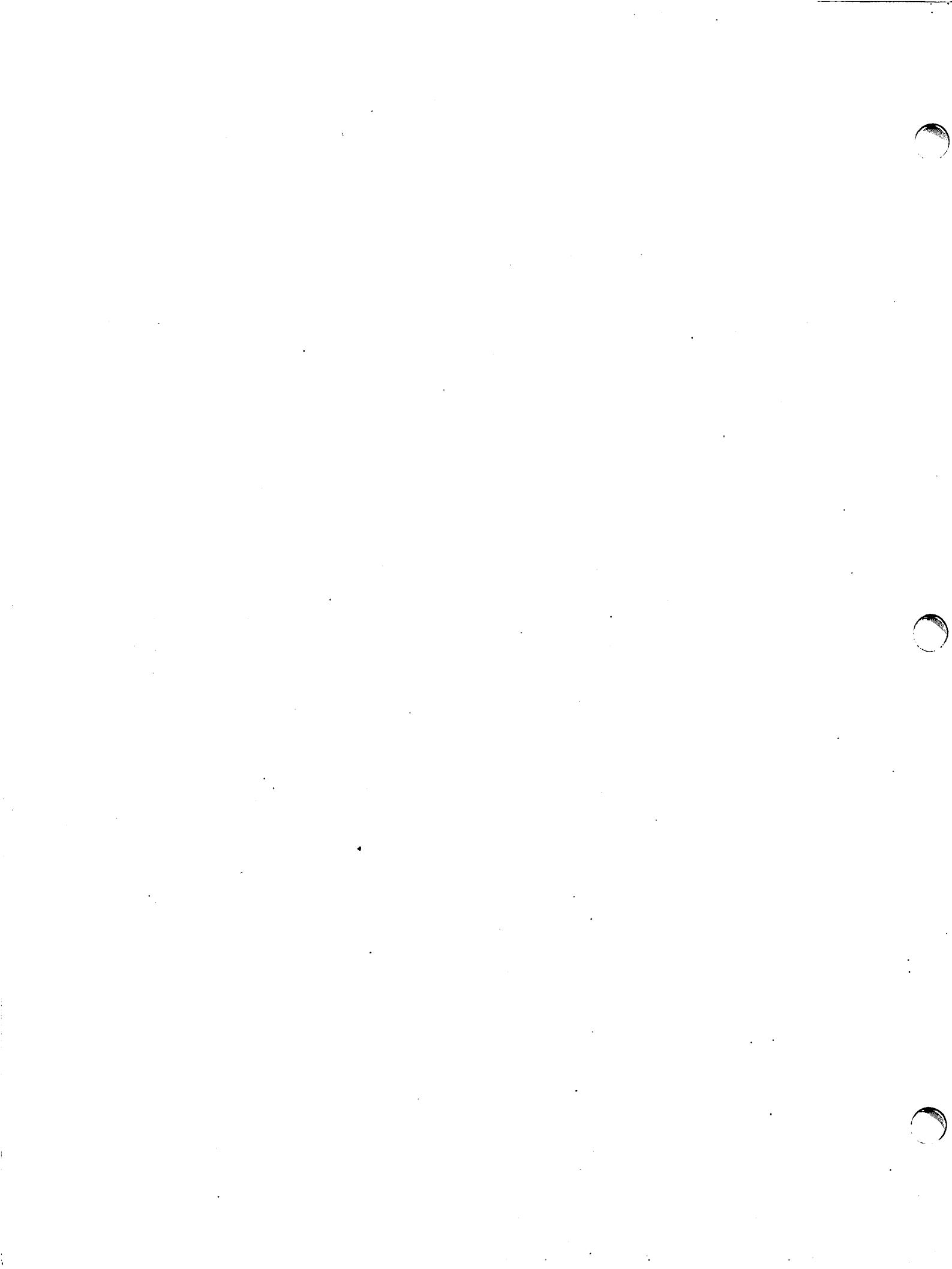
**SECTION VI: STRINGS**

**SECTION VII: LOGICAL OPERATIONS**

**SECTION VIII: FORMATTED OUTPUT**

**SECTION IX: FOR THE PROFESSIONAL**

**APPENDICES AND 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.

A time-shared computer system consists of a central computer, a system of peripheral devices at the computer site, and a number of independent terminals. The terminals, also called ports, may be connected directly to the computer through a multiplexer, or may be located remotely and connected by telephone lines.

The Hewlett-Packard 2000C Time-Shared BASIC system uses two computers -- one for actual computation and the other for controlling access to the main computer -- plus magnetic mass storage devices and other equipment used by the system operator. This system can support up to 32 terminals, all operating simultaneously. The typical user's terminal is a standard teleprinter with a paper tape punch and reader. The user can enter programs into the system either through the keyboard or through the paper tape reader; system output can be punched on paper tape as well as being typed out. The system is so designed that no user should encounter more than a few seconds delay between entering a command and receiving a response from the system, even when all terminals are busy.

The user can work in a simple interactive mode, entering and running programs and reading the results from the teleprinter, or can take advantage of the large storage capacity of the system by using library programs and by storing his own programs for later use.

Time-Shared BASIC employs two distinct languages: BASIC, one of the simpler programming languages, and a series of COMMANDS that permit the user to control system functions such as listing and running programs, storing and retrieving programs and data, and obtaining diagnostics and lists of library programs.

COMMANDS consist of three alphabetic characters: some of these commands require that parameters, such as line numbers within a BASIC program, be stated, while others permit the addition of certain parameters. The system examines the first three characters, then ignores any additional characters except a hyphen (used when additional parameters are either required or permitted). The system takes no action until the carriage return is pressed. As an aid to learning the COMMANDS, characters may be added to make the COMMAND more meaningful. For example, the log-on COMMAND "HEL" may be typed "HELLO" or the "CAT" COMMAND, requesting a program catalog, can be typed "CATALOG." COMMANDS serve the following functions:

- |                  |   |
|------------------|---|
| Logging          | -- COMMANDS used to log on and off the system.                    |
| Program control  | -- COMMANDS used to refer to the current program.                 |
| Terminal control | -- COMMANDS used to control tape reading and punching.            |
| Library access   | -- COMMANDS used to manipulate programs or files in mass storage. |
| Miscellaneous    | -- Messages to the system operator, time checks.                  |

Special (non-printing) keys on the teleprinter are used to control line spacing, return, delete lines or characters, and to terminate programs before they run to completion.

The user can communicate with the system operator by using the MESSage command described in Section II. The operator can send messages to all users or to a specified terminal.

# SPECIAL KEYS

**NOTE:** *Superscript "C" indicates a control character.  
(Hold down ctrl while typing a character.)*

| <u>KEY</u>      | <u>FUNCTION</u>  |
|-----------------|--|
| <u>ctrl</u>     | Converts normal keys to non-printing control characters.   |
| <u>alt-mode</u> |  |
| <u>esc</u>      | Deletes a line being typed.  |
| <u>break</u>    | Terminates a running program, listing, or punching   |
| C <sup>C</sup>  | Terminates an input loop (C <sup>C</sup> <u>return</u> ); causes a jump to the END statement.  |
| <u>linefeed</u> | Causes the teleprinter to advance on line.   |
| N <sup>C</sup>  | Generates a <u>linefeed</u> when used in a PRINT statement.  |
| O <sup>C</sup>  | Generates a <u>return</u> when used in a PRINT statement.  |
| <u>return</u>   | <ol style="list-style-type: none"><li>1. Must follow every command or statement.</li><li>2. Causes the teleprinter typeface to return to the first print position.</li><li>3. TSB responds with a <u>linefeed</u>.</li></ol> |
| ←               | Backspace. Deletes as many preceding characters as ←'s are typed in.   |

## USING THE TELEPRINTER TERMINAL

The terminal can be operated in either of two modes, on-line (connected to the computer) or off-line (independent of the computer). After the user has established the connection to the computer and logged in properly, the user is in contact with the computer through the Time-Shared BASIC System. The system will execute any legal command, and will detect and reject any illegal command, usually printing a message informing the user why the command was rejected.

To enter a command, type either the short form or the full form of the command; if additional parameters are *required* or *permitted*, type a hyphen, then the parameters. Terminate the command with a *return*. Some commands cause an obvious response from the system; CAT(ALOG) and LIB(RARY) and GRO(UP) all result in the printing of a list of programs. Other commands result in computer operations and the only indication at the user's terminal is the generation of a *linefeed*, indicating that the system has accepted the command and is "waiting" for another.

The teleprinter can also be used off-line to prepare paper tape. Off-line operation of the teleprinter is described in Appendix A.

# COMMUNICATING WITH THE TSB SYSTEM

## THE BASIC LANGUAGE

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

BASIC is a formal language used to communicate with the Time-Shared BASIC System.

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 occupies one teleprinter 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 system 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 system keeps them in numerical order no matter how they are entered. For example, statements are input in the sequence 30,10, 20; the system arranges them in the order: 10,20,30.

## INSTRUCTIONS (STATEMENT TYPES)

The statement gives an instruction to the TSB system (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 an 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.

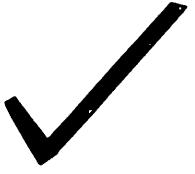
# THE FORMAT OF STATEMENTS

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

```
30 PRINT S  
30 PRINT   S  
30PRINTS
```

## 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 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 work with the TSB system, 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 system that the statement is complete. The system then checks the statement for mistakes.  
(The checking process is explained later.)

## DELETING OR CHANGING CHARACTERS

Typing the reverse arrow ( $\leftarrow$ ) key deletes the immediately preceding character. One character is deleted for each  $\leftarrow$ .

Typing:  $20\ LR\leftarrow ET\ S=10$  return

is equivalent to typing:  $20\ LET\ S=10$  return

And typing:  $30\ LET\leftarrow \leftarrow \leftarrow\ PRINT\ S$  return

is equivalent to typing:  $30\ PRINT\ S$  return

### COMMENTS

The  $\leftarrow$  character is a "shift" 0 on most terminals.

# DELETING OR CHANGING A STATEMENT

To delete the statement being typed, press the esc or alt-mode key. 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 esc key deletes  
the statement being typed: 20 LET S = esc

*NOTE: The system responds with a \ when esc is typed, like this:*  
20 LET S = \

To delete statement 5 in the  
sequence: 5 LET S =  $\emptyset$   
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 esc (or alt-mode)  
before a return prevents  
replacement of a previously  
typed statement.

For example, typing: 5 LET esc  
or: 5 esc

has no effect on the orig-  
inal 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 system skips two lines,  
separating the listing from pre-  
viously printed information.

LIST return

linefeed

linefeed

linefeed

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

linefeed

linefeed indicates that the  
listing is complete.

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-30 return

linefeed

linefeed

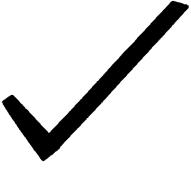
linefeed

30 PRINT S

40 GO TO 10

50 END

linefeed

•   
1. Be sure you understand the use of these features:

return to end statements  
How to delete characters  
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 to the TSB system.

# 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.

*NOTE: When the connection is through telephone lines, the user must LOG IN within a time period (nominally two minutes) specified by the operator.*

### 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 esc. When the connection is made, the computer responds with a "\.". Type any numeral plus return.

EXAMPLE:                           3 return

The system 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.

The above step is optional.

---

## YOUR ID CODE AND PASSWORD

You need your identification code and password to log in. These are assigned by the system operator. The ID code is a single letter followed by a three digit number. The password consists of one to six regular or control characters.

---

## CONTROL CHARACTERS

Control characters are non-printing. They are represented with a superscript "C" to indicate that they are control characters. 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 by  
typing:

HELLO-H200,password return

*HELLO- is a command, not a statement.  
Commands are orders to the system  
which are acted upon (executed) im-  
mediately. Unlike statements, com-  
mands do not have line numbers.*

The system acknowledges that the  
user has correctly logged in, by  
outputting three linefeeds and  
a message, if the operator has put  
a message into the system for users:

linefeed

linefeed

linefeed

MESSAGE TO USERS FROM OPERATOR

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

If there is no message, the system  
responds with a linefeed, then READY,  
indicating that it is awaiting  
input.

linefeed

READY

linefeed

BYE return

To LOG OUT, type:

The elapsed time since log in is  
then printed.

001 MINUTES OF TERMINAL TIME

# MISTAKES DURING LOG IN

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

HELLO H200,password 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 to the system.*

# ENTERING THE SAMPLE PROGRAM

The frame below shows how to enter a program. If you are not sure how the system 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 system 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 system 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 PRIMT S return

*NOTE: PRINT has been misspelled.*

The system 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 PRIMT S return

ERROR

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 system responds with three linefeed's indicating that the command is being executed.

RUN return  
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

The program is finished:

2.1 return linefeed

Log off:

? C<sup>C</sup> return

Time used is printed:

DONE

BYE return

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  
or until the user terminates the procedure.

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:

STOP

after break is pressed.

Remember that:

C<sup>c</sup> return

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

### COMMENTS

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

## HOW THE PROGRAM WORKS

Line 10 tells the system 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. When the program is run, the system will print a question mark (?) at this point and wait for input from the terminal keyboard. The ENTER statement, Section III can also be used to input data.

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 system 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. Explicit instructions must be given for each operation to be performed.*

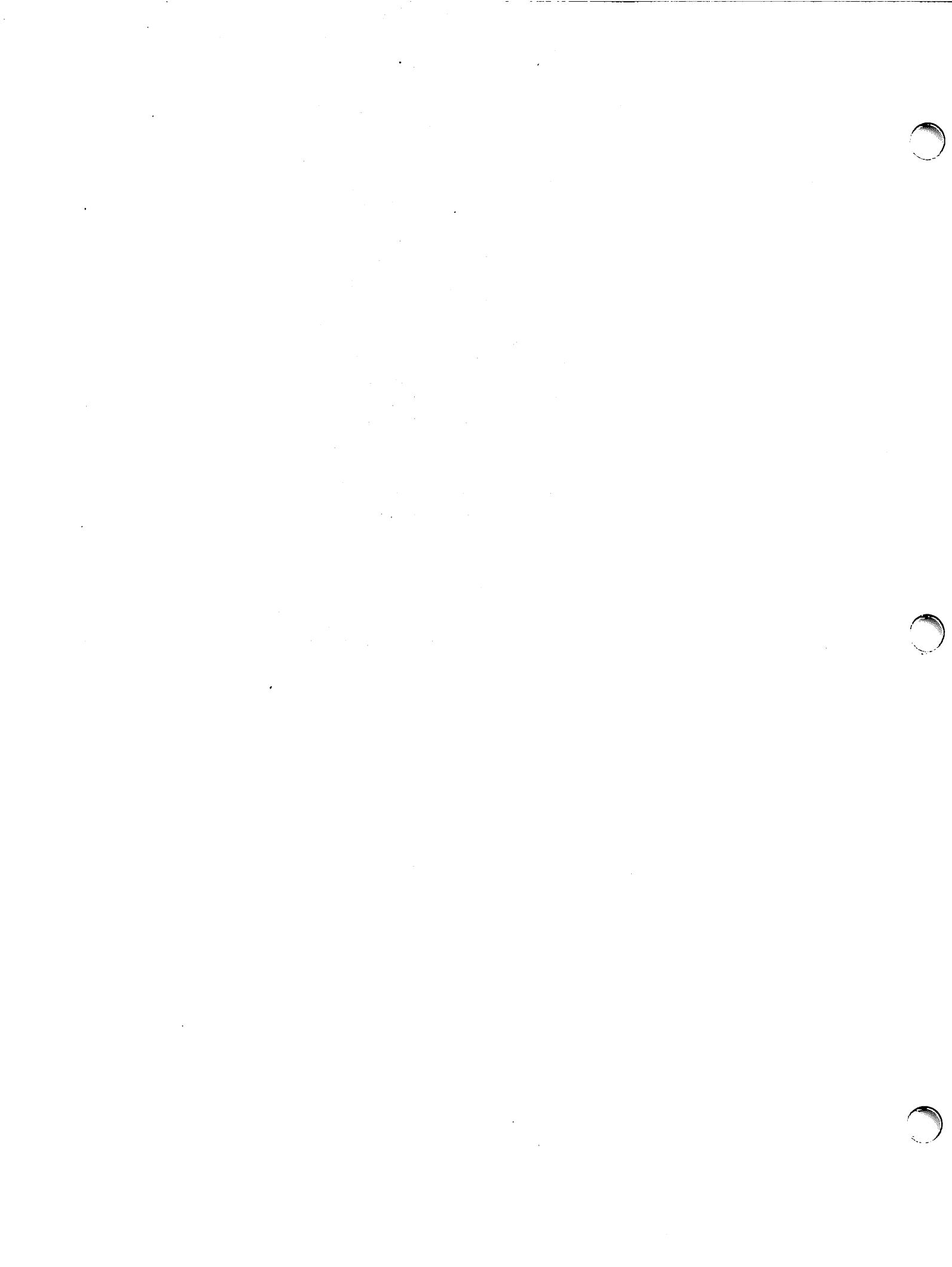
Line 40 tells the system to go to line 10 and execute whatever instruction is there:

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 type of loop is called an input loop (because the user must repeatedly input data). Each time the system prints the value of S (line 30) execution continues to line 40, returns to line 10, then prints the question mark (?) -- requesting five new inputs. The only way a user can escape from a program that is waiting for input is to type C (the character C with the CONTROL key depressed). Execution will then proceed to the last statement in the program, line 50 in this example, and the system will print DONE. When the program is actually running (computing the values or printing results) the break key is used to interrupt the program, as described previously.

Line 50 indicates 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 also allowed.

### COMMENTS

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

If the user types a BASIC statement which contains a number that is not representable in TSB, the system will output a warning and change the number in the statement to the closest representable one.

If an executing program makes a calculation which results in a non-representable number, that number will be set to the closest representable one and a warning message will be printed.

## 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 1000000 and is read:  
                  "1 times 10 to the sixth power" ( $1 \times 10^6$ ).

1.02000E+04 is equal to 10200  
1.02000E-04 is equal to .000102

### COMMENTS

"E" notation is used to print numbers greater than six digits. (See PRINT.) 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.

## 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, array and string 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:

10 LET A = B2 = C = 0

20 LET A9 = C5

30 Y = (N-(R+5))/T

40 N5 = A + B2

50 P5 = P6 = P7 = A = B = 98.6

GENERAL FORM:

LET variable = expression

variable = expression

## 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:

100 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 10 and 50 above.

