A GUIDE TO TIME SHARED BASIC
PREFACE

The Time Shared Basic system (TSB) has provided a major breakthrough by reducing the cost of using a computer. Now, for the first time, it is practical for the programmer to use his time sharing terminal to teach himself more about the BASIC language.

Accordingly, this publication is designed to meet two requirements:
1. To serve as a clear and concise reference text for Time Shared BASIC; 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 Normal Form. Each element of a statement is underlined separately, in red.

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

Please turn to the next page.
CONVENTIONS USED IN THIS TEXT

<table>
<thead>
<tr>
<th>SAMPLE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLEASE LOG IN</td>
<td>Black, all capitals in examples indicates computer-output information</td>
</tr>
<tr>
<td>And then...</td>
<td>Mixed upper and lower case black is used for regular text.</td>
</tr>
<tr>
<td>2Ø PRINT X,Y</td>
<td>Red, all capitals indicates a statement or command typed by the programmer.</td>
</tr>
<tr>
<td>line number PRINT X,Y</td>
<td>Black lower case italics indicates a general form, derived from BASIC syntax requirements (Sect. VIII). Red underlining indicates an essential part of a general form; each underlined item is a separate, essential element.</td>
</tr>
<tr>
<td>return linefeed</td>
<td>Represents the terminal keys: Return, Linefeed, Escape, Control, Alt-Mode, and Break.</td>
</tr>
<tr>
<td>esc ctrl alt-mode break</td>
<td></td>
</tr>
<tr>
<td>Note: Both X and...</td>
<td>Mixed upper and lower case italics is used for notes.</td>
</tr>
</tbody>
</table>

LISTING A PROGRAM

| 0 | The letter "0" |
| 0 | Zeroes are slashed. |

Please examine the sample on the next page.
PAGE FORMAT

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

EXAMPLES:  
Several sample statements or commands

GENERAL FORM:  
(Each essential element underlined in red.)

PURPOSE

A clear and concise explanation of the purpose or function.

COMMENTS

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

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

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

Section No. __  Page No. __  (Revision Date)
HOW TO USE THIS BOOK

If your purpose is: Read:

Quickly acquiring a minimum working knowledge of Time Shared BASIC: Sections I and II.

Acquiring a good working knowledge of Time Shared BASIC: Sections I, II, III, IV, V, VI, in that order.

Learning the complete Time Shared BASIC system: The entire book, in sequence.

Reference only: 1. Contents

2. The index, Appendix "F"
3. The index tabs to locate the appropriate section.
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: FOR THE PROFESSIONAL

APPENDICES AND INDEX
CONTENTS

iii PREFACE
iv CONVENTIONS USED IN THIS TEXT
v PAGE FORMAT
vi HOW TO USE THIS BOOK

1-1 SECTION I
AN INTRODUCTION TO TIME SHARED BASIC

1-1 WHAT IS TIME SHARING?
1-2 COMMUNICATING WITH A COMPUTER
1-3 EXAMPLES OF BASIC STATEMENTS
1-4 STATEMENT NUMBERS
1-5 INSTRUCTIONS (STATEMENT TYPES)
1-6 OPERANDS
1-7 A PROGRAM
1-8 THE FORMAT OF STATEMENTS
1-10 BEFORE GOING ON-LINE
1-11 PRESS RETURN AFTER EACH STATEMENT
1-12 BACKSPACE
1-13 DELETING OR CHANGING A STATEMENT
1-14 LISTING A PROGRAM
1-16 CONNECTION TO THE COMPUTER
1-17 CHECKING THE CONNECTION
1-17 Your ID Code and Password
1-17 Control Characters
1-18 SAMPLE LOG IN AND LOG OUT
1-19 MISTAKES DURING LOG IN
1-20 ENTERING THE SAMPLE PROGRAM
1-21 HOW TO OBTAIN A DIAGNOSTIC MESSAGE
1-22 RUNNING THE SAMPLE PROGRAM
1-23 STOPPING A PROGRAM: THE break KEY
1-24 HOW THE PROGRAM WORKS
CONTENTS CONTINUED

2-1 SECTION II
THE ESSENTIALS OF BASIC

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

VIII 869
CONTENTS CONTINUED

2-40   RENUMBER
2-41   BREAK
2-42   PUNCH
2-43   TAPE
2-44   KEY
2-45   TIME

3-1   SECTION III
      ADVANCED BASIC

3-2   ROUTINE
3-3   ARRAY (OR MATRIX)
3-4   STRING
3-4   FUNCTION
3-5   WORD
3-5   RECORD
3-6   STORING AND DELETING PROGRAMS
3-7   LENGTH
3-8   NAME
3-9   SAVE
3-10   GET- AND GET-$
3-11   KILL-
3-12   APPEND-
3-13   DELETE-
3-14   LIBRARY
3-15   CATALOG
3-16   SUBROUTINES AND FUNCTIONS
3-17   GOSUB...RETURN
3-18   MULTIBRANCH GOSUB
3-19   NESTING GOSUB'S
3-20   FOR...NEXT WITH STEP
3-21   DEF FN
3-22   GENERAL MATHEMATICAL FUNCTIONS
3-23   TRIGONOMETRIC FUNCTIONS
3-24   THE TAB AND SGN FUNCTIONS
3-25   THE TYP FUNCTION
3-26   THE LEN FUNCTION

IX 869
CONTENTS CONTINUED

4-1  SECTION IV
     FILES

4-2  FILE
4-2  END OF RECORD
4-3  END OF FILE
4-3  SERIAL AND RANDOM ACCESS
4-4  OPEN-
4-5  KILL-
4-6  FILES
4-7  PRINT#
4-8  READ#
4-9  IF END#...THEN
4-10 STRUCTURE OF A FILE
4-11 STRUCTURE AND STORAGE PATTERN
4-12 SERIAL FILES
4-13 SUMMARY OF FILE STRUCTURE
4-14 PRINT#...END
4-15 PRINT#.......
4-16 PRINT TO RESET A POINTER
4-17 READ#.....
4-18 READ TO RESET A POINTER