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

# ARITHMETIC OPERATORS

SYMBOLS:

$\uparrow * / + -$

EXAMPLES:

40 LET N1 = X-5

50 LET C2 = N $\uparrow$ 3

60 LET A = (B-C)/4

70 LET X = ((P $\uparrow$ 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<br>110 IF A+B > C THEN 910<br>120 IF A+B < C+E THEN 920<br>130 IF C>= D*E THEN 930<br>140 IF C9<= G*H THEN 940<br>150 IF P2#C9 THEN 950<br>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:

```
60 IF A9<B1 AND C#5 THEN 100
70 IF T7#T AND J=27 THEN 150
80 IF P1 AND R>1 AND N AND V2 THEN 10
90 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 90 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 80 requires four "true" conditions before control is transferred to statement 10.

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 = Ø

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 = Ø, then NOT A = 1; if A has a non-zero value, NOT A = Ø.

The numeric values are: "true" = 1, "false" = Ø; for example, statement 65 above would print "1", since the expression NOT (X 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 Ø, it has a logical value of "false", making 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*

## COMMENTS

If two operators in an expression are on the same level, the order of execution is left to right within the statement.

|                        |                  |                             |
|------------------------|------------------|-----------------------------|
| $5 + 6*7$              | is evaluated as: | $5 + (6 \times 7)$          |
| $\frac{7}{14} * 2 / 5$ | is evaluated as: | $\frac{(7/14) \times 2}{5}$ |

A MIN B MAX C MIN D is evaluated as:

((A MIN B) MAX C) MIN D

Operations enclosed in parentheses are performed before any operations outside the parentheses.

When parentheses are nested, operations within the innermost pair of parentheses are performed first.

# STATEMENTS

Be sure you know the difference between statements and commands.

Statements are instructions to the system. 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 system 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:

## Statements

LET

PRINT

INPUT

## Commands

HELLO

BYE

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:

10 LET A = 5.02  
20 X = Y7 = Z = 0  
30 B9 = 5\* (X+2)  
40 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 20 in the example above shows the use of an assignment to give the same value (0) to several variables. This is a valuable feature for initializing variables in the beginning of a program.

# REM

EXAMPLES:

10 REM--THIS IS AN EXAMPLE  
20 REM: OF REM STATEMENTS  
30 REM----//\*\*\*\*!!  
40 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; how-  
ever, 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  $X < 10$ , 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.  $<=$ ,  $>=$ , 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  $X \geq 10$ , 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 no. 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.

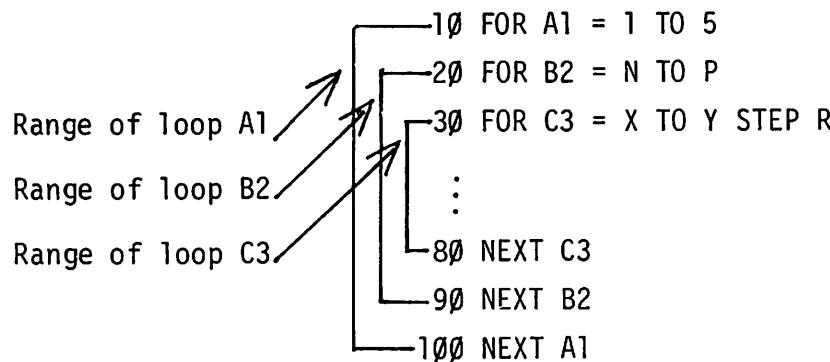
For further details on the STEP feature, see "FOR...NEXT with STEP" in Section III.

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

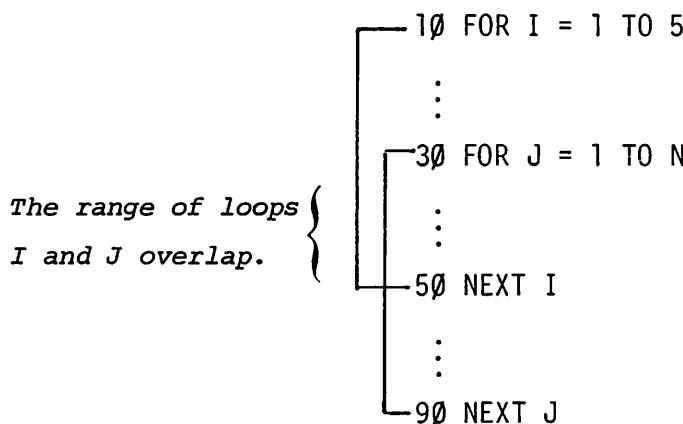
# NESTING FOR...NEXT LOOPS

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 IV.

# 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

??.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:

10 INPUT A, B2, C5, D, E, F, G,

and

20 INPUT X

each cause a single "?" to be printed. Note that the "?" generated by statement 10 requires seven input items, separated by commas, while the "?" generated by statement 20 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 this section for variations on output formats.

# PRINT

## EXAMPLE

```
10 LET A=B=C=D=E=F=G=14
20 LET D1=E9=20
30 PRINT A,D1,B,C,E9
40 PRING 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 REM: "PRINT" WITH NO OPERAND CAUSES THE TELEPRINTER TO SKIP A LINE.
90 PRINT "'A' DIVIDED BY 'E9' =";A/E9
100 PRINT
110 PRINT "11111","22222","33333","AAAAAA","BBBBBB","CCCCC"
120 PRINT "11111","22222","33333","AAAAAA","BBBBBB","CCCCC"
130 PRINT A,B,C,D,D1,E,F,E9,G
140 PRINT A;B;C;D;D1;E;F;F;E9;G
150 PRINT
160 PRINT TAB(8);"CARRIAGE";SPA(5);"CONTROL";LIN(2);"FUNCTIONS"
170 END
```

## RESULTS-

### RUN

```
14      20      14      14      20
1      20.05
```

NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE  
VALUE IN THE SAME STATEMENT.

'A' DIVIDED BY 'E9' = .7

```
11111      22222      33333      AAAA      BBBB
CCCCC
111112222233333AAAAABBBBBCCCCC
14      14      14      14      20      14      14      20
14      14      14      14      20      14      14      14
```

CARRIAGE CONTROL

FUNCTIONS

DONE

## 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 and/or TAB, LIN, and SPA

or

statement number PRINT

### PURPOSE

Causes the value(s) of the expression(s) to be output to the teleprinter or terminal device.

Causes the teleprinter to skip a line when used without an operand. Causes text within quotes to be printed literally.

### COMMENTS

Note the effects of , and ; on the output of the sample program. If a comma is used to separate PRINT operands, up to five fields will be printed per teleprinter line. These five fields begin in columns 0, 15, 30, 45, and 60. If semicolon is used, up to twelve "packed" numeric fields will be output per teleprinter line; the exact number depends on the size of each numeric field. If semicolons are used between text in quotes, it is possible to print a full 72 characters on a line.

## PRINT , CONTINUED

A carriage return and linefeed are output after the execution of any PRINT statement unless the list of items to be printed is terminated by a comma or semicolon, in which case the next PRINT statement will begin on the same line.

Values output by PRINT statements are in one of four possible numeric formats, depending on the value. These values and their formats are:

| <u>Value</u>   | <u>Field</u>  | <u>Examples</u>  |
|--|---|------------------|
| -999 $\leq$ integer $\leq$ 999                           | $\pm$ ddd. <u>  </u>                                  | 733<br>-214      |
| -32767 $\leq$ integer $\leq$ -1000                       | $\pm$ ddddd. <u>  </u>                                | -1234            |
| 1000 $\leq$ integer $\leq$ 32767                         |   | 7515             |
| all other integers                                       | $\pm$ dddddd. <u>  </u>                               | 131072.          |
| .000001 $\leq$ and all $\leq$ 999999.5<br>reals in range | (one d is ".")<br>trailing zeroes<br>are suppressed.) | 14.6<br>-.003456 |
| All numbers n such that<br>$n < .000001$                 |   | 1.97343E+06      |
| 999999.5 $\leq$ n  | $\pm$ d.dddddE $\pm$ dd. <u>  </u>                    | -6.91112E+15     |

Each "d" represents one decimal digit; each " $\pm$ " means the sign if negative, a space if positive; each "  " means a space; each  $\pm$  means the sign. An example of these formats is the following program, which prints the powers of 2 from -5 to 30.

EXAMPLE:

```
10 FOR N=-5 TO 30
20 PRINT 2^N;
30 NEXT N
40 END
```

RUN

|             |             |       |             |        |             |         |             |      |      |
|-------------|-------------|-------|-------------|--------|-------------|---------|-------------|------|------|
| .03125      | .0625       | .125  | .25         | .5     | 1           | 2       |             |      |      |
| 4           | 8           | 16    | 32          | 64     | 128         | 256     | 512         | 1024 | 2048 |
| 4096        | 8192        | 16384 | 32768.      | 65536. | 131072.     | 262144. |             |      |      |
| 524288.     | 1.04858E+06 |       | 2.09715E+06 |        | 4.19430E+06 |         | 8.38861E+06 |      |      |
| 1.67772E+07 | 3.35544E+07 |       | 6.71089E+07 |        | 1.34218E+08 |         | 2.68435E+08 |      |      |
| 5.36871E+08 | 1.07374E+09 |       |             |        |             |         |             |      |      |

DONE

## PRINT , CONTINUED

Insertion of the special functions TAB, SPA, and LIN into the output list provides carriage control:

- |                           |   |
|---------------------------|---|
| TAB ( <i>expression</i> ) | Causes the carriage to move to the specified print column (071). No action is taken if the move would be to the left. The carriage moves to the beginning of the next line if expression >71.                                       |
| SPA ( <i>expression</i> ) | Causes carriage to skip specified number of spaces ("print that number of blanks"). A negative expression does nothing. If more spaces are requested than remain in the line, the carriage moves to the beginning of the next line. |
| LIN ( <i>expression</i> ) | Generates a <u>carriage return</u> and the specified number of <u>linefeeds</u> . If the expression is negative, then no <u>carriage return</u> is generated. LIN (0) produces a single <u>carriage return</u> .                    |

0<sup>C</sup> printed in a character string causes a carriage return to be output instead.

N<sup>C</sup> printed in a character string causes a linefeed to be output instead.

The PRINT USING statement, which provides increased output formatting capabilities, is described in Section VIII.

# 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, Ø IF NO

BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.

YOUR REPLY? 2 return

TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø 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 system, 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 or permitted after a command, a hyphen "-" must be included. For example, when logging in:

HEL-H200,SEC<sup>C</sup>C<sup>C</sup>R<sup>C</sup>E<sup>C</sup>T 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-D007, POS<sup>C</sup>T return

GENERAL FORM:

HELLO- IDcode , password

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 operator.

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

# BYE

EXAMPLE:

BYE return

ØØ9 MINUTES OF TERMINAL TIME

GENERAL FORM:

BYE

## 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 return

ECHO-ON return

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 return

or

RUN- 300 return

GENERAL FORM:

RUN

RUN- statement number

## 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, INPUT, ENTER, 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 return

# LIST

EXAMPLE:

LIST return  
LIST -100 return  
LIST -100, 200 return

GENERAL FORM:

LIST  
LIST- statement number  
LIST- statement number , statement number  
LIST- , statement number  
LIST- statement number , statement number , P  
LIST- P  
LIST- statement number , P  
LIST- , statement number , P

## 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.

When "P" is specified, the listing is spaced for cutting into 11-inch sheets sized for binding or filing. "P" must be the final parameter, and must be preceded by a comma if it follows other parameters.

## COMMENTS

A listing may be stopped by pressing the break key. Library programs designated "RUN ONLY" (protected) by the System Master or Group Master cannot be listed. LIST 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

REN-10, 10, 50, 100 return

GENERAL FORM:

REN

or

REN-number assigned to first statement

or

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

or

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

starting statement number, ending statement number

or

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

starting statement number

## PURPOSE

Renumerates statements in the current program.

## RENUMBER , CONTINUED

### COMMENTS

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

Starting statement number and ending statement number refer to line numbers in the original program at which the renumbering is to start and end.

If ending statement number is not specified, it is assumed to be the last statement in the program.

If starting statement number is not specified, it is assumed to be the first statement in the program.

If both starting and ending statement numbers are omitted, the entire program is renumbered.

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

If no parameters are stated, the entire program is renumbered starting with statement 10 at intervals of 10.

RENUMBER can not be used to change the order of statements in a program.

Any parameter may be omitted, but all parameters following it must also be omitted.

Numbers or text contained in REM and PRINT statements or in the expression list of PRINT USING 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, XPUNCH,  
CATALOG, GROUP 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 mes-  
sage STOP is output.

Pressing break will not terminate the program if  
it is awaiting input from the keyboard while exe-  
cuting an INPUT or ENTER statement. In this case  
the only means of ending the program is typing:

C<sup>c</sup> return

which produces the DONE message.

break will not delete a program. Type RUN to  
restart the program. (See also COM, Section III.)

# PUNCH AND XPUNCH

## EXAMPLES:

PUNCH return  
PUN- 100, 200 return  
PUN- 100, 200, P return  
PUN-65 return  
PUN-, 300 return  
XPUNCH return  
XPU- 65, P return  
XPU- P return

## GENERAL FORM:

PUN  
PUN- statement number  
PUN- statement number, statement number  
PUN- statement number, statement number, P  
PUN-, statement number  
PUN- P  
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).

## PUNCH AND XPUNCH, CONTINUED

### 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 system that following input  
(a group of BASIC statements) 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 return

GENERAL FORM:

KEY

## PURPOSE

Informs the system 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 indicating syntax errors in BASIC statements were generated during tape input.

# TIME

EXAMPLE:      TIME return  
CONSOLE TIME = 12 MINUTES. TOTAL TIME = 1193 MINUTES.

GENERAL FORM:    TIME

## PURPOSE

Informs user 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.

# MESSAGE

EXAMPLE:           MES-PLEASE SANCTIFY PROGRAM "DUMMY", USER J122. return

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 number.

## COMMENTS

Can be used to request information from the system operator, or to have programs sanctified, desecrated, copied, bestowed, or loaded from or dumped to magnetic tape (see Appendix D)

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 sub-program.

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

Examples of routines and subroutines are given in this section.

# TERM: ARRAY

DEFINED IN TSB AS: An ordered collection of numeric data.  
A single program can have up to about  
4900 total array elements (numeric  
values).

## COMMENTS

In BASIC a simple variable is defined by a single letter or a letter followed by a numeral. A and A1 are simple variables. Subscripted variables define elements in an array. A<sub>1</sub>, written A(1), is the first element in the single-dimensioned array called A. In the example below, the value would be 5.0:

### EXAMPLE:

#### Array A

| <u>Element</u> | <u>Value</u> |
|----------------|--------------|
| 1              | 5.0          |
| 2              | 3.2          |
| 3              | 1.1          |
| 4              | 0.3          |

Two-dimensioned array elements are defined by a double subscript, referring to a row and column position in an array. Element B(1,3) in the following example has the value appearing in the first row, third column. In this case the value is 4.

#### Array B

|       | Column 1 | Column 2 | Column 3 |
|-------|----------|----------|----------|
| Row 1 | 6        | 5        | 4        |
| Row 2 | 3        | 2        | 1        |
| Row 3 | Ø        | 9        | 8        |

## **ARRAY, CONTINUED**

Array B is a three-by-three array. Arrays need not be square.

If an array has more than ten elements, a DIM (dimension) statement is required. The DIM statement is described in Section V, which covers matrices; a matrix is a special form of array.

## 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, except when the string is input using an ENTER statement, described later in this section.

## 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

100 LET Y = SQR(X)

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

## TERM: WORD

DEFINED IN TSB AS: The equivalent of approximately two BASIC characters or one-half of a number.

### COMMENTS

The term "word" is used to define the basic unit of computer storage. The TSB system operates on computers having a word structure of 16 binary bits. Each character in BASIC occupies 8 bits of computer storage; each number (when used in computation) occupies 32 bits. A numeral that appears in a literal string (Section VI) is not used for computation, and is considered to be a character.

Therefore, two characters will fit into one computer word, while one number will require two computer words. Actually, the TSB system requires a few additional computer words of storage, so programs and files will require slightly more storage than one word for each two characters or two words for each number. Each user has a working area of 10,000 words. The user need not normally be concerned about computer words.

# 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 to rewrite it!

# LENGTH

EXAMPLES:

LENGTH return

3172 WORDS

LEN return

151 WORDS

GENERAL FORM:

LEN return

## PURPOSE

Prints the number of 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 over 10,000 words (20,000 characters or 5,000 numbers). LEN is a useful check on total program length when writing Long programs. During execution, programs have temporary tables, buffers, etc. which require additional storage space. This larger total length is not permitted to exceed the user's working area. See MEMORY ALLOCATION BY A USER, Section IX.

## NAME-

EXAMPLE:

NAME-PROG.1 return

NAM-ADDER return

NAM-MYPROG return

GENERAL FORM:

NAME-Program name of 1 to 6 characters

or

NAM-Program name of 1 to 6 characters

### PURPOSE

Assigns a name to the program currently being accessed from the tele-printer.

### COMMENTS

The first character of the program named may not be \$ or \*. These symbols are used to access the System Library (\$) and the Group Library (\*). The comma (,) may not be used in the name of a program.

The program name must be used in certain TSB operations (see the SAVE, CSAVE, KILL, GET, and APPEND commands in this section).

## SAVE- AND CSAVE-

|               |  |
|---------------|--|
| EXAMPLES:     | SAVE <u>return</u><br>SAV <u>return</u><br>CSA <u>return</u> |
| GENERAL FORM: | <u>CSAVE</u> or <u>CSA</u><br><u>SAVE</u> or <u>SAV</u>      |

### PURPOSE

Saves a copy of the current program in the user's private library. (CSA stores the program in semi-compiled form so that it will CHAIN more quickly. See CHAIN.)

### COMMENTS

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

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

KILL-SAMPLE return (Deletes the stored version)

linefeed

NAME-SAMPLE return (Names the current program)

linefeed

SAVE return (Saves the current program, named SAMPLE)  
linefeed

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

# GET-, GET-\$, AND GET-\*

## EXAMPLES:

GET-PROGRAM return  
GET-MYPROG return  
GET-\$PUBLIC return  
GET-\$NAMES return  
GET-\*DATES return

## GENERAL FORM:

GET- name of a program in user's library  
GET-\$ name of system library program  
GET-\* name of group library program

## 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.

GET-\* retrieves the specified program from the group library.

## COMMENTS

GET- performs an implicit SCRATCH. The program that was the current program prior to using GET- can not be recovered from the system unless it was previously SAVed or CSAVed.

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

# KILL-

EXAMPLE:

KILL-PROG12 return

KIL-EXMPL3 return

KIL-FILE10 return

GENERAL FORM:

KILL- program or file to be deleted

or

KIL- program or file to be deleted

## 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 can not be recovered from the system unless 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 return

APP-MYPROG return

APPEND-\$PUBLIC return

APP-\$SYSLIB return

APP-\*GPROG return

### GENERAL FORM:

APPEND-program name

or

APP-program name

or

APP-\$system library program name

or

APP-\*group library program

### PURPOSE

Retrieves the named program from the user's own library, or the group or public libraries 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 \$ preceding system library program names is needed to APPEND them; the \* is needed to APPEND group library programs. For details, see LIBRARY in this section.

## DELETE-

### EXAMPLES:

DELETE-27 return

DEL-27, 50 return

### GENERAL FORM:

DEL-statement number at which deletion starts  
or

DEL-statement no. at which deletion starts , statement no. at which deletion ends

### PURPOSE

DEL-statement number erases the current program statements after and including the specified statement. DEL-1 has the same effect as SCRATCH.

DEL-statement number, statement number deletes all statements in the current program between and including the specified statements.

### COMMENTS

It is sometimes useful to SAVE or PUNCH the original version of a program which is being modified, before using the DELETE statement.

Deleted statements are not recoverable.

# LIBRARY - GROUP - CATALOG

EXAMPLES:

## LIBRARY

| NAME   | LENGTH | NAME | LENGTH | NAME | LENGTH | NAME   | LENGTH |      |        |   |      |
|--------|--------|------|--------|------|--------|--------|--------|------|--------|---|------|
| AAA    | FPS    | 2    | AB     | F    | 230    | BAA    | F      | 2    | BAB    | P | 13   |
| BAC    |        | 6    | BAD    | C    | 18     | BB     | F      | 46   | BBA    | F | 2    |
| BBB    | F      | 46   | BFILE  | F    | 128    | BUDGE  |        | 12   | BUDGET |   | 3431 |
| BUDGEU |        | 12   | C      | F    | 31     | C.R    | S      | 1220 | CB     | F | 230  |
| CC     | F      | 31   | CCC    | F    | 31     | D      | F      | 100  | F1     | F | 64   |
| FFF    | F      | 34   | GARY1  |      | 95     | GARY2  |        | 83   | GARY3  |   | 188  |
| GOGO   | P      | 151  | GT     | F    | 128    | STRING | F      | 1    | XY     | F | 256  |

## GROUP

| NAME   | LENGTH | NAME | LENGTH | NAME | LENGTH | NAME   | LENGTH |      |        |   |     |
|--------|--------|------|--------|------|--------|--------|--------|------|--------|---|-----|
| B      | F      | 30   | B1     | F    | 128    | B2     | F      | 128  | BLOCK2 | F | 128 |
| CAICAL |        | 4004 | CALC   | C    | 4081   | MBLOCK |        | 1655 | SP1    | F | 400 |

## CATALOG

| NAME   | LENGTH | NAME | LENGTH | NAME | LENGTH | NAME | LENGTH |     |      |  |   |
|--------|--------|------|--------|------|--------|------|--------|-----|------|--|---|
| BLOCK2 | F      | 128  | CHECK  | C    | 55     | SP1  | F      | 800 | TEST |  | 3 |

GENERAL FORMS:

LIBRARY return

or

LIB return

GROUP return

or

GRO return

CATALOG return

or

CAT return

# LIBRARY - GROUP - CATALOG, CONTINUED

## PURPOSE

To print an alphabetic listing of programs and files stored by the system. LIBRARY or LIB produces a list of system programs and files. GROUP or GRO produces a list of group programs and files. CATALOG or CAT produces a list of programs and files stored in the user's own program library.

## COMMENTS

Code letters preceding LENGTH indicate

F - the entry is a file.

C - the entry is a program in semi-compiled form.

If neither a C nor an F appears, the entry is a program.

P - The entry is "protected," may be either a program  
or a file.

S - the entry is "sanctified," may be either a program or  
a file. (See Appendix D.)

Code letters may be combined as in the first entry, AAA in the LIBRARY listing.

Length is given in words for programs, records for files.

Protected system or group programs may be run but not listed, saved or punched. Protected system or group files may not be accessed by other users. A user's own programs may not be protected, but may be sanctified by the operator.

## LIBRARY - GROUP - CATALOG , CONTINUED

Each user has access to the three libraries described. He has complete control over his own library, using any of the commands used to store, delete, or retrieve programs and files.

The system library is under the control of the System Master, user A000. Only the System Master (actually any user with access to the password for IDcode A000) can enter programs or files into the system library, or delete programs and files from the system library.

Each user is part of a group, all having IDcodes with the same letter and same first digit. The user whose IDcode ends in 00 is the group librarian, or Group Master. The Group Master is responsible for maintaining the group library, entering and deleting programs in the same manner as the System Master controls the system library.

The System Master and all Group Masters have the responsibility of controlling access to their libraries. Regular users can not make entries to, deletions from, or changes to either the system library or their group library. The System Master and all Group Masters have access to special commands called PROTECT, which makes specified programs available on a run-only basis and files unavailable to regular user, and UNPROTECT, which reverses the procedure. These special commands are described in the 2000C Operator's Guide.

A user can call a program from the system library by typing GET-\$, followed by the program name exactly as it appears in the LIBRARY, or append the program by typing APP-\$ followed by the program name. GET-\* and APP-\* are used to access group programs.

Files are accessed with the FILES statement, described in Section IV.

Any of these listings may be terminated by pressing the break key.

The system prints an error message if the user attempts to access a non-existent program, list or punch or save a protected program, or GET or APPEND a file.

## 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, SPA, LIN, 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.

## **GOSUB...RETURN, CONTINUED**

### 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. There may be more than one RETURN per GOSUB.

Variables have the same meaning as in the main program.

## 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 logically to a level of nine. More than nine exits without a return may cause an error message.

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:

60 DEF FNA (B2) = A<sup>2</sup> + (B2/C)

70 DEF FNB (B3) = 7\*B3<sup>2</sup>

80 DEF FNZ (X) = X/5

GENERAL FORM:

statement no. DEF FN single letter A to Z ( simple var. ) = expression

## PURPOSE

Allows the programmer to define functions.

## COMMENTS

The simple variable is a "dummy" variable whose purpose is to indicate where the actual argument of the function is used in the defining expression. After a function has been defined, the value of that function is referenced whenever the function is used by the programmer. For example, in this sequence M is a dummy variable:

```
10 LET Y = 100
20 DEF FNA (M) = M/10
30 PRINT FNA (Y)
40 END
RUN
10
```

When FNA (Y) is called for in statement 30, the formula defined for FNA in statement 20 is used to determine the value printed.

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

## **DEF FN, CONTINUED**

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

10 DEF FNA (Y) = FNB (X)  
20 DEF FNB (X) = FNA (Y)

cause 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 (0)>=.5 THEN 900  
662 IF INT (R) # 5 THEN 910  
672 PRINT SQR (X); LOG (X)
```

## PURPOSE

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

- |                           |  |
|---------------------------|--|
| 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<br>(in statement 642 above, $e^N$ )                           |
| INT ( <u>expression</u> ) | the largest integer $\leq$ the expression (INT (-3.5) would result in -4)  |
| LOG ( <u>expression</u> ) | the logarithm of the expression to the base e  |
| RND ( <u>expression</u> ) | a random number between 0 and 1  |
| SQR ( <u>expression</u> ) | the positive square root of the positively valued expression   |
| SGN ( <u>expression</u> ) | returns: a 1 if the expression is greater than 0, a 0 if the expression equals 0, a -1 if the expression is less than 0. |

## COMMENTS

All these functions may be used as expressions or as parts of expressions. LOG and SQR expressions must have a positive value or a terminal error will occur. A sequence of random numbers generated by RND is repeatable if it follows a call to RND with a given negative argument.

# 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 (in radians) of the expression.

## COMMENTS

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

The expressions (arguments) for SIN, COS, and TAN are interpreted as angles measured in radians. ATN returns the angle in radians.

## 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:

```
20 CHAIN "PROG2"  
50 CHAIN V$  
97 CHAIN "----", A  
150 CHAIN "MELVIN", 80  
200 CHAIN N$,Q+14  
230 CHAIN A$,110
```

## GENERAL FORM:

statement number CHAIN "character string"  
or  
statement number CHAIN string variable  
or  
statement number CHAIN "character string" , expression  
or  
statement number CHAIN string variable , expression

## PURPOSE

To link programs together. "Character string" or string variable specifies a program in the user's own library, the group library or the system library, which is retrieved (replacing the current program) and run.

## COMMENTS

Strings and string variables are described in Section VI. As applied to the CHAIN statement, "character string" is the name of a program in one of the libraries; string variable is an alphabetic character followed by a \$ that leads to a character string that is the name of a program. Expression is a line number in the named program. In the above examples lines 20, 97, and 150 contain character strings. The other examples contain string variables.

## CHAIN, CONTINUED

If the first character of the program name, however defined, is \$, the system will search the system library; if the first character is \*, the system will search the user's group library. If the first character is neither \$ or \*, the system will search the user's own library. Note that the \$ has different meanings as the first character in a program name and when used to define a string variable.

If expression is not specified, the program will be retrieved from the proper library and executed normally -- examples 20 and 50. Expression may be an actual line number as in examples 150 and 230, may be a variable as in example 97, or may be computed as in example line 200.

In any of the above cases common storage is allocated. (See COM.) Before execution can begin, the program chained to must be compiled. Programs which are often chained to should be stored in semi-compiled form by use of the CSAVE command. This significantly reduces the time required to execute CHAIN statements.

Execution of the CHAIN statement can produce the same errors produced in executing the GET command. Such errors terminate execution of the program attempting the chaining, which will remain as the current program, with its common area (if any) intact.

# 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

To designate data that can be passed between two or more programs without intermediate storage. A number of programs may be run sequentially, all accessing and possibly changing data in the common area.

## 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. If the COMMON area in one program is smaller than that in another program to be run sequentially, only the common variables in the smaller area will be preserved.

## 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 uses them. 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. When a program with a common area terminates (whether normally, or because of an execution error or because the user presses break) the variables in common storage retain their values and will remain available until the user GETs a program with a different common area or enters a BASIC statement.

### EXAMPLES

|                                |                |                           |
|--------------------------------|----------------|---------------------------|
| 10 COM A,B,C,Q\$(63),F(3,6),S1 | (In program A) | All variables in common   |
| 10 COM J,K,L,C\$(63),C(3,6),V  | (In program B) |                           |
| 10 COM A,B,C,Q\$(63),F(3,6),S1 | (In program A) | Three variables in common |
| 10 COM H,N,M,O                 | (In program B) |                           |
| 10 COM A,B,C                   | (In program A) | No variables in common    |
| 10 COM S\$(45),A,B,C           | (In program B) |                           |
| 10 COM A,B,C                   | (In program A) | All variables in common.  |
| 10 COM V                       | (In program B) |                           |
| 30 COM B,C                     |                |                           |

# ENTER

EXAMPLES:

100 ENTER #V  
200 ENTER A,B,C\$  
300 ENTER #V,K1,K2,K3  
400 ENTER 25,L,Q

GENERAL FORM:

statement number ENTER # variable 1  
statement number ENTER expression, variable 2, variable 3  
statement number ENTER # variable 1, expression, variable 2, variable 3

## PURPOSE

Allows the program to limit the time allowed for run-time data input, to check the actual time taken to respond, to read in one string or numeric variable, to determine whether the input is of the correct type, and/or to determine the current user's terminal number.

## COMMENTS

The form ENTER # sets variable 1 to the terminal number (between 0 and 31) of the user.

Expression sets the time limit; it must have a value between 1 and 255 seconds. Timing starts when all previous statements have been executed and all printing at the user terminal is completed.

Variable 2 returns the approximate time the user took to respond. If the user's response was of the wrong type, the value is the negative of the response time. If the user failed to respond in time, the value is set to -256.

## ENTER, CONTINUED

Variable 3, the data input variable, may be either a numeric or a string variable. A character string being entered should not be enclosed in quotes, but may contain quotes, leading blanks and embedded blanks. Only one data item can be entered per ENTER statement.

The ENTER statement differs from the INPUT statement in that a "?" is not printed on the user terminal, and the TSB System returns to the program if the user does not respond within a specified time limit. Also, the system does not generate a linefeed after the user types return.

A carriage return is a legitimate input to a string.

A string that is too long to be assigned to a requested string variable is truncated on the right.



## 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, any file can be used as a random access storage device. 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 record to a maximum determined by the device used to store them. The maximum size for files that are to be SANCTIFIED is 32 records. (See Appendix D.) A record contains between sixty-four and 256 16-bit words.

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

Each program must declare its files with a FILES statement before it can access them. Each program can access up to 16 different files at one time. Files being accessed by a program can be changed by use of the ASSIGN statement.

For each file declared in the program, there is a file pointer that keeps track of the item in the file currently being accessed by that program. The RUN command causes all these pointers to be reset to the beginning of the file. The ASSIGN statement repositions the pointer to the beginning of a specified file. As the program reads or writes on a file, the pointer for the file is moved through the file.

# SERIAL FILE ACCESS

This 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 resets the pointer to the beginning of the file and reads through the items until it comes to the desired item. There is only one pointer for each file. When the pointer is repositioned by either a READ or a PRINT statement, it remains pointing to the next item in the file until it is repositioned by another file control statement.

## SAMPLE SERIAL FILE ACCESS

|                      |   |
|----------------------|---|
| OPEN-GHIJK,50        | The OPEN command creates a new file. GHIJK is the name of the file. The file is 50 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 in the file.

```
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.

## OPEN-

EXAMPLES:                   OPEN-FILE27, 20, 64 return

                         OPEN-SAMPLE, 128 return

GENERAL FORM:

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

OPE- 1 to 6 character file name , number of records in file , record size

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

### PURPOSE

Creates a file with a specified number of records of a specified size, 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 system library files can be read by all users, and unprotected group files can be read by all members of the group.) 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 determined by the peripheral devices on the system, the amount of unused storage, and the user's personal storage limit.

The size of a record must be between 64 and 256 words. If not specified, the system assumes 256 words. In any case, each record consumes 256 words of system storage.

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:

**LIBRARY 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**

The OPEN command marks each record of the new file as empty. If the system is heavily loaded, this process could take several minutes for very large files.

## KILL-

EXAMPLE:

KILL-NAMEXX return

KIL-EXMPL~~E~~ return

KIL-FILE10 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, AND, SQRT, NAMES

20 FILES \*GRP, FILE27, SAMPLE

30 FILES MATH, \$DATA, \*, \*

## 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

Up to four FILES statements can appear in a program, but only 16 files total can be declared (duplicate entries are legal). The files are assigned numbers (from 1 to 16) in the order they are declared in the program. In the EXAMPLES above, MATH is file #1, FILE27 is #7 and DATA is #10.

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 may be added any time before running the program.

## FILES, CONTINUED

Public or group library files to be read (they cannot be written on) must also be declared in a FILES statement but with a \$ or \* preceding the file name. DATA is a public file in the example; GRP is a group file. When \* is used without a program name as one of the arguments in a FILES statement, the position occupied by the \* symbol is reserved for a file to be specified later by an ASSIGN statement. ASSIGN statements are described on the following page.

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.

# ASSIGN

## EXAMPLES:

```
20 ASSIGN A$, 3, B1, C$  
30 ASSIGN "NEWFL", S2, J  
40 ASSIGN "$F2", 6, C, "AX1532"
```

## GENERAL FORM:

statement number ASSIGN file name, file number, return variable, mask  
statement number ASSIGN file name, file number, return variable

## PURPOSE

To change the file referred to by a specified file number during the execution of a program

## COMMENTS

The parameters of an ASSIGN statement are:

### file name

The name of a file -- a literal string of up to six characters (seven if the first character is \$ or \*) enclosed in quotes or a string variable leading to a literal string. The symbol \$ as a first character indicates a system file; \* as a first character indicates a group file.

### file number

A number, variable or expression whose value is between 1 and 16, indicating a file position. The file number should not exceed the number of files declared in the FILES statements of the program.

### return variable

One of the following values will be returned to this variable when the statement is executed, depending upon the outcome of the execution:

0 - the file is available for reading and writing.

1 - the file is available on a read-only basis because it is being accessed by another terminal. For users A000 through A999, a return code of 1 indicates only that the named file is being accessed by another terminal. The file is still available for reading and writing.

## ASSIGN, CONTINUED

- 2 - the file is available on a read-only basis because it is a system library or group library file.
- 3 - the requested file does not exist or it is protected (and the user attempting to ASSIGN it is not the owner).
- 4 - the file number in the ASSIGN statement is out of range; it does not correspond to one of the positions reserved by the FILES statements.
- 5 - the requested file has records which are larger than those of the file previously in this position.

If the value given to the return variable is 3, 4, or 5, any access to the requested file will cause a terminal error. If the returned value is 2, any print attempt to the file will cause a terminal error. If the returned value is 1, a print attempt by any user other than AXXX users will cause a terminal error.

### mask

An optional parameter that can be used to insure security of data in the file. Mask can be either a literal string of up to six characters or a string variable of up to six characters used to form a mask through which data is written to or read from the file. If the same mask is used to read a data item that was used to write the item, the results are the same value that was written.

When the ASSIGN statement is executed, the named file replaces the file previously referenced by the file number in the statement. Subsequent file references using this number will apply to the new file. Data written to the old file will be intact.

# SERIAL FILE PRINT

## EXAMPLES:

125 PRINT #5; A1,B2,C\$

130 PRINT #5; 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 16). If the value is n, then the nth file declared in the FILES statements (or the file most recently ASSIGNED to the nth position) 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 if the user types 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 #5; 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 pointer must be reset to the beginning of the file before the data that was just written 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.*

# THE TYP FUNCTION

**EXAMPLES:**

```
100 IF TYP(1)=2 THEN 1000  
250 IF TYP (6)=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 statements. In this case, TYP returns these values for the next data item: 1 = number; 2 = string; 3 for an "out of data" condition.

# 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$[72]  
300 IF END #1 THEN 1000  
500 IF TYP(1)=1 THEN 600 }  
550 IF TYP(1)=2 THEN 700 }  
600 READ #1;A  
650 PRINT A  
675 GOTO 500  
700 READ #1;A$  
750 PRINT A$  
775 GOTO 500  
1000 PRINT "FILE LIST COMPLETED"  
2000 END
```

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.

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

Reads a number from file #1 into variable A.

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

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

## 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 for the same file changes it, or until an ASSIGN statement containing the same file number is executed.

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.

# STRUCTURE OF SERIAL FILES

When a file is OPENed, you can think of it as looking like this:

INFO = EOF OPEN-INFO,5 PEOF

EOF is a mark that shows the end of the data.

**PEOF** is the physical end of the file, beyond which no data can be written.

↑ is the position of the file pointer.

↓

When information is written into the file, the pointer moves and space in the file is used up.

INFO = A B C F\$ Q1 EOF PEOF

六

100 FILES INFO

200 PRINT #1; A,B,C,F\$.01, END

The file is filled solidly from the beginning.

↓

When more information is PRINTED, it follows the previous data and the pointer is changed.

INFO = 

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

↑

300 PRINT #1; G1,G2,G\$,H\$,Z,END

## 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

↑

400 READ #1,1

↓

Now the data can be read.

INFO = A B C F\$ Q1 G1 G2 G\$ H\$ Z EOF

↑

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

↑

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 |
|---|---|---|----|----|----|----|-----|-----|---|-----|--|------|

↑

# EXAMPLE OF SERIAL FILE MODIFICATION

OPEN-DATUMS, 128

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

NAM-ADIT

```
100 FILES DATUMS
200 DIM A$[72]
300 IF END #1 THEN 1500
400 REM THIS PROGRAM FIRST FINDS THE END OF THE FILE. IT ASKS THE
410 REM USER FOR A STRING AND A NUMBER. IF THIS IS NOT THE PHYSICAL
420 REM END OF THE FILE, IT ADDS THEM TO THE END OF THE FILE.
430 REM THEN THE PROGRAM ASK THE USER IF HE WANTS TO ADD ANY MORE ITEMS.
440 REM IF THE USER ANSWERS YES, THE PROGRAM REPEATS THE INPUT AND
450 REM WRITE LOOP.
800 READ #1;A$,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.

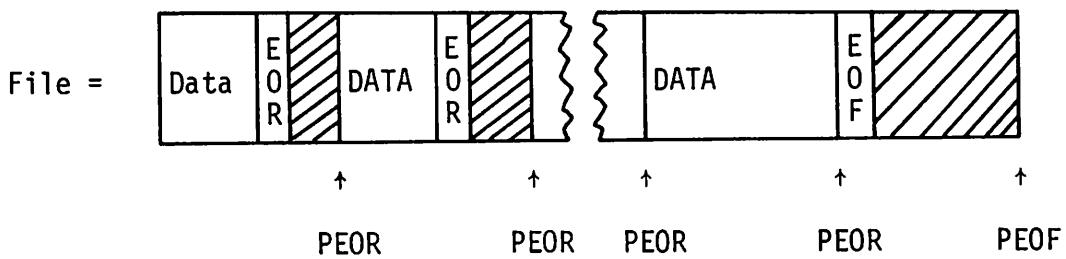
You can use the listing sample program to check the contents of the file.

# TERM: RECORD

DEFINED IN TSB AS: A physical division of a file; consisting of from 64 to 256 words.

The number of records in a file is subject to several constraints, but in no case may it exceed 32767.