5-1  SECTION V
     MATRICES

5-1  MATRIX (ARRAY)
5-2  DIM
5-3  MAT...ZER
5-4  MAT...CON
5-5  INPUT
5-6  MAT INPUT
5-7  PRINT MATRICES
5-8  MAT PRINT
5-9  READ
5-10 MAT READ
5-11 MATRIX ADDITION

X 869
CONTENTS CONTINUED

5-12 MATRIX SUBTRACTION
5-13 MATRIX MULTIPLICATION
5-14 SCALAR MULTIPLICATION
5-15 COPYING A MATRIX
5-16 IDENTITY MATRIX
5-17 MATRIX TRANSPOSITION
5-18 MATRIX INVERSION
5-19 MAT PRINT#
5-20 MAT READ#

6-1 SECTION VI
STRINGS

6-2 STRING
6-3 SUBSTRING
6-4 THE STRING DIM STATEMENT
6-5 THE STRING ASSIGNMENT STATEMENT
6-6 THE STRING INPUT STATEMENT
6-7 PRINTING STRINGS
6-8 READING STRINGS
6-9 STRING IF
6-10 THE LEN FUNCTION
6-11 STRING IN DATA STATEMENTS
6-12 PRINTING STRINGS ON FILES
6-13 READING STRINGS FROM FILES

7-1 SECTION VII
LOGICAL OPERATIONS

7-1 LOGICAL VALUES AND NUMERIC VALUES
7-2 RELATIONAL OPERATORS
7-4 BOOLEAN OPERATORS
7-5 SOME EXAMPLES

XI 869
CONTENTS CONTINUED

8-1 SECTION VIII
FOR THE PROFESSIONAL

8-2 SYNTAX REQUIREMENTS OF TSB
8-7 STRING EVALUATION BY ASCII CODES
8-8 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
SAMPLE PROGRAMS

D-1 APPENDIX D
DIAGNOSTIC MESSAGES

E-1 APPENDIX E
INSTANT GUIDE TO TIME SHARED BASIC

F-1 APPENDIX F
INDEX
SECTION I
AN INTRODUCTION TO TIME SHARED BASIC

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

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

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

WHAT IS TIME SHARING?

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

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

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

THE BASIC LANGUAGE

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

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

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

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

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

This is a BASIC statement:

10 INPUT A,B,C,D,E

COMMENTS

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

A statement may also be called a line.
STATEMENT NUMBERS

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

\[20 \text{ LET } S = \frac{(A+B+C+D+E)}{5}\]

COMMENTS

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

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

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

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

Continued on the next page.
INSTRUCTIONS (STATEMENT TYPES)

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

30 PRINT S

COMMENT

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

Continued on the next page.
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.

Continued on the next page.
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.

Continued on the next page.
THE FORMAT OF STATEMENTS

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

30 PRINT S
30 PRINT S
30 PRINT S

COMMENTS

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

Continued on the next page.
(Spot check)

Be sure you are familiar with these terms before continuing:

- statement
- instruction (statement type)
- statement type
- statement number (line number)
- operand
- program

All of these terms are defined in the context of this section.
The following pages explain the mechanics of entering, correcting, and checking statements.

Since you will probably have to make several corrections in your first attempts to use the computer, these features should be learned before beginning.
The \textit{return} key must be pressed after each statement.

Examples:

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

\textbf{COMMENTS}

Pressing \textit{return} informs the computer that the statement is complete. The computer then checks the statement for mistakes. (The checking process is explained later.)
The reverse arrow (←) key acts as a backspace, deleting the immediately preceding character.

Typing:       
is equivalent to typing:  
And typing:       
is equivalent to typing:  

```
2Ø LR+ET S=1Ø return
2Ø LET S=1Ø return
3Ø LET+ ← + PRINT S return
30 PRINT S return
```

**COMMENTS**

The + 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:

NOTE: The computer responds with a \ when esc is typed, like this:

To delete statement 5 in the sequence:

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

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

type:

Or, to change statement 5 in the above sequence, type:

The old statement is replaced by the new one.

Typing an esc (or alt-mode) before a return prevents replacement of a previously typed statement.

For example, typing:

or:

has no effect on the original statement 5.
LISTING A PROGRAM

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

\begin{quote}
\textbf{NOTE: The program has already been entered.}

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

\begin{verbatim}
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
\end{verbatim}

\textit{linefeed} indicates that the listing is complete.

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

\begin{quote}
A list of the same sample program produces these lines:

\begin{verbatim}
LIST-30 return
linefeed
linefeed
linefeed
linefeed
30 PRINT S
40 GO TO 10
50 END
\end{verbatim}
\end{quote}
1. Be sure you understand the use of these features before using the computer:

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

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

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

IF YOUR TERMINAL EQUIPMENT IS A TELEPRINTER WITH

ACOUSTIC COUPLER AND TELEPHONE:

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

HALF DUPLEX COUPLER AND TELEPHONE

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

DATA SET:

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

DIRECT CONNECTION TO THE COMPUTER:

Turn the teleprinter control knob to the LINE position.
CHECKING THE CONNECTION

The computer does not respond when the connection is established. If you wish to make sure that the connection has been made, type any number followed by a return.

EXAMPLE: 3 return

The computer then responds with the message:

PLEASE LOG IN return linefeed

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

This step is optional

YOUR 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 SEC₃CRC₆CT 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 computer which are acted upon (executed) immediately. Unlike statements, commands do not require line numbers.