## 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. If a full-size record is filled completely with numbers, it contains 128 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

$$1 + \frac{\text{number of characters in the string} + 1}{2}$$

If the number of characters is even, then

$$1 + \frac{\text{number of characters in the string}}{2}$$

Eight 62-character strings will completely fill a 256-word 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 256-word record could contain five strings of 72 characters (each using 37 words) and a maximum of 35 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

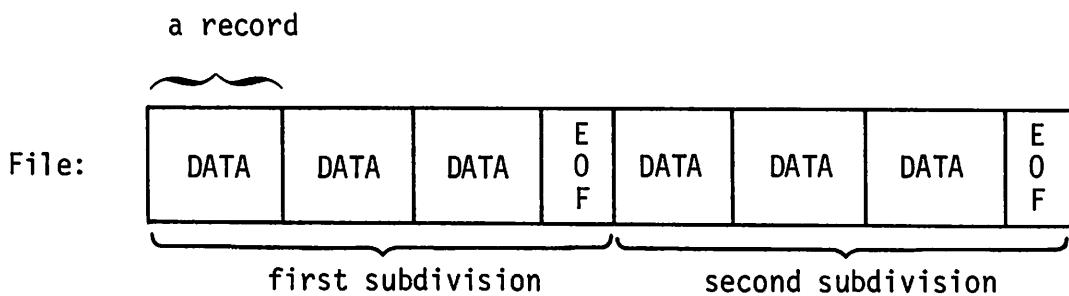
Here is a sample program that determines the number of records in a file. It uses the READ #M,N statement through the records until it comes to the physical end of the file.

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 32767  
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

```
100 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:      `100 GO TO TYP(-1) OF 200, 250, 300, 400`  
`2000 A=TYP(-5) + 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.

# 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$[72]
30   IF END #1 THEN 170
40   IF END #2 THEN 180
50   FOR I=1 TO 32767
60   READ #1,I
70   PRINT #2,I
80   GOTO TYP(-1) OF 90,120,150,160
90   READ #1;X
100  PRINT #2;X
110  GOTO 80
120  READ #1;A$
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 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, so a maximum of 128 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. If the READ requests 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:

```
320 PRINT #M+N, R+S  
330 PRINT #1,2  
340 PRINT #12,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
10 FILES X
20 IF END #1 THEN 60
30 FOR I=1 TO 32767
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 32767
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
45 IF END #1 THEN 270
50 READ #1,N+2
60 PRINT "STRING";
70 INPUT G$
72 IF N#0 THEN 80
74 R=2
76 GOTO 180
80 F=2
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+1 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
255 IF L-F#1 THEN 140
260 F=F+1
265 GOTO 100
270 PRINT "FILE FULL."
280 STOP
290 N=0
300 GOTO 45
310 END
```

## FILE ACCESSING ERRORS

If a data error occurs while the computer is performing a requested file read or write, the program will be terminated and one of the following messages will be printed:

BAD FILE READ IN LINE nn  
BAD FILE WRITE DETECTED IN LINE nn

As is the case with other errors which terminate a running program, the specific contents of any file written on during execution cannot be easily predicted.

Most of the information in the file on which the data error occurred may be recoverable. If file errors persist, the information should be copied item by item or record by record to another file.



## SECTION V MATRICES

A matrix is a doubly subscripted array, or a collection of data arranged in rows and columns. Arrays are described in Section III. This section describes a series of special instructions used to manipulate matrices. Instructions starting with MAT refer to an entire matrix, or to two or more matrices. Instructions such as PRINT and INPUT refer to specific elements of the array by row and column. The DIM statement is used to define the dimensions of the matrix and to reserve storage space for it. Some typical matrix operations are:

|                   |  |
|-------------------|--|
| MAT READ A,B,C    | Read the three matrices, their dimensions having been previously specified. Data is stored in the matrix row by row. |
| MAT INPUT A,B     | Input matrices A and B from the teleprinter--same restrictions as MAT READ.  |
| MAT C = ZER       | Fill C with zeros.   |
| MAT C = CON       | Fill C with ones.  |
| MAT C = IDN       | Set up C as an identity matrix.  |
| MAT PRINT A,B;C   | Print the three matrices, with A and C in the regular format, but B closely packed.                                  |
| MAT B = A         | Set the matrix B equal to the matrix A   |
| MAT C = A + B     | Add the two matrices A and B   |
| MAT C = A - B     | Subtract the matrix B from the matrix A.   |
| MAT C = A*B       | Multiply the matrix A by the matrix B.   |
| MAT C = TRN(A)    | Transpose the matrix A.  |
| MAT C = (K)*A     | Multiply the matrix A by K. K, which must be in parentheses, may be a formula.                                       |
| MAT C = INV(A)    | Invert the matrix A.   |
| MAT PRINT #5;A    | Print matrix A onto a file.  |
| MAT READ #M,N+2;D | Read matrix D from a file, row by row--same restrictions as MAT READ.  |

Use of these statements is described in this section. Formatted printing of matrices is described in Section VIII.

# 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 10 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. When the working size of a matrix is changed using one of the MAT statements described on the following pages, the values of excluded positions are lost.

The absolute maximum matrix size is about 4900 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.

# MAT...CON

## EXAMPLES:

205 MAT C = CON

210 MAT A = CON (N,N)

220 MAT Z = CON (5,20)

230 MAT Y = CON (50)

## GENERAL FORM:

statement number MAT matrix variable = CON

or

statement number MAT matrix variable = CON ( expression )

or

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

## PURPOSE

Sets up a matrix with all elements equal to 1; a new working size may be specified, within the limits of the original DIM statement on the total number of elements.

## COMMENTS

The new working size (an implicit DIM statement) may be omitted, as in example statement 205.

Note that since 1 has a logical value of "true", the MAT...CON statement is useful for 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.

See ENTER, Section III for an additional means of inputting specific matrix elements.

# 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.

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

"??" response to an input item means that more values are required.

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 individual matrix elements.  
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.

### COMMENTS

The MAT PRINT statement causes the matrix elements to be printed row by row across the page. Each matrix row starts a new teleprinter line. The spacing between row elements is controlled by the use of , and ; in the same manner as for the PRINT statement. Rows containing more elements than can be printed on a line are continued on consecutive lines. Each row of a matrix is started on a new line and is separated from the previous row by a blank line. Thus the instruction

```
MAT PRINT A, B;C
```

will cause the three matrices to be printed A and C with five components to a line and B with up to twelve.

Singly subscripted arrays may be interpreted as column vectors. Vectors may be used in place of matrices, as long as the above rules are observed. Since

## MAT PRINT CONTINUED

a vector like V(I) is treated as a column vector by BASIC, a row vector has to be introduced as a matrix that has only one row, namely row 1. Thus

```
DIM X(7), Y(1,5)
```

introduces a 7-component column vector and a 5-component row vector.

A column vector will be printed one element to the line with double spacing between lines. A row vector will be printed in the manner indicated by the form of the statement. For example: if V is a row vector then, "MAT PRINT V" or "MAT PRINT V," will print V as a row vector, five numbers to the line, while

```
MAT PRINT V;
```

will print V as a row vector with up to twelve numbers to the line.

# 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:

350 MAT READ A  
370 MAT READ B(5),C,D  
380 MAT READ Z (5,8)  
390 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:

310 MAT C = B + A

320 MAT X = X + Y

330 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 320.

# MATRIX SUBTRACTION

EXAMPLES:

550 MAT C = A - B

560 MAT B = B - Z

570 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 560.

# MATRIX MULTIPLICATION

EXAMPLES:

930 MAT Z = B \* C

940 MAT X = A \* A

950 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 TRANSPOSITION

EXAMPLES:

959 MAT Z = TRN (A)

969 MAT X = TRN (B)

979 MAT Z = TRN (C)

GENERAL FORM:

statement number MAT matrix variable = TRN (matrix variable)

## PURPOSE

Establishes a matrix as the transposition of a specified matrix; transposes rows and columns.

## COMMENTS

Sample transposition:

| <u>Original</u> | <u>Transposed</u> |
|-----------------|-------------------|
| 1 2 3           | 1 4 7             |
| 4 5 6           | 2 5 8             |
| 7 8 9           | 3 6 9             |

Note that the dimensions of the resulting matrix must be the reverse of the original matrix. For instance, if A has dimensions of 6,5 and MAT C = TRN (A), C must have dimensions of 5,6. The same matrix can not be on both sides of the "=" sign.

# 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.

In performing the inversion, the system must generate an additional internal matrix, so that an additional amount of storage equal to that needed for the original matrix is required. It may not be possible to invert an extremely large matrix.

## MAT PRINT #

### EXAMPLES:

520 MAT PRINT #5; A  
530 MAT PRINT #6, 3; B  
540 MAT PRINT #4,M; A  
550 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 128 numeric values because that is the maximum a record can hold. Attempting to exceed 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.

*NOTE: A matrix may also be printed with formatted output. See PRINT USING, Section VIII.*

## MAT READ #

EXAMPLES:

720 MAT READ #2;A  
730 MAT READ #2,3;B  
740 MAT READ #M,N;B(5)  
750 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.)...

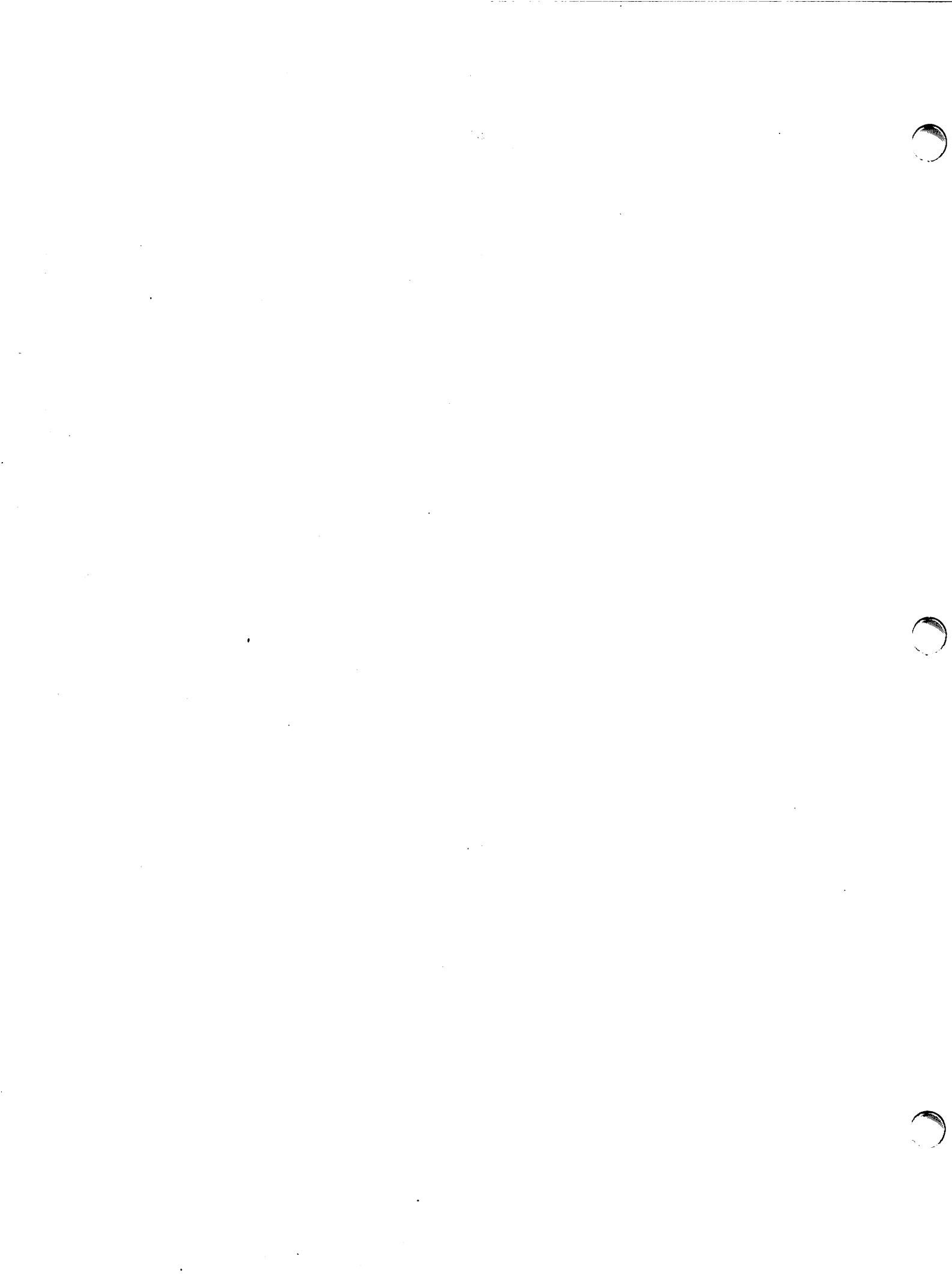
### PURPOSE

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 128 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.

Lower-case alphabetic characters can be input from or output to user terminals having this capability. When lower-case characters are output to a terminal not capable of printing them, most terminals will print such characters as the upper case equivalent. Lower-case characters are automatically converted to upper case by the system, except when they occur in strings or REM statements. Lower-case characters in strings used as file names in ASSIGN statements or program names in CHAIN statements are also converted to upper case when used.

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:

"ABCDEFGHIJKLMNP"  
"12345"  
"BOB AND TOM"  
"MARCH 13, 1970"

Special purpose characters such as esc (or alt-mode) and quotation marks cannot be used within a string. Quotation marks are used to delimit a string.

**NOTE:** Quotation marks are accepted in strings by the ENTER statement.

Apostrophes and control characters are legal as string characters.

Strings are manipulated in string variables.  
For example:

100 A\$ = "THIS IS A 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:

710 DIM A\$(72),B\$(20),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:

|                                  |                                   |
|----------------------------------|-----------------------------------|
| 100 DIM A\$(72)                  | (Sets physical length of A\$)     |
| 200 A\$ = "SAMPLE STRING"        | (Logical length of A\$ is 13)     |
| 300 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 from within a string variable. The substring is defined by a subscript 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\$ = "ABCDEFGHI"

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 , CONTINUED

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

400 PRINT Z\$(2,2)

prints the substring

B

# STRINGS AND SUBSTRINGS

A string can be made into a null string (no value, as distinguished from a blank space which has a value). This is 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 of A\$ is now a string in B\$.

Substrings can be used as 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

See STRING IF.

## 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. String variables having more than 1 character must be mentioned in a DIM statement.

Special purpose characters, such as `+` or (esc or alt-mode) 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).

*NOTE: The ENTER statement (Section III) can be used to input a character string. When using the ENTER statement for character strings, the string being entered should not be enclosed in quotation marks, but may contain quotation marks.*

# 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.

**NOTE:** The PRINT USING statement (Section VIII) can be used to provide greater control of format over strings and substrings.

# 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.

Mixed string and numeric values may be read (example statement 310 above); see "The TYP Function," Section IV 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 STRING EVALUATION BY ASCII CODES, Section IX.)

If substring subscripts extend beyond the length of the string, null characters (rather than blanks) are appended.

String compares may appear only in IF...THEN statements and not in conjunction with logical operators (Section VII).

## 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 IV, for description of a data type (string, numeric) check which may be used with DATA statements.

# PRINTING STRINGS ON FILES

## EXAMPLES:

```
350 PRINT #5; "THIS IS A STRING."  
355 PRINT #8; C$, B$, X$, Y$, D$  
360 PRINT #7,3; 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.

See "The TYP Function," Section IV for description of a data type check.