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

```
linefeed
linefeed
linefeed
```

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

```
MESSAGE TO USERS FROM OPERATOR
```

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

To LOG OUT, type:

```
BYE return
```

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 computer 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 (not legal) to the Time Shared BASIC system.
ENTERING THE SAMPLE PROGRAM

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

NOTE: Connection to the computer is made.

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

or

READY return linefeed

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

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

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

30 PRINT S return linefeed

40 GO TO 10 return linefeed

50 END return linefeed

Now the program is ready to run.
**HOW TO OBTAIN A DIAGNOSTIC MESSAGE**

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

If the user types:

30 PRIMT S `return`

Note: `PRINT` has been misspelled. `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 return`

30 PRINT S

Appendix "D" contains a list of TSB diagnostic messages and probable causes.
RUNNING THE SAMPLE PROGRAM

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

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

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

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.

The answer is printed:

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

The program is finished:

Log off:

Time used is printed:
STopping A PROGRAM: THE break KEY

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

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

When a program is running or being listed, TSB responds with the message:

after break is pressed.

Remember that:

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

COMMENTS

break must be held down for at least 1/10 second.
HOW THE PROGRAM WORKS

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

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

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

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

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

30 PRINT S

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

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

40 GO TO 10

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

Line 50 informs the computer that the program is finished:

50 END
SECTION II
THE ESSENTIALS OF BASIC

HOW TO READ THIS SECTION

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

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

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

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

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

TERM: E NOTATION

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

EXAMPLES:

1.00000E+06 is equal to 1,000,000 and is read
   "1 times 10 to the sixth power (1x10^6).

1.02000E+04 is equal to 10,200

1.02000E-04 is equal to .000102

COMMENTS

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

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

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

2-2 869
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:

<table>
<thead>
<tr>
<th>Defined</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AØ</td>
<td>B</td>
</tr>
<tr>
<td>M5</td>
<td>C2</td>
</tr>
<tr>
<td>Z9</td>
<td>D</td>
</tr>
</tbody>
</table>

COMMENTS

Variables are used to represent numeric values. For instance, in the statement:

1Ø LET M5 = 96.7

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

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

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

**EXAMPLES:**

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

## TERM: ARITHMETIC EVALUATION

**DEFINED IN TSB AS:** The process of calculating the value of an expression.
THE ASSIGNMENT OPERATOR

<table>
<thead>
<tr>
<th>SYMBOL:</th>
<th>=</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLES:</td>
<td>10 LET A = B2 = C = 0</td>
</tr>
<tr>
<td></td>
<td>20 LET A9 = C5</td>
</tr>
<tr>
<td></td>
<td>30 Y = (N-(R+5))/T</td>
</tr>
<tr>
<td></td>
<td>40 N5 = A + B2</td>
</tr>
<tr>
<td></td>
<td>50 P5 = P6 = P7 = A = B = 98.6</td>
</tr>
<tr>
<td>GENERAL FORM:</td>
<td>LET variable = expression</td>
</tr>
</tbody>
</table>

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:

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:  
\[ \pm \times / \div + - \]

EXAMPIES:

\[
\begin{align*}
40 & \text{ LET } N1 = X-5 \\
50 & \text{ LET } C2 = N+3 \\
60 & \text{ LET } A = (B-C)/4 \\
70 & \text{ LET } X = ((P+2)-(Y*X))/N+Q \\
\end{align*}
\]

PURPOSE

Represents an arithmetic operation, as:

- exponentiate: \( \pm \)
- multiply: \( \times \)
- 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:

\[
\begin{align*}
\pm \\
\times / \\
+ - \\
\end{align*}
\]

with \( \pm \) having the highest priority. Operators on the same level of priority are acted upon from left to right in a statement. See "Order of Precedence" in this Section for examples.
### RELATIONAL OPERATORS

**Symbols:**

| =  | #  | <> | >  | <  | >= | <= |

**Examples:**

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

### Purpose

Determines the logical relationship between two expressions, as

- equality: =
- inequality: # or: <>
- greater than: >
- less than: <
- greater than or equal to: >=
- less than or equal to: <=

### Comments

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

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

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

EXAMPLES:

10 LET A=A9=P2=P5=C2=X=7.5
20 LET B5=D8=Q1=Q4=Y=B=12.0

80 PRINT (A MIN 10)
90 LET B=(A MIN 10)+100
100 IF (A MIN B5) > (C2 MIN D8) THEN 10
110 PRINT (X MAX Y)
120 IF (A9 MAX B) <= 5 THEN 150

PURPOSE

Selects the larger or smaller value of two expressions.

COMMENTS

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

**Symbol:** AND

**Examples:**

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.

2-10 869
# 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 = 0.

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

+ \quad \text{highest precedence}
NOT
* / 
+ - 
MIN MAX

Relational Operators
AND
OR \quad \text{lowest precedence}

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

5 + 6*7 \quad \text{is evaluated as: } 5 + (6\times7)
7/14*2/5 \quad \text{is evaluated as: } (7/14)\times2 \quad \frac{2}{5}

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

Parentheses override the order of precedence in all cases.
STATEMENTS

Be sure you know the difference between statements and commands.

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

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

Here are some examples mentioned in Section I:

<table>
<thead>
<tr>
<th>Statements</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>LET</td>
<td>HELLO</td>
</tr>
<tr>
<td>PRINT</td>
<td>BYE</td>
</tr>
<tr>
<td>INPUT</td>
<td>LIST</td>
</tr>
</tbody>
</table>

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 = Ø
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 (Ø) 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; however, they are ignored when the program is executing.

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

EXAMPLES:  
10 LET X = 20  
  :  
40 GO TO 3 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...transfers control to the statement number specified by the expression.

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.

The expression in a multibranch GO TO specifies the statement to which control is transferred. For example, statement 40 above transfers control to statement 430.

If the expression evaluates to a number greater than the number of statements specified, or less than 1, 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>=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:**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td><code>FOR P1 = 1 TO 5</code></td>
</tr>
<tr>
<td>110</td>
<td><code>FOR Q1 = N TO X</code></td>
</tr>
<tr>
<td>120</td>
<td><code>FOR R2 = N TO X STEP 1</code></td>
</tr>
<tr>
<td>130</td>
<td><code>FOR S = 1 TO X STEP Y</code></td>
</tr>
<tr>
<td>140</td>
<td><code>NEXT S</code></td>
</tr>
<tr>
<td>150</td>
<td><code>NEXT R2</code></td>
</tr>
<tr>
<td>160</td>
<td><code>NEXT Q1</code></td>
</tr>
<tr>
<td>170</td>
<td><code>NEXT P1</code></td>
</tr>
</tbody>
</table>

---

**Sample Program - Variable Number Of Loops**

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

**GENERAL FORM:**

```
statement number FOR simple variable = initial value TO final value
```

or

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

(Statements to be repeated)

```
statement number NEXT simple variable
```

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

PURPOSE

Allows repetition of a group of statements within a program.

COMMENTS

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

How the loop works:

The simple variable is assigned the value of the initial value; the value of the simple variable is increased by 1 (or by the 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.

```
10 FOR A1 = 1 TO 5
20 FOR B2 = N TO P
30 FOR C3 = X TO Y STEP R
     ...
80 NEXT C3
90 NEXT B2
100 NEXT A1
```

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

```
10 FOR I = 1 TO 5
     ...
30 FOR J = 1 TO N
     ...
50 NEXT I
     ...
90 NEXT J
```

The range of loops \{ I and J \} overlap.
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.
Sample Program Using READ, DATA and RESTORE

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

GENERAL FORM:

statement number READ variable , variable ,...
statement number DATA number or string , number or string ,...
statement number RESTORE
statement number RESTORE statement number

PURPOSE

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

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

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

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

COMMENTS

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

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

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

DATA statements do not execute; they merely specify data.

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

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

Programmers mixing string and numeric data may find the TYP function useful. See "The TYP Function", Section III.
This program shows several variations of the INPUT statement and their effects.

Sample Program Using INPUT

```plaintext
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
?2.5,3.5,4.5,6.7,7.2 return
??8.1 return ?? indicates that more input is expected
WHAT VALUE SHOULD BE ASSIGNED TO R? -99
1.5 2.5 3.5 4.5 6 7.2
8.1 R=-99

DONE

GENERAL FORM:

```
statement number INPUT variable , variable , ...
```

PURPOSE

Assigns a value input from the teleprinter to a variable.

Continued on next page.
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 return. Note that the C 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.
This sample program gives a variety of examples of the PRINT statement. The results are shown below.

10 LET A=B=C=10
20 LET D1=E9=20
30 PRINT A,B,C,D1,E9
40 PRINT A/B,B/C/D1+E9
50 PRINT "NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE"
60 PRINT "VALUE IN THE SAME STATEMENT."
70 PRINT
80 PRINT
90 REM* "PRINT" WITH NO OPERAND CAUSES THE TELEPRINTER TO SKIP A LINE.
100 PRINT "'A' DIVIDED BY 'E9' =";A/E9
110 PRINT "11111","22222","33333","44444","55555","66666"
120 PRINT "1111111; "22222" ;"33333" ;"44444" ;"55555" ;"66666"
130 END

--- RESULTS ---

RUN return
10 10 10 20 20
1 20.05
NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE VALUE IN THE SAME STATEMENT.
'A' DIVIDED BY 'E9' = .5
11111 22222 33333 44444 55555
66666
1111122222333334444445555566666
DONE

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

Continued on next page.
PRINT, CONTINUED

GENERAL FORM:

```
statement_number PRINT expression , expression , ...
or
statement_number PRINT "any text" : expression : ...
or
statement_number PRINT "text" : expression : "text" : "text" : ...
or
statement_number PRINT any combination of text and/or expressions
or
statement_number PRINT
```

PURPOSE

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

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

COMMENTS

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

EXAMPLES:

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

GENERAL FORM:

any statement number STOP
any statement number END

Highest statement number in program END

PURPOSE

Terminates execution of the program and returns control to TSB.

COMMENTS

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

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

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

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

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

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

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

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

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

RUN return

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY.
IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS.
PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.
HOW MANY NUMBERS DO YOU WANT TO AVERAGE?
O.K., TYPE IN ONE OF THE 5 NUMBERS AFTER EACH QUES. MARK.
DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER.
NOW, LET'S BEGIN

? 99 return
? 87.6 return
? 92.7 return
? 79.5 return
? 84 return

5 NUMBERS WERE INPUT.
THEIR SUM IS: 442.8
THEIR AVERAGE IS: 88.56
DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?
TYPE 1 IF YES, 0 IF NO
BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.
YOUR REPLY? 2 return
TO REITERATE, YOU SHOULD TYPE 1 IF YES, 0 IF NO.
YOUR REPLY? 1 return
HOW MANY NUMBERS DO YOU WISH TO AVERAGE? C return
DONE
COMMANDS

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

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

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

```
HEL-H2Ø0,SEC\ 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, POSCT return or HEL-D007, POSCT 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 master or operator.

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

EXAMPLE:

BYE return
009 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, 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:

Cc return
LIST

EXAMPLE:

LIST return
or
LIST-100 return

GENERAL FORM:

LIST statement number

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.