# READING STRINGS FROM FILES

EXAMPLES:

```
710 READ #1, 5; A$, B$  
715 READ #2; C$, A1, B2, X  
720 READ #3,6; C$(5),X$(4,7),Y$  
730 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 IV  
for description of a data type check.



## SECTION VII

### LOGICAL OPERATIONS

Logical evaluation simply determines whether a given statement or expression is true or false. When applied to numeric values, any non-zero expression is considered "true"; a value of zero is considered "false."

When an expression or statement is logically evaluated by the TSB system, it is assigned one of two numeric values -- a 1 if the expression or statement is logically "true," or a 0 if the expression or statement is logically "false."

Logical decisions are used to select one of two or more paths of execution through a program. Executing an IF statement, described in this section, causes the system to perform a specified statement next if the condition in the IF statement is true, and a different statement if the condition is false.

The truth or falsity of a statement or expression can also be determined and printed as a 1 for true or a 0 for 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<br>200 IF A9#27.65 THEN 700<br>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. If the condition is false, the next sequential statement after 150 is executed.

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.

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<br>615 PRINT X=27<br>620 PRINT X#27<br>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."

## RELATIONAL OPERATORS CONTINUED

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: | 510 IF A1 OR B THEN 670<br>520 IF B3 AND C9 THEN 680<br>530 IF NOT C9 THEN 690<br>540 IF X THEN 700 |
|-----------|---|

Statement 510 is interpreted: "if either A1 is true (has a non-zero value) or B is true (has a non-zero value) then transfer control to statement 670."

Similarly, statement 540 is interpreted: "if X is true (has a non-zero value) then transfer control to statement 700."

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 520 would evaluate to "true," since both B3 and C9 have a non-zero value.

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = 0.

|           |   |
|-----------|---|
| EXAMPLES: | 490 LET B = C = 7<br>500 PRINT B AND C<br>510 PRINT C OR B<br>520 PRINT NOT B |
|-----------|---|

Statements 500 - 520 returns a numeric value of either: 1, indicating that the statement has a logical value of "true," or 0, indicating a logical value of "false."

## BOOLEAN OPERATORS CONTINUED

Note that the criteria for determining the logical values are:

true = any non-zero expression value

false = an expression value of  $\emptyset$ .

The numeric value 1 or  $\emptyset$  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<br>310 PRINT (A>B) AND (X<Y)<br>320 LET C = NOT D<br>330 IF (C7 OR D4) AND (X2 OR Y3) THEN 930<br>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. N must have been given a value earlier in the program.

## SECTION VIII

# FORMATTED OUTPUT

The PRINT USING and IMAGE statements give the user more explicit and exact control over the format of his output. Numbers can be printed in three forms--integer, fixed-point, or floating point--with control of + and - signs. Strings may be printed in specified fields. Blanks can be inserted wherever needed. Carriage return and linefeed can be controlled, and lines longer than 72 characters can be printed. PRINT USING requires more programming effort than a simple PRINT, but it provides the ability to output data in whatever format the programmer needs.

# DEFINITIONS

| <u>Term</u>          | <u>Defined in TSB as:</u>  |
|----------------------|--|
| FORMATTED OUTPUT     | Similar to normal output (PRINT statement) except that, in addition to an expression list of output values, the PRINT USING statement also specifies a format string that determines the form in which the values are printed.   |
| EXPRESSION LIST      | A list of expressions and string variables separated by commas and optionally interspersed with space functions. An expression list must not contain literal strings.  |
| FORMAT STRING        | A string of up to 72 characters, consisting of an optional carriage control character followed by a list of format specifications separated by commas or slashes (/).  |
| FORMAT SPECIFICATION | A series of format characters and repetition factors that determines the format (field width, decimal point, sign, etc.) of one item in the expression list. Can also be a literal string in certain situations. Format specifications can be gathered into a repeatable group through the use of parentheses. |
| FORMAT CHARACTERS    | The characters A, X, D, S, ., and E are used to specify output fields for strings and numbers.   |

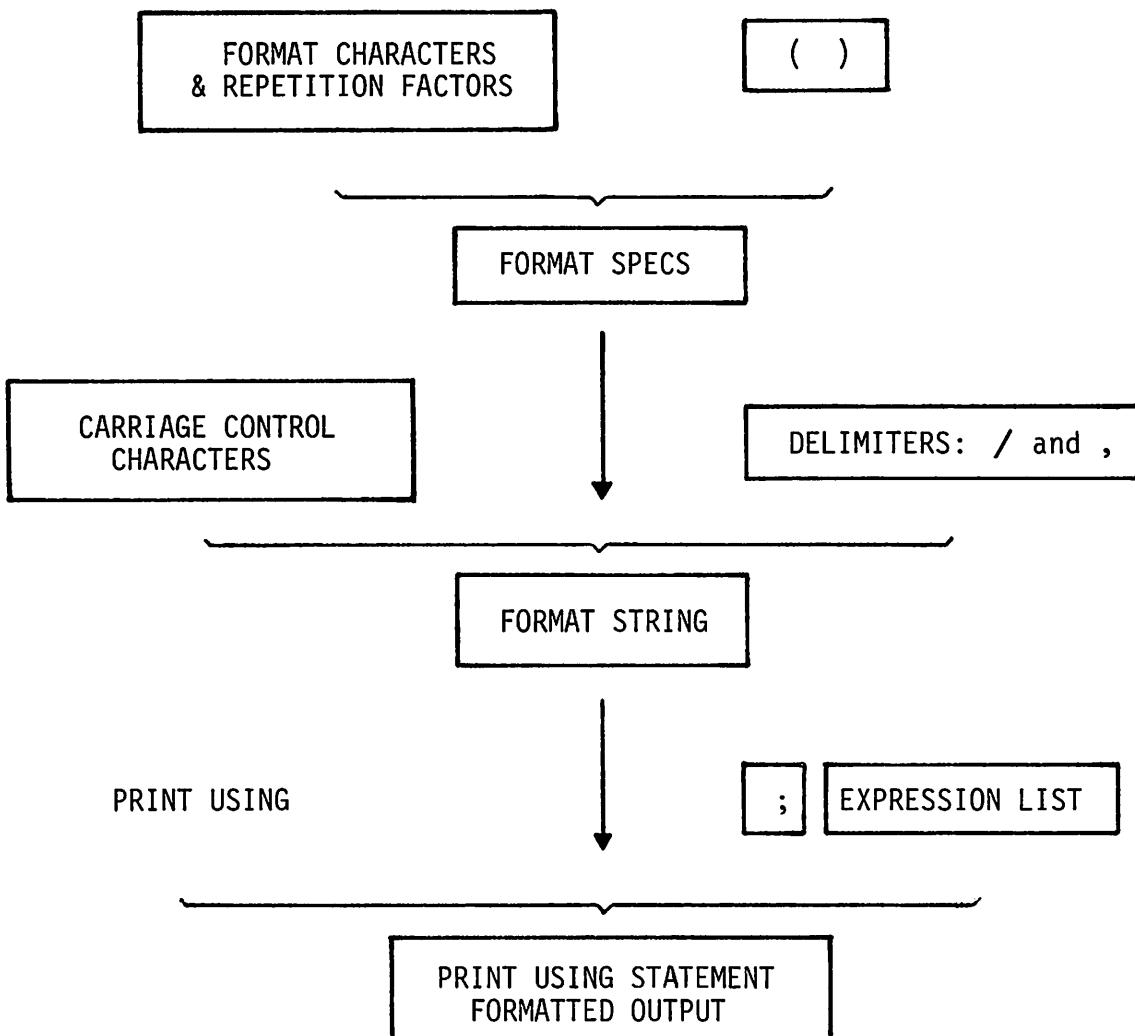
## DEFINITIONS, CONTINUED

| <u>Term</u>       | <u>Defined in TSB as:</u>   |
|-------------------|---|
| REPETITION FACTOR | An unsigned integer (e.g., 3, 6, 12, 32) that is placed before a format character or group of format specifications in order to repeat it (e.g., 3A = AAA; 2(3A,4A) = 3A,4A,3A,4A). The repetition factor must be between 1 and 72 inclusive.   |
| SLASH             | A delimiter (/) used to separate specifications when a carriage return-linefeed is desired before processing the next specification. Multiple slashes may be used (///).  |
| LITERAL STRING    | Any sequence of characters, other than quote marks ("), that is surrounded by quote marks and is to be printed as it appears.   |
| SPACE FUNCTIONS   | <p>Three functions can appear in an expression list:</p> <p>TAB(x) - Tabs out to column x before printing next item.</p> <p>LIN(x) - Skips <math> x </math> lines before printing next item. (If <math>x &lt; 0</math>, no carriage return is generated. If <math>x = 0</math>, only a carriage return is generated.)</p> <p>SPA(x) - Skips x spaces before printing next item.</p> |

## DEFINITIONS, CONTINUED

| <u>Term</u>                 | <u>Defined in TSB as:</u>  |
|-----------------------------|--|
| CARRIAGE CONTROL CHARACTERS | <p>At the beginning of any format string there may appear one of three optional characters set off by a comma:</p> <ul style="list-style-type: none"><li>+ means to suppress linefeed.</li><li>- means to suppress carriage return.</li><li># means to suppress carriage return and linefeed.</li></ul> <p>These characters specify action to be taken after the PRINT USING statement is complete. If no character is specified, the default condition is a carriage return and linefeed.</p> |

# SUMMARY



EXAMPLES: PRINT USING "DDD.DDD" ; Z1  
                  FORMAT STRING           EXPRESSION LIST

PRINT USING "2X,3(3D.3D,2X)" ; Z1,Z2,Z3  
                  FORMAT STRING           EXPRESSION LIST

# STRING FORMAT SPECIFICATIONS

## FORMAT CHARACTERS USED:

A - calls for one ASCII character to be output from a string in the expression list.

X - specifies that a blank be printed next.

nA - calls for n ASCII characters (n = repetition factor).

nX - specifies that n blanks be printed.

## COMBINATION RULES:

Any combination of X's, A's, and repetition factors specifies a legal STRING FORMAT SPECIFICATION. When such a specification is encountered in a format, the next item in the expression list must be a string.

## FORMAT EXAMPLES

|         |   |  |
|---------|---|--|
| AAAA    | } | equivalent   |
| 4A      |   |  |
| 2A2A    |   |  |
| 4X      |   | special case (all blanks, so no variable required) |
| AXAXAXA |   | alternate characters and blanks                    |
| 2X20A   |   |  |

## OUTPUT EXAMPLES

| Format Spec | Contents of String Variable | Format of Output |
|-------------|-----------------------------|------------------|
| 6A          | ABCDEF                      | ABCDEF           |
| 5A          | ABCDEF                      | ABCDE            |
| 8A          | ABCDEF                      | ABCDEF..         |
| 2X6A        | ABCDEF                      | ..ABCDEF         |
| AXAXAXAXAXA | ABCDEF                      | A.B.C.D.E.F      |

## STRING FORMAT SPECIFICATIONS, CONTINUED

### COMMENTS

The string is left-justified in the field and any leftover spaces are filled with blanks. If the string variable contains more characters than the specification allows, characters on the right are truncated.

# INTEGER FORMAT SPECIFICATIONS

## FORMAT CHARACTERS USED:

- D - calls for one decimal digit to be printed from a number in the expression list.
- nD - calls for n contiguous decimal digits to be printed from a number in the expression list.
- X - specifies that a blank is to be printed within the field for the number (nX is also allowed).
- S - specifies that the sign (+ or -) of the number is to be printed.

## COMBINATION RULES:

Any combination of X and D is allowed, but at least one D must be present and only one S is allowed. When such a specification occurs in a format, the next item in the expression list must be a number. This number is rounded to an integer and printed right-justified. Although the requested number of digits will be printed, only six can be guaranteed to be significant.

## FORMAT EXAMPLES

|       |   |            |
|-------|---|------------|
| DDDD  | } | equivalent |
| 4D    |   |            |
| 2DDD  |   |            |
| 2D2D  |   |            |
| 2DX3D |   |            |
| SDDD  |   |            |
| S4D   |   |            |
| DX3DS |   |            |

# INTEGER FORMAT SPECIFICATIONS, CONTINUED

| <u>Format Spec</u> | <u>OUTPUT EXAMPLES</u> |                         |
|--------------------|------------------------|-------------------------|
|                    | <u>Value</u>           | <u>Format of Output</u> |
| 4D                 | 1234                   | 1234                    |
| S4D                | 1234                   | +1234                   |
| 4DS                | 1234                   | 1234+                   |
| 5D                 | 1234                   | .1234                   |
| 4D                 | 1234.8                 | 1235                    |
| DXDDD              | 1234                   | 1.234                   |
| S10D               | 1234                   | ~~~~~+1234              |
| DSDDD              | 1234                   | 1+234                   |
| 5D                 | -1234                  | -1234                   |
| 4D                 | 1234.2                 | 1234                    |

## COMMENTS

If there is not enough room in the field for the number (i.e., the number of digits is greater than the number of D's in the format spec), then the value is printed on a separate line in a floating-point format (SD.5DE) so that the programmer can analyze what went wrong.

If an S precedes all D's, the sign is printed immediately preceding the first digit of the number. If an S appears past the first D, the sign is printed at the location of the S.

If an S is not included in the format, then an extra D should be provided if the value could possibly be negative. When the value is negative, the - sign is always printed preceding the most significant digit and a space must be provided for it with a D or the field may overflow.

The ability to insert blanks can be combined with the ability to overprint (carriage control) in order to produce useful results. For example, large numbers can be printed with blanks left in the correct spots for commas to be inserted after each group of three digits (e.g., \$10,937).

# FIXED-POINT FORMAT SPECIFICATIONS

## FORMAT CHARACTERS USED:

Same as INTEGER FORMAT, plus

- . - specifies the location of the decimal point.

## COMBINATION RULES:

Any combination of D and X to the left and right of the decimal point is allowed, but at least one D must be present and only one S and one "." are allowed. For this specification, the next item in the expression list must be a number. The digits to the right of the decimal point are rounded to fit in the field. Leading zeros to the left are suppressed, but trailing zeros are always printed.

## FORMAT EXAMPLES

DDD.DDD  
DDD.3D  
3D.3D  
3D.DDD  
S3D.3D  
DXDXDX.DDXD  
XD6X4D.8D  
DDSDD.3D

} equivalent

## OUTPUT EXAMPLES

| <u>Format Spec</u> | <u>Value</u> | <u>Format of Output</u> |
|--------------------|--------------|-------------------------|
| 3D.4D              | 465.465      | 465.4650                |
| 4D.2D              | 465.465      | .465.47                 |
| 4D.3D              | -465.465     | -465.465                |
| SDD2D.D            | 465.465      | .+465.5                 |
| S2D.4D             | .465         | .+0.4650                |
| S.4D               | .465         | +.4650                  |
| D.4D               | -.465        | -.4650                  |
| 2D.4D              | -.465        | -0.4650                 |

## FIXED-POINT FORMAT SPECIFICATIONS, CONTINUED

### COMMENTS

If the number to be printed has no digits to the left of the decimal point but D's are provided to the left, then a zero ("0") will be printed in the rightmost D on the left side. If an S is provided to the left, it is moved to the right through D's and X's until it comes to the first non-blank character. If an S is not provided and the number is negative, then one of three things will happen: 1) no D's to the left causes overflow; 2) one D to the left will be used for the "-" sign and the "0" will not be printed; or 3) two or more D's to the left, then the "-" and "0" will be printed in the positions reserved by the rightmost two D's.

# FLOATING-POINT FORMAT SPECIFICATIONS

## FORMAT CHARACTERS USED:

Same as FIXED-POINT FORMAT, plus  
E - signifies floating point format.  
X - as defined earlier may follow E.

## COMBINATION RULES:

Any legal INTEGER or FIXED-POINT format specification followed by an E is a legal FLOATING-POINT format. The E stands for "exponent" and signifies a four-character field consisting of an "E" followed by "+" or "-" and two decimal digits. When 10 is raised to the power printed after E and multiplied by the number in the integer or fixed-point field, the result is the value being output. This format is useful for numbers that are very large or very small. For example, .00005 =  $.5 \times 10^{-4}$  = .5E-4. X's may follow the E and they cause Blanks to be printed between the E and the exponent sign.

## FORMAT EXAMPLES

|            |          |
|------------|----------|
| SD.5DE     | S6D.E    |
| DDD.DDDXEX | S6D.XE   |
| SD.8DXE    | S6D.DDDE |
| S6DE       |          |

## OUTPUT EXAMPLES

| <u>Format Spec</u> | <u>Value</u>             | <u>Format of Output</u> |
|--------------------|--------------------------|-------------------------|
| SDXE               | $4.82716 \times 10^{21}$ | +5~E+21                 |
| DDDD.DDE           | same                     | 4827.16E+18             |
| S5DX.X5DEX         | same                     | ~~+48~.27159E~+20       |
| SD.5DE             | same                     | +4.82716E+21            |
| S.10DE3X           | same                     | +.4827159382E~~+22      |

# FLOATING-POINT FORMAT SPECIFICATIONS, CONTINUED

## COMMENTS

Note again that the format can specify an unlimited number of digits in a specification, but only six of these are guaranteed accurate. When more than six digits are requested, non-significant digits are printed as in the preceding examples.

To produce the output, the output value from the expression list is multiplied or divided by 10, the number of times necessary to fit the value into the field. It is then rounded from the right, and the exponent is adjusted to account for the multiplications or divisions.

If the format allows for more digits than there are significant digits in the output value, two rules are followed:

- 1) If there are more than 6 D's on the right side of the decimal point, the leftmost digit is printed in the first D (if any) to the left of the decimal point or the first D to the right of the decimal point; extra D's beyond 7 on the right are filled with non-significant digits. In the following examples, the arrow indicates the position of the leftmost digit printed:

|        |          |          |
|--------|----------|----------|
| DD.40D | XX.DD40D | 40DD.40D |
| ↑      | ↑        | ↑        |

- 2) If there are less than 7 D's on the right side of the decimal point, the leftmost digit is printed in the seventh D position from the right (or the leftmost if there are not 7). In the following examples, the arrow indicates the position of the leftmost digit printed:

|            |       |      |
|------------|-------|------|
| 6DDDD.DDDD | DD.DD | D.6D |
| ↑          | ↑     | ↑    |

# POSITION OF THE SIGN

## 1. When S is used.

If S precedes any D, the sign position is moved to the right through X's and D's and is printed immediately to the left of the first non-blank character. If the number to be printed is a fraction with no digits to the left of the decimal point and any D's appear on the left of the decimal point, then a "0" appears in the rightmost D and the sign floats up to that "0".

If S is preceded by one or more D's, the sign is printed at the position of the S and does not float.

## 2. When S is not used.

When the number is negative, an extra D must be present to reserve a place for the sign. The position of the sign is moved through unused D's and X's to the first non-blank character. If not enough D's are provided for all the significant digits and sign of a negative number, then the field overflows and the number is printed on a separate line in SD.5DE format.

## GROUPED FORMAT SPECIFICATIONS

One or more format specifications can be gathered within parentheses to make a group. This group must be repeated by prefacing it with a repetition factor between 1 and 72 inclusive. Within the parentheses, the specifications must be separated by commas or slashes and the group must be set off from other specifications by a comma or slashes, just as if it were a single specification. Groups can be nested two levels deep.

### EXAMPLES

4(10A,2X,4D,2X//)  
3(10D,2(3DX,4DX),4A)  
3D.3D//3(20A,6D,4(2A2X)//)

## FORMAT STRINGS

DEFINED IN TSB AS: A collection of format specifications (or groups of format specifications) set off by commas or slashes and optionally preceded by a carriage control character set off by a comma. One format string is used by one PRINT USING statement. The first character of a format string must not be a slash (/) or a comma.

### EXAMPLES

+,20A,2X,S4D.2D  
6D,2X,6DSX,13AXAX,2(4D,2X,3AX)  
-,20A/20X20A/40A20X/

## TERM: EXPRESSION LIST

DEFINED IN TSB AS: The list of items to be printed using the format string. The items must be separated by commas (not semicolons), and the list must not contain any literal strings. The types of the items (numerical or string) must match the types of items called for in the format string. Space functions (SPA,LIN,TAB) may appear in the list.

# PRINT USING

EXAMPLES:

```
300 PRINT USING 200;A,B4,C$,TAB(50),67.78  
400 PRINT USING A$;A,A3,C$,D$  
500 PRINT USING "6DX,25A";A,A$
```

GENERAL FORM:

statement number PRINT USING format string ; expression list

## PURPOSE

To print out data according to a specified format.

## COMMENTS

The format string can be specified in one of three ways:

- a. an actual string ("6D,X20A")
- b. a string variable containing the format string (A\$,B\$(5,20))
- c. the statement number of an IMAGE statement containing the format string (200).

The expression list is a list of expressions separated by commas; the semicolon and expression list are optional.

When the PRINT USING statement is executed, the format string is examined and the carriage control character, if any, is saved. Each specification is extracted and examined. If it calls for a string or numerical item, the next expression in the expression list is taken and printed according to the specification.

If there are no more specifications or the specification is of the wrong type, execution of PRINT USING terminates.

If the specification does not require an item from the list (e.g., a blank or literal specification), the specification is printed without examining the expression list.

## PRINT USING, CONTINUED

If the end of the format string is reached before the end of the expression list, processing continues from the beginning of the format string.

When all expressions have been printed, a carriage return and linefeed (modified by the carriage control character) are printed.

### EXAMPLES

In these examples, the variables have these values: A = +12345, B = -1234, C = 123, D = 12, E = -12345, F = 123456, G = -1, H = 1234.

100 PRINT USING "3(S6D2X)/ ";A,B,C,D,E,F

#### Output

```
^+12345~~~~~-1234~~~~~+123  
~~~~~+12~~~-12345~~+123456
```

100 PRINT USING "3(S6D2X)/";A,G

#### Output

```
^+12345~~~~~-1
```

50 IMAGE "MONEY ",6DX,"COST ",6DX,"INPUT ",6DX  
100 PRINT USING 50;H,D

#### Output

```
MONEY~~~1234~COST~~~~~12
```

## MAT PRINT USING

EXAMPLES:

200 MAT PRINT USING 300;A,B,SPA(M),C

350 MAT PRINT USING A\$; B,N,M

400 MAT PRINT USING "SD.5DE2X";K

GENERAL FORM:

statement number MAT PRINT USING format string ; matrix list

### PURPOSE

To print out data from matrices in a specified format.

### COMMENTS

The format string is the same as in PRINT USING except that it must not contain any string specifications.

The matrix list is a list of matrices separated by commas. (The semicolon and matrix list are optional.) Space functions are allowed in the matrix list.

As in MAT PRINT, the matrices are printed in row by row order.

### EXAMPLE

```
10 DIM A(5,5)
:
:
100 PRINT USING "6(SD.5DE)/";A
```

## FORMAT IN A STRING VARIABLE

One way to specify the format string in a PRINT USING or MAT PRINT USING statement is by using a string variable that contains the format string. This allows the programmer to change the format during the execution of the program. The following excerpt from a sample program shows what can be done:

```
100 LET A$ = "DD,~~~~~DD"  
110 IF X<Y THEN 130  
120 A$(4,8) = "SD.E,"  
130 PRINT USING A$; ....
```

If X is not less than Y, then the format string becomes

DD,SD.E,DD

instead of

DD,DD

# IMAGE

EXAMPLES:

100 IMAGE 6D,"LITERAL STRING",SD.5DE

200 IMAGE XDDXDD.DDE,20A,3D

GENERAL FORM:

statement number IMAGE format string

## PURPOSE

To specify a format to be used in a PRINT USING statement.