COMMENTS

A listing may be stopped by pressing the break key.

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

May be abbreviated to its first three letters.
SCRATCH

EXAMPLE:

SCRATCH return

or

SCR return

GENERAL FORM:

SCRATCH

or

SCR

PURPOSE

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

COMMENTS

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

EXAMPLES:

RENUMBER return

REN return

REN-100

REN-10, 1 return

REN-20, 50 return

GENERAL FORM:

REN

or

REN-number assigned to first statement

or

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

PURPOSE

Renumber statements in a Program.

COMMENTS

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

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

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

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

EXAMPLES: `break` (Press the `break` key.)

PURPOSE

Terminates a program being run.

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

COMMENTS

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

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

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

```
Cc return
```

which produces the DONE message.

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

2-41 869
PUNCH

EXAMPLES:

PUNCH return
PUN return
PUN-65 return
PUN-5 return

GENERAL FORM:

PUNCH
or
PUN
or

PUN-statement number at which PUNCHing is to begin

PURPOSE

Punches a program, onto paper tape; also
punches the program name, and leading and
trailing guide holes on the tape; lists the
program as it is punched.

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.

PUN-statement number causes the punching to
begin at the specified statement.
TAPE

EXAMPLES:  
TAPE return  
TAP return

GENERAL FORM:  
TAPE  
or  
TAP

PURPOSE

Informs the computer that following input  
is from paper tape.

COMMENTS

TAPE suppresses any diagnostic messages which are  
generated by input errors, as well as the automatic  
linefeed after return. The KEY command  
(KEY return ) or any other command, causes the di- 
agnostic 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 computer that following input will be from the teleprinter keyboard; used only after a TAPE (paper tape input) sequence is complete; causes error messages suppressed by TAPE to be output to the teleprinter.

COMMENTS

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

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

GENERAL FORM: TIME

PURPOSE

Produces listings of terminal time used since log on, and total time used for the account since the automatic accounting system was last reset to zero.

COMMENTS

Time used by each ID code is recorded automatically by TSB. The system operator controls the accounting system.
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 (OR MATRIX)

DEFINED IN TSB AS: An ordered collection of numeric data containing not more than 2500 elements (numeric values).

COMMENTS

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

Arrays may have one or two dimensions. For example,

\[
\begin{align*}
1.0 \\
2.1 \\
3.2 \\
4.3
\end{align*}
\]

is a one dimensional array, while

\[
\begin{align*}
6, 5, 4 \\
3, 2, 1 \\
0, 9, 8
\end{align*}
\]

is a two dimensional array.

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

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

COMMENTS

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

Quotation marks may not be used within a string.

TERM: FUNCTION

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

COMMENTS

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

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 function.
TERM: WORD

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

COMMENTS

Numbers require two words of storage each when stored as numbers.

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

TERM: RECORD

DEFINED IN TSB AS: A storage unit containing 64 2-character words.

COMMENTS

Further details on file storage are given in Section IV, "FILES".
Up to this point manipulation of programs has been limited to the "current" program, that is, the program being written or run at the moment. The only means of saving a program introduced thus far is the PUNCH command.

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

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

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

EXAMPLES:

LENGTH return
LEN return
0000 WORDS

GENERAL FORM: LEN return

PURPOSE

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

COMMENTS

Each user has a working "space" of approximately 5100 two character words. LEN is a useful check on total program length when writing long programs.
NAME-

EXAMPLE:

NAME-PROG.1  return
NAME-**GO**  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 teleprinter.

COMMENTS

The first character of the program named may not be a $.

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

EXAMPLES:
SAVE return
SAV return

GENERAL FORM:
SAVE
or
SAV

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

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

No 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".

3-9 869
**GET- AND GET-$**

**EXAMPLES:**

- GET-PROGRM return
- GET-MYPROG return
- GET-$PUBLIC return
- GET-$NAME$ return

**GENERAL FORM:**

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

**PURPOSE**

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

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

**COMMENTS**

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

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

EXAMPLE:

KILL-PROG12 return
KILL-EXAMPLE return
KILL-FILE10 return

GENERAL FORM:

KILL- program or file to be deleted
or
KILL- 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 is not recoverable unless:

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

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

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

GET-PROG** (Retrieves the program.)
(make changes)
KILL-PROG** (Deletes the stored version.)
SAVE (Saves the current version.)

3-11 869
APPEND-

EXAMPLES:  
APPEND-MYPROG  return
APP-MYPROG  return
APPEND-$PUBLIC  return
APP-$SYSLIB  return

GENERAL FORM:  
APPEND- program name  
or
APP-program name  
or
APP-$system  library  program  name

PURPOSE

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

COMMENTS

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

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

The $ in system library program names is needed to APPEND them. 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

EXAMPLES:

LIBRARY return
LIB return

BINOP 0594  CDETER 0706  CSHFLO 1598  CURFIT 1618  DIFFEQ 0133  DIVID 1367
FNCTS 0652  GEOMEN 0199  IN4  5440  IN5  5440  INHIL 0250  LINFIT 0492
ROMINT 0299  SQE  0209  STAT11 0568  TAB  098  YELLOW 0227  Z123  0413

GENERAL FORM:

LIBRARY
or
LIB

PURPOSE

Produces an alphabetical listing of TSB system library program and file names, followed by the size of each, in two-character words.

COMMENTS

Public library programs are available to users; typing:

GET-$ program name return

retrieves the specified program.

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

Certain programs designated "run-only" or by the system operator may be RUN but not listed, or punched.

LIBRARY listings may be terminated with the break key.

3-14 869
CATALOG

EXAMPLES: CAT return
          CATALOG return
          PROG1 0024  PROG2   2348  PROG3 1489

GENERAL FORM:

       CATALOG
       or
       CAT

PURPOSE

Produces an alphabetical listing of the names of the programs and files stored on-system, under the user's account name and size of each in two-character words.