## COMMENTS

An IMAGE statement is the one means by which a literal string can be introduced into a format string. Literal strings are printed exactly as they appear in the format string, similar to the way blanks are printed in a blank specification.

The format string is any legal format string; it is not enclosed in quotes and can therefore contain literal strings as format specifications.

# USING CARRIAGE CONTROL

This example demonstrates the use of the LIN function (statement 5), the carriage control characters (statements 20, 40, and 60), and literal strings in IMAGE statements.

## PROGRAM

```
5 PRINT LIN(5)
10 PRINT USING 20
20 IMAGE #,"# "
30 PRINT USING 40
40 IMAGE -,"SUPPRESSES LINEFEED AND CARRIAGE RETURN"
50 PRINT USING 60
60 IMAGE+,"- SUPPRESSES CARRIAGE RETURN"
70 PRINT USING 80
80 IMAGE "AND + SUPPRESSES LINEFEED."
90 END
```

## OUTPUT

```
# SUPPRESSES LINEFEED AND CARRIAGE RETURN
AND + SUPPRESSES LINEFEED.          - SUPPRESSES CARRIAGE RETURN
```

# NUMERICAL OUTPUT

This example program prints out the values of  $2^N$  and  $(-2)^N$ , where N varies from -5 to 20. Floating-point and integer formats are used (statement 350).

## PROGRAM

```
200 PRINT USING 210
210 IMAGE" N ",3X,"2 TO THE N",3X,"-2 TO THE N"
300 FOR N=-5 TO 20
350 PRINT USING "3D,2X,SD.5DE,2X,SD.5DE";N,2↑N,(-2)↑N
360 NEXT N
1000 END
```

## OUTPUT

| N  | 2 TO THE N   | -2 TO THE N  |
|----|--------------|--------------|
| -5 | +3.12500E-02 | -3.12500E-02 |
| -4 | +6.25000E-02 | +6.25000E-02 |
| -3 | +1.25000E-01 | -1.25000E-01 |
| -2 | +2.50000E-01 | +2.50000E-01 |
| -1 | +5.00000E-01 | -5.00000E-01 |
| 0  | +1.00000E+00 | +1.00000E+00 |
| 1  | +2.00000E+00 | -2.00000E+00 |
| 2  | +4.00000E+00 | +4.00000E+00 |
| 3  | +8.00000E+00 | -8.00000E+00 |
| 4  | +1.60000E+01 | +1.60000E+01 |
| 5  | +3.20000E+01 | -3.20000E+01 |
| 6  | +6.40000E+01 | +6.40000E+01 |
| 7  | +1.28000E+02 | -1.28000E+02 |
| 8  | +2.56000E+02 | +2.56000E+02 |
| 9  | +5.12000E+02 | -5.12000E+02 |
| 10 | +1.02400E+03 | +1.02400E+03 |

## NUMERICAL OUTPUT, CONTINUED

|    |              |              |
|----|--------------|--------------|
| 11 | +2.04800E+03 | -2.04800E+03 |
| 12 | +4.09600E+03 | +4.09600E+03 |
| 13 | +8.19200E+03 | -8.19200E+03 |
| 14 | +1.63840E+04 | +1.63840E+04 |
| 15 | +3.27680E+04 | -3.27680E+04 |
| 16 | +6.55360E+04 | +6.55360E+04 |
| 17 | +1.31072E+05 | -1.31072E+05 |
| 18 | +2.62144E+05 | +2.62144E+05 |
| 19 | +5.24288E+05 | -5.24288E+05 |
| 20 | +1.04858E+06 | +1.04858E+06 |

# REPORT GENERATION

This program is a sample report generator. It first requests a school number from the terminal, then reads and prints out information about the school's teachers from a file. Note that a carriage control character is used to advantage (statement 100), slashes (/) are used (statement 200), string and fixed-point fields are used (statement 210), and an error occurs in the output for the eighth teacher (number too large for field; therefore, it is printed in E format on a separate line).

## PROGRAM

```
10 REM THIS PROGRAM GENERATES A REPORT ON TEACHERS.  
50 DIM A$[25],B$[19],C$[19]  
60 FILES SCH1,SCH2,SCH3,SCH4,SCH5  
100 IMAGE#, "ENTER SCHOOL NUMBER:"  
150 IMAGE"TEACHER",13X,"SUBJECT",13X,"SALARY",4X,"ATTND."  
175 IMAGE "-----",13X,"-----",13X"-----",4x,"-----"/  
200 IMAGE"CENTRAL CITY SCHOOL DISTRICT"/"DAILY REPORT OF ",25A//  
210 IMAGE 20A,20A,"$",DDD.DD,DD.DDDD  
230 PRINT USING 100  
250 INPUT Z  
260 READ #Z;A$,N  
270 PRINT LIN(6)  
500 PRINT USING 200;A$  
550 PRINT USING 150  
555 PRINT USING 175  
557 FOR A1=1 TO N  
560 READ #1;B$,C$,A,B  
600 PRINT USING 210;B$,C$,A,TAB(50),B  
620 NEXT A1  
1000 END
```

# REPORT GENERATION, CONTINUED

## OUTPUT

ENTER SCHOOL NUMBER: ?1

### CENTRAL CITY SCHOOL DISTRICT DAILY REPORT OF B. BAKER HIGH SCHOOL

| <u>TEACHER</u>  | <u>SUBJECT</u>   | <u>SALARY</u> | <u>ATTND.</u> |
|-----------------|------------------|---------------|---------------|
| MISS BROOKS     | ENGLISH          | \$450.34      | 12.5000       |
| MISS CRABTREE   | REM. READING     | \$400.00      | 64.3200       |
| MISS GRUNDIE    | HISTORY          | \$350.00      | 1.0010        |
| MRS. HUMPREY    | SPELLING         | \$700.00      | 99.9900       |
| COLONEL MUSTARD | CRIMINOLOGY      | \$700.00      | 21.4500       |
| MISS PEACH      | LIFE PREPARATION | \$232.00      | 23.2320       |
| PROF. PLUM      | AGRICULTURE      | \$777.77      | 65.0050       |
| MISS H. PRYNNE  | SOCIAL STUDIES   | \$100.25      |               |
| +5.00500E+02    |                  |               |               |
| MISS SCARLETT   | P.E.             | \$205.10      | 25.0000       |
| MR. SIR         | HOME ROOM        | \$890.00      | 99.9000       |
| MR. T. TIM      | MUSIC            | \$ 10.99      | 0.0500        |
| MR. WEATHERBY   | ECONOMICS        | \$767.99      | 10.0400       |

## FATAL ERRORS

These errors cause termination of execution of the PRINT USING statement. An appropriate message is printed, along with the format specification that caused the error.

1. The replicator is outside the range  $1 \leq n \leq 72$ .
2. Appearance of a D,S,E or . in a string specification.
3. Appearance of an A in an integer specification, a fixed specification, or a floating specification.
4. Appearance of any character other than A,X,D,S,E,/ or . in any specification but literal.
5. A comma followed by a slash.
6. More than two levels of parentheses.
7. No D in a fixed or floating specification.
8. An S in a blank specification.
9. String expression attempted to output in non-string format.
10. A slash followed by a comma.
11. Two or more E's or . in a specification.
12. Literal string not separated by delimiters.
13. Missing quotes in a literal.
14. Specifications enclosed in parentheses without a replicator.
15. Specified statement is not IMAGE.
16. Attempt to print number with string format.

## NON-FATAL ERRORS

These errors do not cause termination of the PRINT USING statement. The action taken is indicated.

1. String specification field too narrow -- truncate string on right.
2. Field too narrow for integer or integer part of fixed specification -- number is printed in SD.5DE format on next line and printing resumes on following line.
3. Field too narrow for fraction part of fixed or floating specification -- round from right to fit into field.
4. Specification requires the printing of more than 46 digits -- 46 digits will be printed preceded by blanks filling the rest of the field.
5. More than one S -- subsequent S's are ignored.

## **SECTION IX**

### **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.
7. A character string that is not a literal can contain the character ", through the use of the ENTER statement.
8. A replicator must lie between 1 and 72, inclusive.

*NOTE: Parentheses, () , and square brackets [] , are accepted interchangeably by the syntax analyzer.*

# SYNTAX REQUIREMENTS OF TSB

|                          |  |
|--------------------------|--|
| <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 (see rule 1)  |
| <literal string>         | ::= "<character string>"   |
| <character string>       | ::= <character>   <character string><character><br><i>(See Rule 7.)</i>  |
| <character>              | ::= Any ASCII character except null, line feed,<br>return, x-off, alt-mode, escape, <code>&lt;</code> , <code>"</code> , and<br>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 <br>W X Y Z  |
| <subscripted variable>   | ::= <letter>(<sublist>)  |
| <sublist>                | ::= <expression>   <expression>, <expression>  |
| <string variable>        | ::= <string simple variable>   <string simple variable><br>(<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>(rule2) (<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> <ENTER statement> <PRINT statement> <PRINT USING statement> <RESTORE statement> <DIM statement> <COM statement> <DEF statement> <FILES statement> <REM statement> <CHAIN statement> <MAT statement> <ASSIGN statement> |
| <LET statement>        | ::= LET <leftpart><expression> <br>LET <destination string>=<source string> <br><leftpart><expression> <br><destination string>=<source string>   |

## SYNTAX REQUIREMENTS OF TSB , CONTINUED

|                       |  |
|-----------------------|--|
| <leftpart>            | ::= <variable>= <leftpart><variable>=  |
| <IF statement>        | ::= IF<decision expression>THEN<sequence number> <br>IF END #<file formula>THEN<sequence number>                             |
| <decision expression> | ::= <expression> <br><comparison string 1><relational operator><br><comparison string 2>                                     |
| <comparison string 1> | ::= <string variable>  |
| <comparison string 2> | ::= <string variable> <literal string>   |
| <GOTO statement>      | ::= GOTO<sequence number><br>GOTO<expression>OF<sequence list>   |
| <sequence list>       | ::= <sequence number> <sequence list>,<sequence number>  |
| <GOSUB statement>     | ::= GOSUB<sequence number> <br>GOSUB<expression>OF<sequence list>  |
| <RETURN statement>    | ::= RETURN   |
| <FOR statement>       | ::= FOR<for variable>=<initial value>TO<final value> <br>FOR<for variable>=<initial value>TO<final value><br>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> <br>READ<file reference>;<variable list>  |
| <variable list>       | ::= <read variable> <variable list>,<read variable>  |
| <read variable>       | ::= <variable> <destination string>  |
| <INPUT statement>     | ::= INPUT<variable list>   |

# SYNTAX REQUIREMENTS OF TSB CONTINUED

|                       |   |
|-----------------------|---|
| <ENTER statement>     | ::= ENTER #<variable>  <br>ENTER<expression>,<variable>,<variable>  <br>ENTER<expression>,<variable>,<string variable>  <br>ENTER #<variable>,<expression>,<variable>,<br><variable>  <br>ENTER #<variable>,<expression>,<variable>,<br><string variable> |
| <PRINT statement>     | ::= <type statement>   <file write statement>  <br>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>   <special function>  <br><source string>  |
| <A part>              | ::= A   <A part>A   <replicator><A part>  |
| <D part>              | ::= D   <D part>D   <replicator><D part>  |
| <X part>              | ::= X   <X part>X   <replicator><X part>  |
| <replicator>          | ::= <integer>   |
| <empty>               | ::=   |
| <string spec. comp.>  | ::= <A part>   <X part>   |
| <string spec. 1>      | ::= <string spec. comp.>  <br><string spec. comp.><string spec. 1>  |
| <string spec. 2>      | ::= <string spec. 1>  <br><empty>   |
| <string spec.>        | ::= <string spec. 2><A part><string spec 2>   |
| <integer spec. comp.> | ::= <D part>   <X part>   S   <empty>   |
| <integer spec.>       | ::= <D part>   <integer spec. comp><br><integer spec.><integer spec. comp.>   |
| <fraction spec.>      | ::= <integer spec. comp.>  <br><fraction spec.><integer spec. comp.>  |
| <fixed spec.>         | ::= <integer spec.><fraction spec.>  <br><fraction spec.>. <integer spec.>  |

# SYNTAX REQUIREMENTS OF TSB, CONTINUED

|                           |  |
|---------------------------|--|
| <floating spec>           | ::= <fixed spec.>E <integer spec.>E <br><floating spec.><X part>   |
| <format list element>     | ::= <string spec.> <fixed spec.> <br><floating spec.> <integer spec.> <br><X part> <literal string>  |
| <format list>             | ::= <format list element> <br><format list element>,<format list> <br><replicator>(<format list>) <br><format list>/<format list element> <br><format list>/   |
| <carriage control>        | ::= + - #  |
| <format string>           | ::= <format list> <carriage control>,<br><format list>   |
| <special function type>   | ::= TAB LIN SPA  |
| <special function>        | ::= <special function type>(<expression>)  |
| <expression list>         | ::= <expression list element> <br><expression list>,<expression list element>  |
| <expression list element> | ::= =<expression> <special function>   |
| <PRINT USING 1>           | ::= PRINT USING"<format string>" <br>PRINT USING<sequence number> <br>PRINT USING<string variable>   |
| <PRINT USING statement>   | ::= <PRINT USING 1>;<expression list> <br><PRINT USING 1>  |
| <IMAGE statement>         | ::= IMAGE<format string>   |
| <file write statement>    | ::= PRINT<file reference>;<write expression> <br><file write statement>,<write expression> <br><file write statement>;<write expression> <br><file write statement><literal string> <br><file write statement><literal string><br><write expression> |
| <write expression>        | ::= <expression> END <source string>   |
| <RESTORE statement>       | ::= RESTORE RESTORE<sequence number>   |
| <DIM statement>           | ::= DIM<dimspec> <DIM statement>,<dimspec>   |

# SYNTAX REQUIREMENTS OF TSB CONTINUED

<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 \$<name> |  
                  FILES\* name> | FILES\* |  
                  <FILES statement>,<name> |  
                  <FILES statement>,\$<name> |  
                  <FILES statement>,\*<name> |  
                  <FILES statement>,\*

<name> ::= a string of up to 6 printing <character>'s  
              except comma, and not beginning with "\$" or  
              "\*".

<REM statement> ::= REM<character string> (see rule 5)

<CHAIN statement> ::= CHAIN<source string> |  
                  CHAIN<source string>,<expression>

<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>

## **SYNTAX REQUIREMENTS OF TSB , CONTINUED**

|  |  |
|--|--|
| <MAT PRINT 2>                          | ::= <MAT PRINT 1>,   <MAT PRINT 1>;  |
| <MAT PRINT USING statement>            | ::= <MAT PRINT USING 1>  <br><MAT PRINT USING 1>;<array identifier list>   |
| <MAT PRINT USING 1>                    | ::= <MAT PRINT USING >"format string">  <br><MAT PRINT USING><sequence number>  <br><MAT PRINT USING><string variable>   |
| <array identifier list>                | ::= <array identifier list element>  <br><array identifier list>,  <br><array identifier list element>   |
| <array identifier list element>        | ::= <array identifier>   <special function>  |
| <MAT initialization statement>         | ::= MAT<array identifier>=<initialization function>  <br>MAT<array identifier>=<initialization function> (<dimensions>)  |
| <initialization function>              | ::= ZER   CON   IDN  |
| <MAT assignment statement><br>(rule 6) | ::= MAT<array identifier>=<array identifier>  <br>MAT<array identifier>=<array identifier><br><mat operator><array identifier><br>MAT<array identifier>=INV(<array identifier>)  <br>MAT<array identifier>=TRN(<array identifier>)  <br>MAT<array identifier>=(<expression>)*<array<br>identifier> |
| <mat operator>                         | ::= +   -   *  |
| <ASSIGN statement>                     | ::= ASSIGN<source string>,<expression>,<variable>  <br>ASSIGN<source string>,<expression>,<variable>,<source string>   |

# STRING EVALUATION BY ASCII CODES

Each teleprinter character is represented by an A.S.C.I.I.  
(American Standard Code for Information Interchange) number.

Strings are compared by their A.S.C.I.I. representation.

The A.S.C.I.I sequence, from lowest to highest is:

*Lowest:      bell*

|       |   |   |   |   |                |
|-------|---|---|---|---|----------------|
| space | 5 | J | a | v |                |
| !     | 6 | K | b | w |                |
| "     | 7 | L | c | x |                |
| #     | 8 | M | d | y |                |
| \$    | 9 | N | e | z | <i>Highest</i> |
| %     | : | O | f |   |                |
| &     | ; | P | g |   |                |
| '     | < | Q | h |   |                |
| (     | = | R | i |   |                |
| )     | > | S | j |   |                |
| *     | ? | T | k |   |                |
| +     | @ | U | l |   |                |
| ,     | A | V | m |   |                |
| -     | B | W | n |   |                |
| .     | C | X | o |   |                |
| /     | D | Y | p |   |                |
| Ø     | E | Z | q |   |                |
| 1     | F | [ | r |   |                |
| 2     | G | \ | s |   |                |
| 3     | H | ] | t |   |                |
| 4     | I | ↑ | u |   |                |

# MEMORY ALLOCATION BY A USER

Approximate space available for user allocation: 10,000  
2-character words.

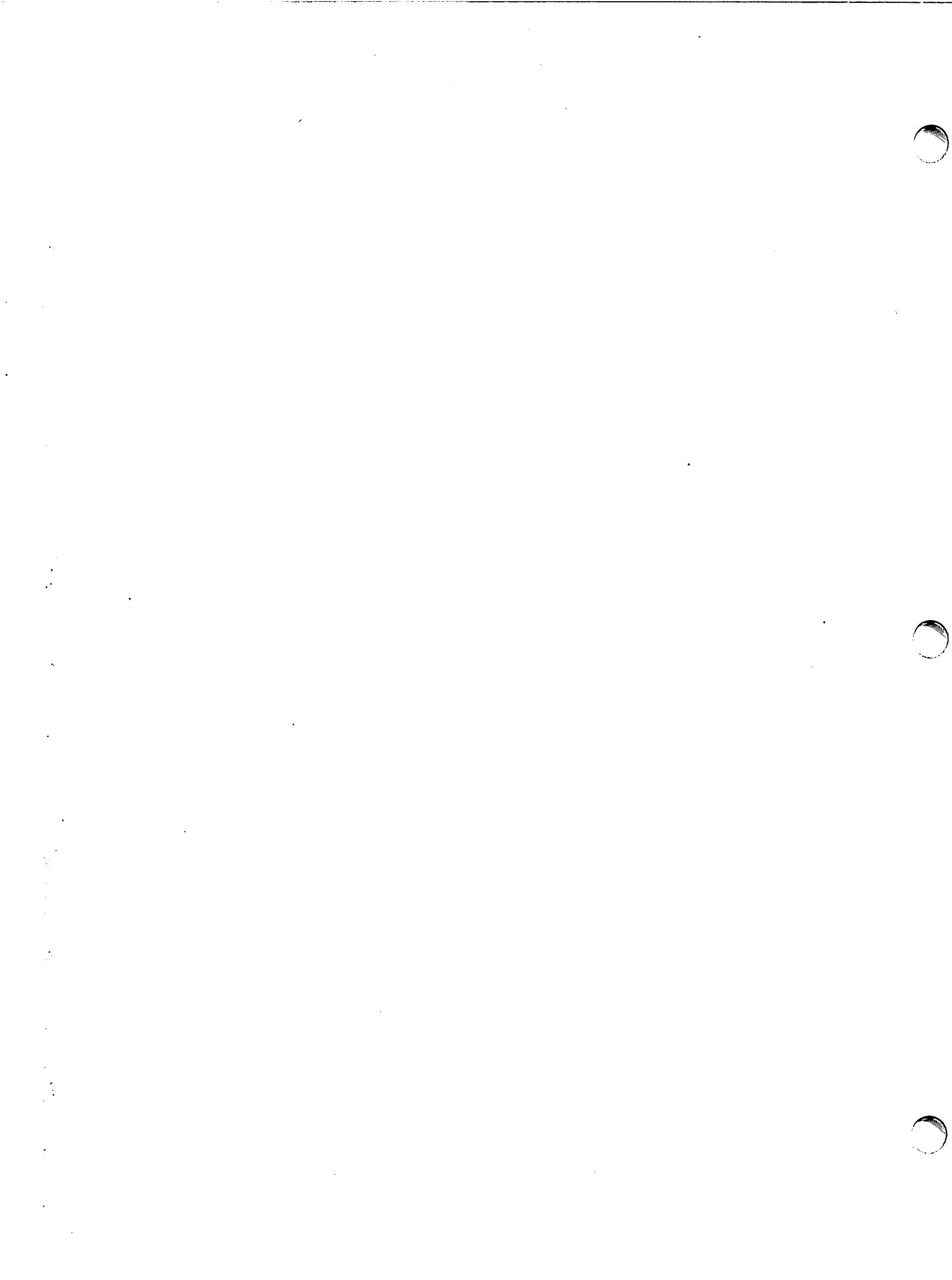
## SOME EXAMPLES OF USER-DETERMINED ALLOCATION\*

- 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 as many words for buffer space as there are words in each logical record of the file. Each "\*" in a FILES statement reserves 256 words of BUFFER space; each file and "\*" also reserves 15 words of table space.
- e) An approximate estimate of space required for a program is:  
  
11 words per BASIC statement  
 $+2X(\text{number of matrix elements dimensioned})$   
 $+1/2X(\text{number of string characters used})$

Semicompiled programs require slightly more space than that shown by the LEN command. CATALOG gives the actual length of CSAVED programs.