COMMENTS

May be terminated with the break key.

Programs are accessed with the GET command.

Files are accessed with a FILES statement. See Section IV, "Files" for details.
The following pages show TSB features useful for repetitive operations -- subroutines, programmer-defined and standard functions.

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

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

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

EXAMPLE:

50 READ A2
60 IF A2<100 THEN 80
70 GOSUB 400

: 
380 STOP (STOP, END, or GO TO's frequently precede 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: \textit{statement number} GOSUB \textit{statement number starting subroutine}

: 

\textit{statement number} RETURN

PURPOSE

GOSUB transfers control to the specified statement number.

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

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

COMMENTS

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

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

GOSUB...RETURN's may be nested to a level of 9 (see the next page).

3-17 869
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

Transfers control to the statement number indicated by the expression following GOSUB.

COMMENTS

Subroutines should be exited only with a RETURN statement.

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

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

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

Statement numbers in the sequence following OF must be separated by commas.
NESTING GOSUB'S

EXAMPLES:

100 GOSUB 200

::

200 LET A = R2/7
210 IF A THEN 230
220 GOSUB 250

::

250 IF A>B THEN 270
260 RETURN
270 GOSUB 600

::

PURPOSE

Allows selective use of subroutines within subroutines.

COMMENTS

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

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

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

GENERAL FORM:
statement number FOR simple variable = expression TO expression STEP expression

PURPOSE

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

COMMENTS

The step size need not be an integer. For instance,
100 FOR N = 1 TO 2 STEP .01
is a valid statement which produces approximately 100 loop executions, incrementing N by .01 each time. Since no binary computer represents all decimal numbers exactly, round-off errors may increase or decrease the number of steps when a non-integer step size is used.

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

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

EXAMPLE:
60 DEF FNA (B2) = A+2 + (B2/C)
70 DEF FNB (B3) = 7*B3+2
80 DEF FNZ (X) = X/5

GENERAL FORM:

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

PURPOSE

Allows the programmer to define functions.

COMMENTS

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

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

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

causes infinite looping.

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

EXAMPLES:

642 PRINT EXP(N); ABS(N)
652 IF RND (0) >= .5 THEN 900
662 IF INT (R) # 5 THEN 910
672 PRINT SQR (X); LOG (X)

GENERAL FORM:
The general mathematical functions may be used as expressions, or as parts of an expression.

PURPOSE

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

ABS (expression) the absolute value of the expression;
EXP (expression) the constant e raised to the power of the expression value (in statement 642 above, e^N)
INT (expression) the largest integer ≤ the expression;
LOG (expression) the logarithm of the positively valued expression to the base e;
RND (expression) a random number between 1 and 0; the expression is a dummy argument;
SQR (expression) the square root of the positively valued expression.

COMMENTS

The RND function is not restartable; it is virtually impossible to duplicate a sequence of random numbers using RND. See Appendix C for an example of RND in a program.
TRIGONOMETRIC FUNCTIONS

EXAMPLES:

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

PURPOSE

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

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

COMMENTS

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

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

ATN returns the angle in radians.

See the next three pages for other standard functions.
THE TAB AND SGN FUNCTIONS

EXAMPLES:

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

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

TAB (expression indicating column number )
SGN (expression )

PURPOSE

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

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

EXAMPLES:
800 IF TYP (3) = 2 THEN 1000
850 PRINT TYP (N)
900 IF TYP (R) # X THEN 1200

GENERAL FORM:
TYP may be used as an expression or as part of an expression; the function form is:
TYP ( file number formula )

PURPOSE

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

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

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

COMMENTS

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

3-25 869
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.
SECTION IV
FILES

This section is divided into two parts:

The first part defines terms, and explains how to open, close, read, and write on a file. These pages contain the minimum information needed to use files. This part was designed to allow the problem-oriented user to quickly obtain minimal file access.

The second part, beginning with "Structure of a File", contains information helpful in gaining an understanding of TSB files. The programmer who intends to use files consistently for information storage and retrieval should make an effort to learn the structure of TSB files. Considerable time (both programmer and machine) can be saved if the programmer has a good understanding of files.

Note that special variations of READ and PRINT pertinent to files have been included in both the serial and random access sections.
TERM: FILE

DEFINED IN TSB AS:
A storage area in the TSB system, which may be accessed from a program. Data may be written on and read from files.

Smaller divisions within a file are called records and words.

File structure is explained later in this section.

TERM: END OF RECORD

DEFINED IN TSB AS:
A marker placed (by TSB) at the end of each record used in a file. The mark is a reference point for the computer, and is written by the computer when a record is full, or when the programmer has finished writing on a file record.
TERM: END OF FILE

DEFINED IN TSB AS:

A mark placed (by TSB) at the end of a file. The mark is a reference point for the computer, and may be placed by the computer when a file is full, or when the programmer is finished writing on a file.

TERM: SERIAL AND RANDOM ACCESS

DEFINED IN TSB AS:

These denote the two methods of using files mentioned previously. When files are used as serial devices the computer selects the appropriate location within the file to read or write data. Random file access means that the programmer chooses to control the internal location of data within a file.
OPEN-

EXAMPLES:
OPEN-FILE27, 85 return
OPEN-SAMPLE, 128 return
OPEN-**FI**, 10 return

GENERAL FORM:
OPEN- 1 to 6 character file name, number of 64-word records in file
OPEN- 1 to 6 character file name, number of 64-word records in file

PURPOSE

Opens and assigns a name to a file; reserves the specified number of 64-word records of storage for file contents (1 word = 2 teleprinter characters).

Places an "end of file" marker at the beginning of each record.

COMMENTS

The minimum number of records per file is 1.

The maximum number of records per file varies with computer options.* Contact the system operator for the specific number on your system.

The maximum number of files available to each user is determined by the system operator.

Files are accessible only to users with the same I.D. code as their creator.

* 90 to 128
KILL-

EXAMPLE:
KILL-NAMEXX  return
KIL-EXAMPLE  return
KIL-FILE10  return

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

PURPOSE

Deletes the named 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.

It is not possible to KILL a file while it is being accessed by another user.

KILL-should be used carefully, as the KILLED file is not recoverable unless a paper tape was previously punched, with the data on it.
FILES

EXAMPLE: 10 MATH, SCORE, AND, SQRT, NAME5, $DATA

GENERAL FORM:
statement number FILES maximum of 8 file names, separated by commas.

PURPOSE

Declares which files will be used in a program; TSB assigns a file reference number (from 1 to 8) to each file listed.

COMMENTS

The FILES statement may be used only once in a program; however, the same file name may be repeated in a FILES statement.

Files are referenced in the order in which they are listed in the FILES statement. For instance, in the example above,

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

prints the value of A on the file named SCORE.

Public files in the system library are "read only"; they are accessed with a FILES statement. Public file names must be preceded by a $, as the file DATA in the example above.

Users with the same ID code may share files. Only one user at a time may write on a shared file.
**PRINT #**

**EXAMPLES:**

- 125 PRINT #5; A1, B2; C
- 130 PRINT #1; B; C; D
- 140 PRINT #M+N; B

**GENERAL FORM:**

```
statement number PRINT# file number formula ; ...
```

**PURPOSE**

Prints variables or text on the file number specified in the file formula.

**COMMENTS**

Non-integer file formula numbers are rounded to the nearest integer (from 1 through 8), since a maximum of 8 files may be accessed by a single program.

The maximum capacity of a file varies with computer options from 90 to 128 64-word records. Consult your system operator for specific information.

There are several other variations of PRINT#. This is the easiest one to use -- it fills available space within the specified file; however, it is not always the most efficient form of a print-to-file.

Other versions of the print-to-file, are described in this section.
READ #

EXAMPLES:

65 READ#5; A,B,C
70 READ#3; B$
80 READ#N; A, B$, C(5,3)
90 READ#(N+1); A,B$, C(5,3)

GENERAL FORM:  statement number READ# file number formula ; ...

PURPOSE

Reads values consecutively from the specified file.

COMMENTS

Since a maximum of 8 files may be specified in the FILES statement, the file number formula should not exceed 8. Non-integer file formula numbers are rounded to the nearest integer.

Each item of data stored on a file may be read only once with this statement. Other, more selective versions of the read-from-file are described later in this section.
IF END# ...THEN

EXAMPLES:

300 IF END #N THEN 800
310 IF END #2 THEN 830
320 IF END #3 THEN 9999
...
800 LET N = N + 1
810 IF N > 8 THEN 9999
820 GO TO 1
830 PRINT #3; A,B,C
840 PRINT "DATA IS STORED"
...
9999 END

GENERAL FORM:

statement number IF END# file number formula THEN statement number

PURPOSE

Defines an exit procedure when an "end of file" mark is encountered; also detects "end of record" conditions.

COMMENTS

The IF END statement defines an exit procedure which remains in effect until another IF END statement is encountered. Subsequent IF END statements with the same file number formula are used to change the exit procedure.

The normal exit procedure when an "end of file" mark is encountered, and no IF END statement used, is termination of the program, and printing "END OF FILE/END OF RECORD IN STATEMENT XXXX".

See "Structure of a File" in this section for further details on using files as serial devices.

4-9 869
STRUCTURE OF A FILE

A simple method of using files is to treat them as "black boxes" which store information. By using only the statements presented on the preceding pages, you may PRINT and READ information on files.

There are disadvantages to this method:

a) File space may be wasted by creating files larger than necessary.

b) Time may be wasted if a file is too small: it terminates the program, and must be enlarged, and the program re-run.

It is much more efficient to use files as one, or a group of, random storage devices. The next pages show the structure of files as random devices.
COMMENT

--- and ---- indicate the order of record use when record numbers are not specified in PRINT and READ statements.

Each record contains 64 words of storage space.

One file contains a maximum of 90-128 records (see your system operator for the exact limit).

Continued on the next page.
SERIAL FILES

(See the previous page for details of the structure of a single file.)

File 1

File 2

Last File

End of File Mark

Up to 8 files may be accessed in series (or any other configuration) by using the IF END statement.

--- indicates the sequence of file access, using an IF END# statement, but not record-controlled PRINT or READ statements. For example, the sequence

100 PRINT #N; A,B,C
110 IF END #N THEN 800
:
800 LET N=N+1
810 GO TO 10

fills files sequentially, moving to the next file when the current file is filled.
FILE STRUCTURE-SUMMARY

Each file is made up of a maximum of 90 to 128 64-word records. (Consult your system operator for the exact figure.) Each word = 2 teleprinter characters.

Numerical data requires 2 words of file space. String data requires about 1/2 word of file space per string character.

The formula for determining the number of words needed to store strings is:

\[ 1 + \frac{\text{number of characters in string}}{2} + 1 \]

if the number of characters is odd.

or

\[ 1 + \frac{\text{number of characters in string}}{2} \]

if the number of character is even.

Each file has a "pointer" used to reference data printed on or read from that file. This pointer references data sequentially when statements described previously are used to access files.

The following pages describe how the programmer may access files at random, by manipulating file pointers.
**PRINT ...,END**

**EXAMPLES:**

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

**GENERAL FORM:**

```
statement number PRINT# file number formula ; items to be printed , END
```

**PURPOSE**

Places an "end of file" marker after the value written on the file; END is significant only when the "end of file" marker is the last item written.