---

*\*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 BASIC program on paper tape for input:

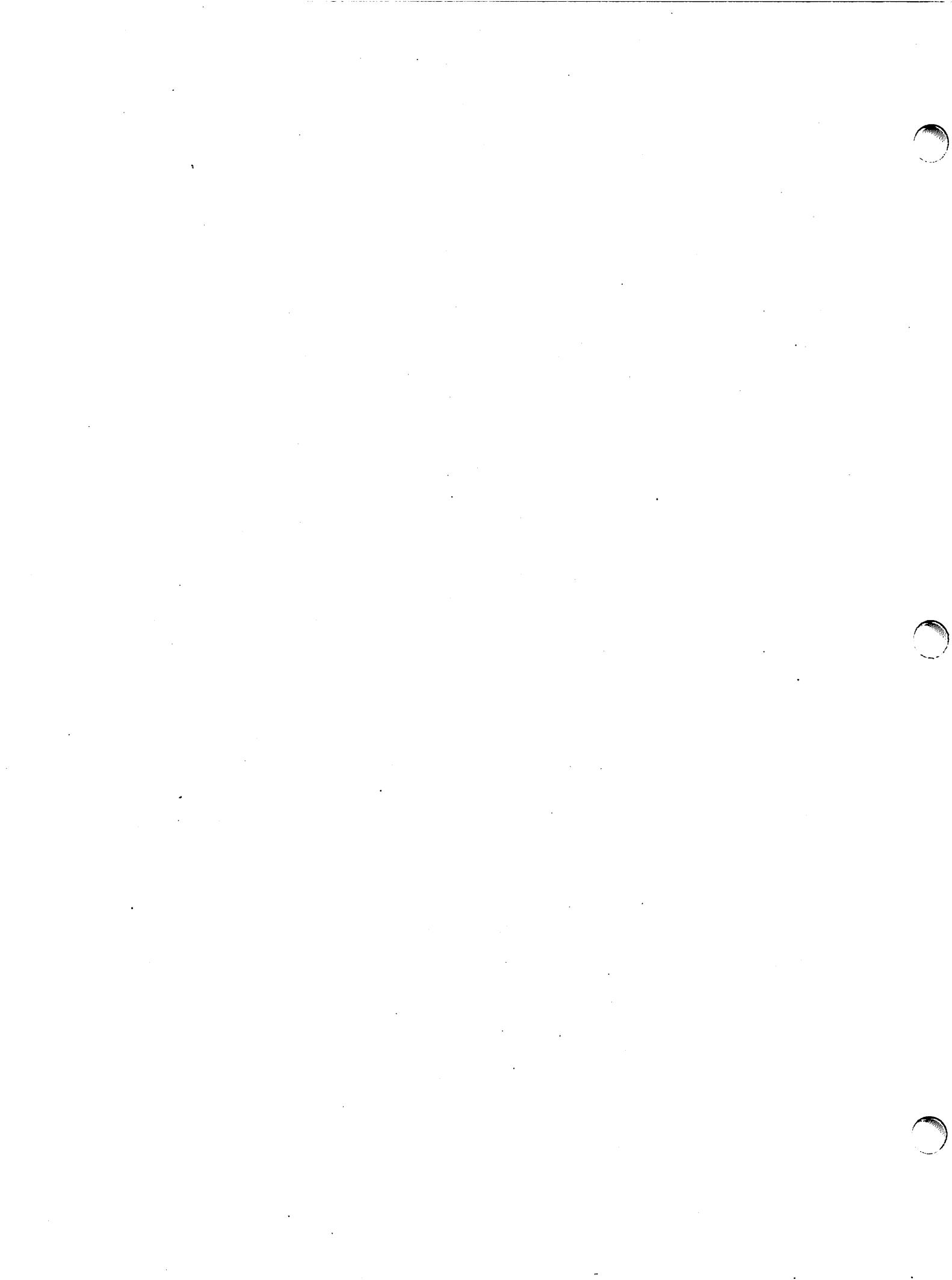
1. Turn teleprinter control knob to "LOCAL".
2. Press the "ON" button (on tape punch).
3. Press the "HERE IS" key; or press  $\text{@}^{\text{C}}$  (control shift "p") several times to put leading holes on the tape. The function of the "HERE IS" key may vary on some teleprinters.
4. Type the program as usual, following each line with return linefeed.
5. Press "HERE IS"; or press  $\text{@}^{\text{C}}$  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 esc,  $\leftarrow$ , and repeating the same line number may be punched on tape; esc 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.

Programs punched onto paper tape in the above manner, or produced by the PUNCH command, may be input to the system through the paper tape reader after typing the TAPE command. They may not be input as data using INPUT or ENTER statements. (See Section II and Appendix B.)



## APPENDIX B

### THE X-ON, X-OFF FEATURE

Terminals equipped with the X-ON, X-OFF feature must be used if it is desired 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

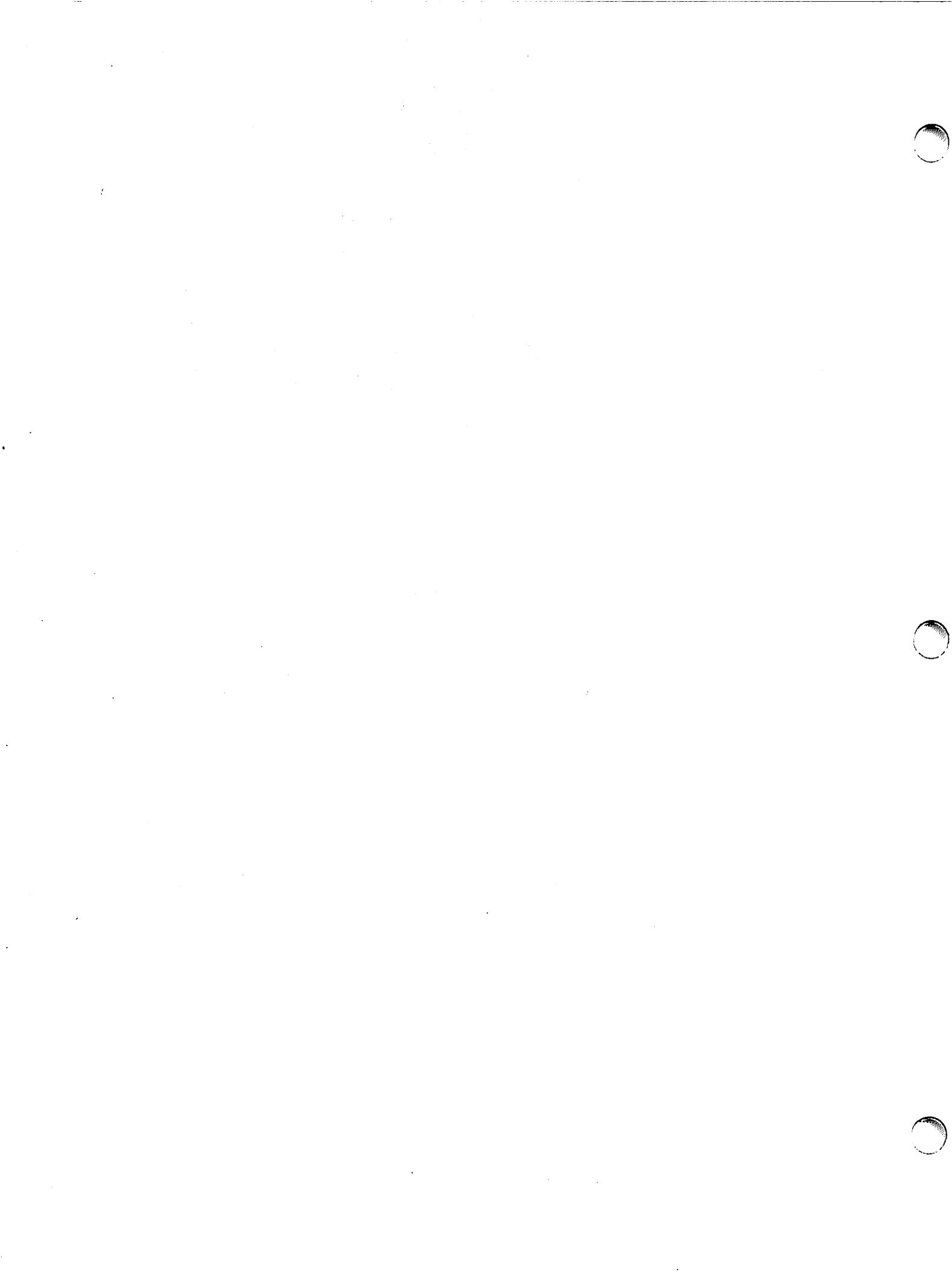
(x-off, return and linefeed are teleprinter keys.)

#### COMMENTS

Remember that each line of data must end with x-off return linefeed.

The X-OFF character causes the paper tape reader to stop reading tape after each carriage return until more input is requested by the program. Lines output by PRINT and PRINT USING statements are terminated by the X-off character.

Programs on paper tape produced by the XPUNCH command are in the correct format to be input as data strings from terminals with the X-ON, X-OFF feature. No line of such a program should exceed 72 characters (not counting X-OFF, carriage return, and linefeed), since each must fit into a single string. Programs produced by XPUNCH are not suitable for input in TAPE mode (Appendix A) from terminals with the X-ON, X-OFF feature.



# APPENDIX C

## DIAGNOSTIC MESSAGES

### User Command Error Messages

Error messages are listed below with the command which may invoke them. The message ILLEGAL FORMAT may be typed in response to many commands. The message PLEASE LOG IN is typed if a command (other than HELLO) or a line of syntax is entered from a port on which no user is logged in.

#### APPEND

INVALID NAME  
NO SUCH PROGRAM  
ILL-STORED PROGRAM  
ENTRY IS A FILE  
SEMI-COMPILED PROGRAM  
NO COMMON AREA ALLOWED  
PROGRAM TOO LARGE  
UNABLE TO RETRIEVE FROM LIBRARY  
SEQUENCE NUMBER OVERLAP

#### CATALOG

CAN'T READ DIRECTORY

#### CSAVE

See SAVE.

#### DELETE

NOTHING DELETED

#### GET

INVALID NAME  
NO SUCH PROGRAM  
ILL-STORED PROGRAM  
ENTRY IS A FILE  
PROGRAM TOO LARGE  
UNABLE TO RETRIEVE FROM LIBRARY

## DIAGNOSTIC MESSAGES (continued)

### GROUP

See CATALOG.

### HELLO

ILLEGAL ACCESS  
NO TIME LEFT

### KILL

ILLEGAL NAME  
NO SUCH ENTRY  
FILE IN USE

### LIBRARY

See CATALOG.

### LIST

RUN ONLY

### MESSAGE

CONSOLE BUSY

### NAME

ONLY 6 CHARACTERS ACCEPTED  
ILLEGAL FIRST CHARACTER

### OPEN

NAME TOO LONG  
ILLEGAL FIRST CHARACTER  
LIBRARY SPACE FULL  
SYSTEM OVERLOAD  
DUPLICATE ENTRY  
UNSUCCESSFUL; KILL AND REPEAT.

### PROTECT

PRIVILEGED COMMAND  
INVALID NAME  
NO SUCH ENTRY

## DIAGNOSTIC MESSAGES (continued)

### PUNCH

See LIST.

### RENUMBER

SEQUENCE NUMBER OVERFLOW/OVERLAP  
BAD PARAMETER

### RUN

See Execution Errors.

### SAVE

RUN ONLY  
NO PROGRAM NAME  
NO PROGRAM  
OUT OF STORAGE IN LINE n  
LIBRARY SPACE FULL  
SYSTEM OVERLOAD  
DUPLICATE ENTRY  
UNSUCCESSFUL; KILL AND REPEAT.

### UNPROTECT

See PROTECT.

### XPUNCH

See LIST.

### Language Processor Error Messages

The following messages are output by the BASIC language processor to indicate errors or possible errors in users' BASIC programs.

## DIAGNOSTIC MESSAGES (continued)

### Syntax Errors

One of the following error messages will be typed by the system after the entry of a BASIC statement with incorrect syntax. In all cases but the last, the line will be deleted.

OUT OF STORAGE  
ILLEGAL OR MISSING INTEGER  
EXTRANEOUS 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

## DIAGNOSTIC MESSAGES (continued)

MISSING OR BAD FILE REFERENCE  
'PRINT' MUST PRECEDE 'USING'  
ILLEGAL OPERAND AFTER 'USING'  
VARIABLE MISSING OR WRONG TYPE  
OVER/UNDERFLOWS - WARNING ONLY

### Execution Errors

This section lists messages output to indicate errors detected during program execution. Such errors cause termination of the execution.

UNDEFINED STATEMENT REFERENCE  
NEXT WITHOUT MATCHING FOR  
SAME FOR-VARIABLE NESTED  
FUNCTION DEFINED TWICE  
VARIABLE DIMENSIONED TWICE  
LAST STATEMENT NOT 'END'  
UNMATCHED FOR  
UNDEFINED FUNCTION  
ARRAY TOO LARGE  
ARRAY OF UNKNOWN DIMENSIONS  
OUT OF STORAGE  
DIMENSIONS NOT COMPATIBLE  
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

## DIAGNOSTIC MESSAGES (continued)

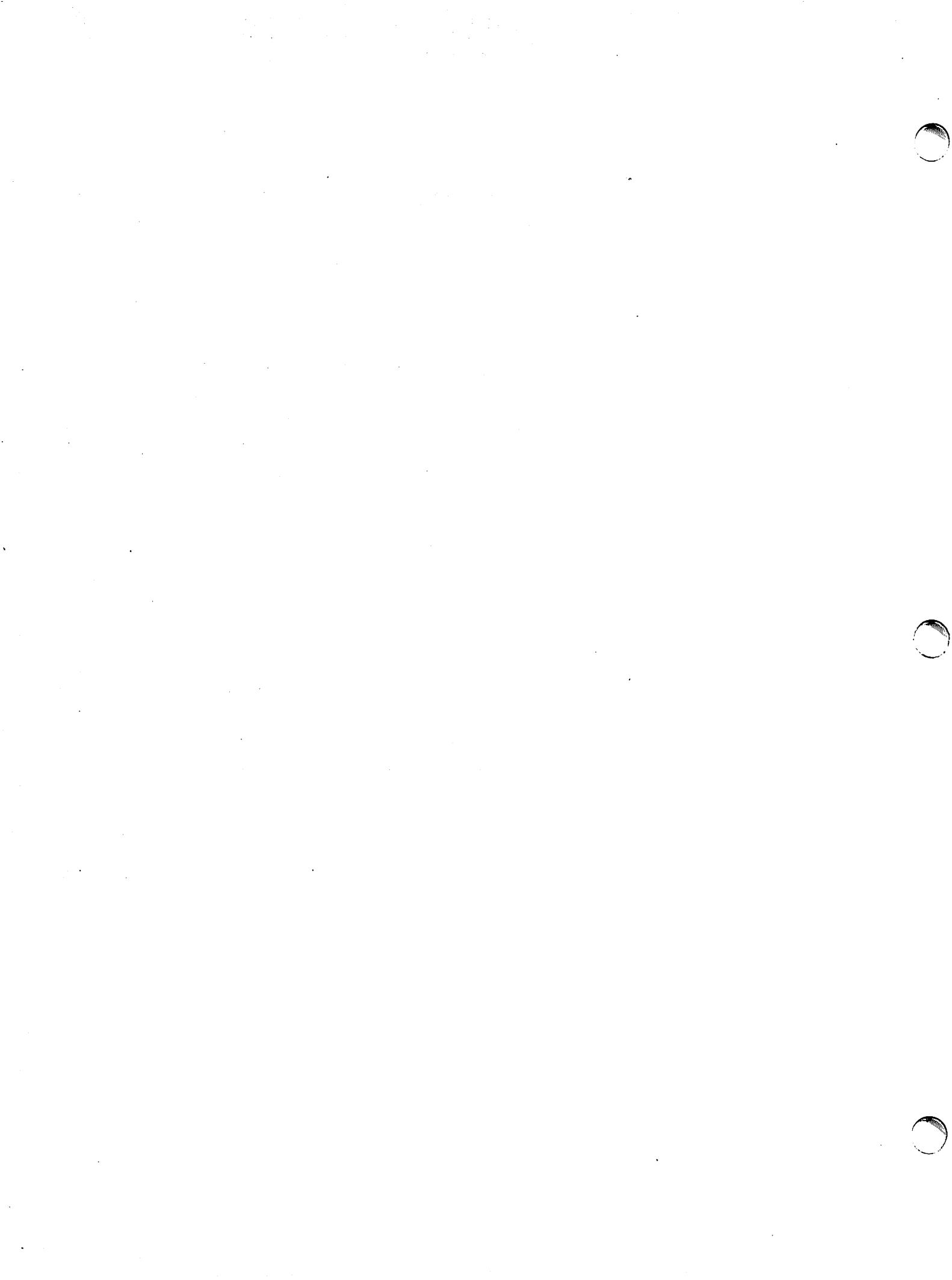
REDIMENSIONED ARRAY TOO LARGE  
NEARLY SINGULAR MATRIX  
LOG OF NEGATIVE ARGUMENT  
SQR OF NEGATIVE ARGUMENT  
ZERO TO ZERO POWER  
NEGATIVE NUMBER TO REAL POWER  
ARGUMENT OF SIN OR TAN TOO BIG  
TOO MANY FILES STATEMENTS  
NON-EXISTENT FILE REQUESTED  
WRITE TRIED ON READ-ONLY FILE  
END-OF-FILE/END OF RECORD  
STATEMENT NOT IMAGE  
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  
BAD FORMAT STRING SUBSCRIPT  
BAD FILE READ  
BAD FILE WRITE DETECTED  
CAN'T READ PROGRAM CHAINED TO  
ILL-STORED PROGRAM CHAINED TO  
PROGRAM BAD  
MISSING FORMAT SPECIFICATION  
ILLEGAL OR MISSING DELIMITER  
NO CLOSING QUOTE  
BAD CHARACTER AFTER REPLICATOR  
REPLICATOR TOO LARGE  
REPLICATOR ZERO  
MULTIPLE DECIMAL POINTS  
BAD FLOATING SPECIFICATION  
ILLEGAL CHARACTER IN FORMAT  
ILLEGAL FORMAT FOR STRING  
MISSING RIGHT PARENTHESIS  
MISSING REPLICATOR  
TOO MANY PARENTHESIS LEVELS  
MISSING LEFT PARENTHESIS  
ILLEGAL FORMAT FOR NUMBER

## DIAGNOSTIC MESSAGES (continued)

### Execution Warnings

The following messages are printed by the system to inform the user of conditions which may be unexpected or undesirable. These conditions do not terminate execution.

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:



## APPENDIX D

### ADDITIONAL LIBRARY FEATURES

Normally, programs and files in a user's library are stored on a mass storage device called a disc, which is external to the computer. Only the current program and portions of currently accessed files occupy the user's "working space" in the computer. TSB also makes use of another, usually smaller, mass storage device called a drum, on which many system tables are stored. There may also be room on the drum for a limited number of user programs and files. In certain cases, programs and particularly files which reside on the drum have improved (shorter) access times over those on the disc.

The system operator has control over placement and removal of programs and files on the drum. He also has several other program and file movement capabilities of which the user should be aware. These operator commands, and their functions, are listed here.

|           |  |
|-----------|--|
| SANCTIFY  | This command enables the operator to move a program (no longer than 8192 words) or a file (no longer than 32 records) from the disc to the drum. The area on the disc where it resided is retained. The entry will remain on the drum until it is removed by the operator (see below) or KILLED by the user who owns it. Only entries whose access times are critical should be sanctified.                    |
| DESECRATE | This command moves a sanctified file from the drum back to its original location on the disc, or deletes the drum copy of a sanctified program. (The disc copy of the program is retained.)<br><br><i>NOTE: If a sanctified <u>program</u> cannot be retrieved from a user's library because of a data error on the drum, it may be possible to DESECRATE the program and retrieve the copy from the disc.</i> |

## APPENDIX D

|              |   |
|--------------|---|
| BESTOW       | This command enables the operator to remove a program or file from one user's library and place it in another user's library, or to transfer ownership of an entire library.  |
| COPY         | This command is used to make a duplicate copy of any user program or file in the library of any other user (or the same user). The copy may be given a new name.  |
| LOAD<br>DUMP | The LOAD command enables the operator to load selected programs and files or entire user libraries from magnetic tape. DUMP allows the operator to write such programs, files or libraries onto magnetic tape. This can be done only at system start-up time (commonly once a day) and is a convenient way of transferring entries between 2000C systems, or dumping TSB files for other utility purposes. (See the "DOS-M TIME-SHARED BASIC FILE HANDLER" in the <i>SOFTWARE OPERATING PROCEDURES</i> .) |

*NOTE: Any of the above may be requested using the MESSAGE command. All pertinent idcodes and program or file names must be included.*

# INDEX

↑.....2-6  
←.....1-16  
.....1-17  
;.....2-27  
,.....2-22,2-24,2-27  
+.....2-6,5-12  
-.....2-6,5-13  
/.....2-6  
\*.....2-6,5-14,5-15  
=.....2-5,2-7,2-18,5-16,6-9  
#.....2-7,6-13  
<>.....2-7,6-13  
<.....2-7,6-13  
>.....2-7,6-13  
≥.....2-7,2-17,6-13  
≤.....2-7,2-17,6-13

**A**

ABS Function.....3-26  
Accuracy.....2-2  
Add.....2-6,5-12  
Adding Matrices.....5-12  
Adding to a Serial File.....4-20  
Advanced BASIC.....3-1  
Alphabetical File.....4-44  
alt-mode.....1-4  
AND Operator.....2-9  
APPEND Command.....3-13  
Arithmetic Evaluation.....2-4  
Arithmetic Operators.....2-6  
Arguments.....3-5  
Array.....3-3  
Assignment Operator.....2-5,6-9  
Assignment Statement.....2-14,6-9

ASSIGN Statement.....4-10  
ATN Function.....3-27

**B**

Backspace.....1-16  
BASIC.....1-6,1-7,2-1  
Before Going On-Line.....1-14  
BESTOW.....D-2  
break.....1-4,1-27,2-44  
BYE Command.....2-37

**C**

CATALOG Command.....3-15  
CHAIN Statement.....3-30  
Changing Characters.....1-16  
Changing Statements.....1-17  
Columns.....3-3,5-1  
Communication Between Programs..3-32  
COM Statement.....3-32  
Commands.....1-2,2-13,2-35  
Comments.....2-15  
Comparing Strings.....6-13  
Conditionals.....2-17  
Connection to the Computer.....1-20  
CONTENTS.....v  
Control C (CC)....1-4,1-26,1-27,1-29  
Control N (NC).....1-4  
Control O (OC).....1-4  
Control Characters.....1-21  
CONVENTIONS.....iv  
COPY.....D-2  
Copying a File.....4-33,D-2  
Copying a Matrix.....5-16  
COS Function.....3-27  
CSAVE Command.....3-10

|                           |                       |                                |                        |
|---------------------------|-----------------------|--------------------------------|------------------------|
| <u>ctrl</u> .....         | 1-4,1-21              | File Accessing Errors.....     | 4-45                   |
| Current Program.....      | 3-7                   | File Names.....                | 4-5                    |
| <b>D</b>                  |                       | File Numbers.....              | 4-3,4-8,4-10,4-12      |
| Data Set.....             | 1-20                  | File Pointer..                 | 4-2,4-3,4-14,4-16,4-29 |
| DATA Statement.....       | 2-21,2-22,6-15        | Files.....                     | 4-1,4-2,4-5            |
| Data Types.....           | 3-24,4-17,4-32        | FILES Statement.....           | 4-3,4-8                |
| Declaration of Files..... | 4-8                   | Fixed-Point Format.....        | 8-10,8-11              |
| DEF FN Statement.....     | 3-24                  | Floating-Point Format.....     | 8-12,8-13              |
| Definitions, Format.....  | 8-2                   | Format, Floating-Point.....    | 8-12                   |
| DELETE Command.....       | 3-14                  | Format, Fixed-Point.....       | 8-10                   |
| Deleting Characters.....  | 1-16                  | Format, Integer.....           | 8-8                    |
| Deleting Programs.....    | 3-7,3-12              | Format, Specifications..       | 8-6,8-8,8-10           |
| Deleting Statements.....  | 1-17                  |                                | 8-12,8-15,8-20         |
| DESECRATE.....            | D-1                   | Format, String.....            | 8-6                    |
| Diagnostic Messages.      | 1-25,4-45,C-1,ff      | Format, String Variable.....   | 8-20                   |
| Divide.....               | 2-6                   | Formatted Output.....          | 8-1,ff                 |
| DIM Statement.....        | 5-2,6-8               | FOR..NEXT Statements.....      | 2-18,2-20              |
| DUMP.....                 | D-2                   | FOR Statement.....             | 2-18                   |
| <b>E</b>                  |                       | Functions.....                 | 3-5,3-24,3-26,3-27     |
| ECHO Command.....         | 2-38                  | <b>G</b>                       |                        |
| End-of-File.....          | 4-19                  | GET Command.....               | 3-11                   |
| End-of-Record Marker..... | 4-27                  | GOSUB...RETURN Statements..... | 3-19                   |
| END Statement.....        | 2-30                  | GO TO Statement.....           | 2-16                   |
| E Notation.....           | 2-2                   | Greater Than.....              | 2-7,6-13               |
| ENTER.....                | 3-34                  | Greater Than or Equal To....   | 2-7,6-13               |
| Entering Commands.....    | 1-5                   | GROUP Command.....             | 3-15                   |
| Equality.....             | 2-7,6-13              | <b>H</b>                       |                        |
| Erasing a Record.....     | 4-41                  | HELLO Command.....             | 2-36                   |
| Error Messages.           | 1-25,8-27,8-28,C-1,ff | <b>I</b>                       |                        |
| <u>esc</u> .....          | 1-4,1-17              | IDcode.....                    | 1-21,1-22,2-36         |
| Essentials of BASIC.....  | 2-1                   | Identity Matrix.....           | 5-17                   |
| Execution.....            | 1-26                  | IDN.....                       | 5-17                   |
| EXP Function.....         | 3-26                  | IF END Statement.....          | 4-20                   |
| Exponentiate.....         | 2-6                   | IF..THEN Statement.....        | 2-17                   |
| Expression.....           | 2-4                   | IMAGE.....                     | 8-21                   |
| <b>F</b>                  |                       | Inequality.....                | 2-7,6-13               |
| False.....                | 2-7                   | Input Logs.....                | 1-22,1-23              |

|                                      |                |
|--------------------------------------|----------------|
| INPUT Statement.....                 | 2-24,5-5,6-10  |
| Inputting a String.....              | 6-10           |
| Inputting Matrix Elements....        | 5-5,5-6        |
| Instructions.....                    | 1-9            |
| INT Function.....                    | 3-26           |
| Introduction.....                    | 1-1            |
| INV.....                             | 5-19           |
| Inverting Matrices.....              | 5-19           |
| <b>K</b>                             |                |
| KEY Command.....                     | 2-48           |
| KILL Command.....                    | 3-12,4-7       |
| <b>L</b>                             |                |
| LEN Function.....                    | 3-28,6-14      |
| LENGTH Command.....                  | 3-8,6-14       |
| Length of Programs.....              | 3-8,3-28       |
| Less Than.....                       | 2-7,6-13       |
| Less Than or Equal To.....           | 2-7,6-13       |
| LET Statement.....                   | 2-5,2-14       |
| LIBRARY Command.....                 | 3-15           |
| Library Features, Additional....     | D-1            |
| Line Number.....                     | 1-8            |
| LIN Function.....                    | 8-3            |
| Linking Programs.....                | 3-30           |
| LIST Command.....                    | 1-18,2-40      |
| List Contents of a Record.....       | 4-37           |
| Listing a Program.....               | 1-18,2-40      |
| Listing a File.....                  | 4-12           |
| LOAD.....                            | D-2            |
| LOG Function.....                    | 3-26           |
| Logging In.....                      | 1-21,1-22,1-23 |
| Logging Out.....                     | 1-22           |
| Logical Length of Strings...6-3,6-14 |                |
| Logical Operations.....              | 7-1            |
| Logical Value.....                   | 7-1            |
| Loop.....                            | 1-27           |
| Looping.....                         | 2-20,3-22      |

## M

|                                |               |
|--------------------------------|---------------|
| Mathematical Functions.....    | 3-26          |
| MAT...CON Statement.....       | 5-4           |
| MAT INPUT Statement.....       | 5-6           |
| MAT PRINT Statement.....       | 5-8           |
| MAT PRINT# Statement.....      | 5-20          |
| MAT PRINT USING Statement..... | 8-19          |
| MAT READ Statement.....        | 5-11          |
| MAT READ# Statement.....       | 5-21          |
| MAT...ZER Statement.....       | 5-3           |
| Matrix.....                    | 3-4,5-1       |
| Matrix Addition.....           | 5-12          |
| Matrix Element.....            | 5-1           |
| Matrix File Print.....         | 5-20          |
| Matrix File Read.....          | 5-21          |
| Matrix Inversion.....          | 5-19          |
| Matrix Multiplication.....     | 5-14          |
| Matrix Subtraction.....        | 5-13          |
| Matrix Transposition.....      | 5-18          |
| Matrix Variable.....           | 5-2           |
| Maximum Number.....            | 2-2           |
| MAX Operator.....              | 2-8           |
| Memory Allocation.....         | 9-11          |
| MESSAGE.....                   | 2-50          |
| MIN Operator.....              | 2-8           |
| Minimum Number.....            | 2-2           |
| Modifying A Record.....        | 4-40,4-43     |
| Modifying a Serial File....    | 4-24,4-25     |
| Moving the Pointer.....        | 4-29          |
| Multibranch GOSUB.....         | 3-21          |
| Multibranch GOTO.....          | 2-16          |
| Multiply.....                  | 2-6,5-14,5-15 |
| Multiplying Matrices.....      | 5-14          |
| <b>N</b>                       |               |
| NAME Command.....              | 3-9           |
| Nested GOSUBS.....             | 3-22          |

|                                 |   |
|---------------------------------|---|
| Nesting Loops.....              | 2-20  |
| NEXT Statement.....             | 2-18  |
| NOT Operator.....               | 2-11  |
| Numbers.....                    | 2-2   |
| <b>O</b>                        |   |
| OPEN Command.....               | 4-5,4-6   |
| Operands.....                   | 1-10  |
| OR Operator.....                | 2-10  |
| Output, Formatted.....          | 8-1,ff  |
| <b>P</b>                        |   |
| Paper-Tape.....                 | A-1   |
| Parentheses.....                | 2-6,2-12  |
| Password.....                   | 1-21,2-36                                       |
| Physical Length of Strings..... | 6-3   |
| Pointers.....                   | 2-21  |
| Precedence.....                 | 2-6,2-12  |
| Precision.....                  | 2-2   |
| Printing Matrix Elements....    | 5-7,5-8<br>5-20                                 |
| Printing Strings.....           | 6-11  |
| PRINT Statement.....            | 2-26  |
| PRINT #..END Statement.....     | 4-21  |
| PRINT# Statement.....           | 4-37,4-38                                       |
| PRINT USING.....                | 8-17,8-18                                       |
| Program.....                    | 1-11,1-28                                       |
| Public Library Programs....     | 3-15,3-16<br>3-17                               |
| PUNCH Command.....              | 2-45  |
| <b>R</b>                        |   |
| Random File Access.....         | 4-35,4-36                                       |
| Random Numbers.....             | 3-26  |
| Reading Matrix Elements....     | 5-10,5-11<br>5-21                               |
| Reading Strings.....            | 6-12  |
| READ Statement.....             | 2-21,6-12                                       |
| READ# Statement.....            | 4-14  |
| READ#M; Statement.....          | 6-12  |
| READ#M,N Statement.....         | 4-33  |
| READ#M,N; Statement.....        | 4-33  |
| Records.....                    | 4-27  |
| Relational Operators.....       | 2-7,6-13<br>7-2,7-3                             |
| Remark.....                     | 2-15  |
| REM Statement.....              | 2-15  |
| RENUMBER Command.....           | 2-42  |
| Resetting the File Pointer..... | 4-16  |
| RESTORE Statement.....          | 2-21  |
| Retrieving Programs.....        | 3-11  |
| <u>return</u> .....             | 1-4,1-15  |
| RETURN.....                     | 3-19  |
| RND Function.....               | 3-26  |
| Routine.....                    | 3-2,3-18  |
| RUN Command.....                | 1-26,2-39                                       |
| Running a Program.....          | 1-26,2-34,2-39                                  |
| Run-Only.....                   | 3-13  |
| <b>S</b>                        |   |
| Sample Programs..               | 1-24,1-26,1-28,2-31<br>4-25,4-30,4-33,4-34,4-36 |
| SANCTIFY.....                   | D-1   |
| SAVE Command.....               | 3-10  |
| Scalar Multiplication.....      | 5-15  |
| SCRATCH Command.....            | 2-41  |
| Sequence of Statements.....     | 1-8   |
| Serial File Access....          | 4-3,4-10,4-12<br>4-14,4-16,4-22,4-23,4-31       |
| Serial File Print.....          | 4-12  |
| Serial File Read.....           | 4-14  |
| SGN Function.....               | 3-26  |
| SIN Function.....               | 3-27  |
| SPA Function.....               | 8-3   |
| Spaces.....                     | 1-12  |
| Special Keys.....               | 1-4   |
| Spot Checks.....                | 1-13,1-19                                       |
| SQR Function.....               | 3-26  |
| Statement Format.....           | 1-12  |
| Statement Numbers.....          | 1-8,2-42  |

|                                  |                      |                                |           |
|----------------------------------|----------------------|--------------------------------|-----------|
| Statement Types.....             | 1-9                  | Syntax Requirements of BASIC.. | 9-2,ff    |
| Statements.....                  | 1-7,2-13             | T                              |           |
| STEP.....                        | 3-23                 | TAB Function.....              | 8-3       |
| Stopping a Program.....          | 1-27,2-30            | TAN Function.....              | 3-27      |
| STOP Statement.....              | 2-30                 | TAPE Command.....              | 2-47      |
| Storage Requirements.....        | 4-28                 | Teleprinter.....               | 1-5       |
| Storing Programs.....            | 3-7,3-10             | Telephone.....                 | 1-1       |
| String Assignment Statement..... | 6-9                  | TIME Command.....              | 2-49      |
| String Comparison.....           | 6-13                 | Time-Out on Input.....         | 3-29      |
| String Evaluation.....           | 9-10                 | TYP Function.....              | 4-17,4-32 |
| String File Print.....           | 6-16                 | Time Sharing.....              | 1-1       |
| String File Read.....            | 6-17                 | TIM Function.....              | 3-29      |
| String Variable.....             | 6-3,6-6              | Transposing Matrices.....      | 5-18      |
| Strings.....                     | 3-5,4-28,6-1,6-2,6-6 | Trigonometric Functions.....   | 3-27      |
| Strings in DATA Statements.....  | 6-15                 | TRN.....                       | 5-18      |
| Strings in IF Statements.....    | 6-13                 | True.....                      | 2-7       |
| Structure of Serial Files.....   | 4-22                 | W                              |           |
| Subdividing Serial Files.....    | 4-31                 | Word.....                      | 3-6       |
| Subroutines.....                 | 3-18,3-19            | Working Size.....              | 5-2       |
| Subscripts.....                  | 5-1,6-4,6-6          | X                              |           |
| Substring.....                   | 6-4,6-6              | XPUNCH.....                    | 2-45      |
| Subtract.....                    | 2-6,5-13             | X-ON, X-OFF.....               | B-1       |
| Subtracting Matrices.....        | 5-13                 | Z                              |           |
|                                  |                      | Zeroing A Matrix.....          | 5-3       |

Notes:

Notes:

Notes:

over half — yard

**HEWLETT**  **PACKARD**

**READER COMMENT SHEET**

**2000C: A Guide To Time-Shared BASIC**

**02000-90016**

**April 1971**

Hewlett-Packard welcomes your evaluation of this text.  
Any errors, suggested additions, deletions, or general comments  
may be made below. Use extra pages if you like.

**FROM**

**NAME:** \_\_\_\_\_

**ADDRESS:** \_\_\_\_\_

**NO POSTAGE NECESSARY IF MAILED IN U.S.A.**

**FOLD ON DOTTED LINES AS SHOWN ON OTHER SIDE AND TAPE**

Gurney Seed & Nursery Co  
Yankton S.D.  
57078

FIRST CLASS  
PERMIT NO. 141  
CUPERTINO  
CALIFORNIA

**BUSINESS REPLY MAIL**

No Postage Necessary if Mailed in the United States Postage will be paid by

MANAGER, SOFTWARE PUBLICATIONS

HEWLETT - PACKARD  
CUPERTINO DIVISION  
11000 Wolfe Road  
Cupertino, California  
95014

02000-90016