**COMMENTS**

The "end of file" marker written by this statement is a logical marker, rather than a physical boundary marker.

The "end of file" is overlaid by the first item in the next PRINT statement. An "end of file" condition is generated only on READ attempts, or an attempt to PRINT beyond the physical boundary of a file.

PRINT#,END may be used to put an "end of file" mark in the middle of a data file, to be used as a flag for an "IF END#" statement.

The IF END# statement transfers control when this end marker is encountered.

4-14 869
PRINT#,......

EXAMPLES:
165 PRINT #N,X; G2,H,I, "TEXT"
170 PRINT #1,3; X, Y4, Z
175 PRINT #(N+1), (X+2); F,P5

GENERAL FORM:
statement number PRINT# file number formula , record number formula ; print list

PURPOSE
Prints to a specified file and specified record within that file; permits selective positioning of data within a file.

COMMENTS
The record number formula should evaluate to an integer between 1 and the number of records in the file. Non-integers are rounded to the nearest integer value.

The corresponding controlled READ statement works in the same manner as the controlled PRINT. See "READ"...,...." in this section for details.

PRINT to a specified record erases the record before writing the new information. PRINT without specifying a record fills the file sequentially.
PRINT TO RESET A POINTER

EXAMPLES:
320 PRINT #M+N, R+S
330 PRINT #(N-P),X
340 PRINT #5,1

GENERAL FORM:
statement number PRINT# file number formula , record number formula

PURPOSE

Resets the file pointer to the first position in the specified record.

Erases the contents of the specified record.

COMMENTS

File and record number formulas should evaluate to integers. Non-integers are rounded to the nearest integer value.

Only the contents of the specified record are erased; the rest of the file remains intact.

A specified record may be rewritten during a program without a separate "print-to-reset" by including the record number formula in the PRINT statement. Remember that PRINT without specifying a record must be used to fill a file sequentially.
READ #.....

EXAMPLES:

100 READ #2,3; A,B,C3,X$
110 READ #N,2; N1,N2,N3
120 READ #N,M; R2, P7, A$, T(3,5)
130 READ #(N+1);(M+2); X,Y$,Z(S,S)

GENERAL FORM:

statement number READ# file number formula, record number formula; variable, variable...

PURPOSE

Reads data from a specified record of a file.

COMMENTS

The record formula number should evaluate to an integer between 1 and the number of records allowed per file.

Non-integers are rounded to the nearest integer value.

To read data sequentially, use READ without specifying a record number.

The corresponding PRINT statement works in the same manner as the controlled READ. See "PRINT#.....,..." in this section for details.

Attempting to read an end-of-record mark generates an "end-of-file" condition.

4-17 869
READ TO RESET A POINTER

EXAMPLES:
410 READ #2, 3
420 READ #N, 1
430 READ #(N-P), 5
440 READ # N, P3

GENERAL FORM:
statement number READ# file formula number
or
statement number READ# file number formula , record number formula

PURPOSE

Resets the file pointer to the first position of the specified record.

COMMENTS

A specified record may be reread without resetting the pointer.

READ# to reset a file pointer does not erase the specified record.

File and record numbers should evaluate to integers. Non-integers are rounded to the nearest integer value.

Once a record is accessed, READ without specifying a record is used to read sequentially from the file.
SECTION V  
MATRICES

This section explains matrix manipulation. It is intended to show the matrix capabilities of TS8, and assumes that the programmer has some knowledge of matrix theory.

TERM: MATRIX (ARRAY)

DEFINED IN TS8 AS: An ordered collection of numeric data containing not more than 2500 elements (numeric values).

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

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

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

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

**EXAMPLES:**

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

**GENERAL FORM:**

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

or

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

**PURPOSE**

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

**COMMENTS**

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

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

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

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

The absolute maximum matrix size is 2500 elements; a matrix of this size is practical only in conjunction with a very small program.

5-2 869
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:

\begin{align*}
\text{statement number} & \quad \text{MAT} & \quad \text{matrix variable} & \quad = & \quad \text{CON} \\
\text{or} & \\
\text{statement number} & \quad \text{MAT} & \quad \text{matrix variable} & \quad = & \quad \text{CON} \ (\text{expression}) \\
\text{or} & \\
\text{statement number} & \quad \text{MAT} & \quad \text{matrix variable} & \quad = & \quad \text{CON} \ (\text{expression}, \text{expression})
\end{align*}

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.
MAT INPUT

EXAMPLES:

<table>
<thead>
<tr>
<th>Statement Number</th>
<th>MAT INPUT Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>355</td>
<td>MAT INPUT A</td>
</tr>
<tr>
<td>360</td>
<td>MAT INPUT B(5)</td>
</tr>
<tr>
<td>365</td>
<td>MAT INPUT Z(5,5)</td>
</tr>
<tr>
<td>370</td>
<td>MAT INPUT A(N)</td>
</tr>
<tr>
<td>375</td>
<td>MAT INPUT B(N,M)</td>
</tr>
</tbody>
</table>

GENERAL FORM:

statement number  MAT INPUT  matrix variable

or

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

or

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

PURPOSE

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

COMMENTS

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

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

Elements being input must be separated by commas.

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

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

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

EXAMPLES:

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

GENERAL FORM:

statement number PRINT matrix variable ( expression ) ...

or

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

PURPOSE

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

COMMENTS

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

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

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

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

EXAMPLES:

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

GENERAL FORM:

statement number MAT PRINT matrix variable

or

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

PURPOSE

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

COMMENTS

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

### EXAMPLES:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>900 READ A(6)</td>
<td></td>
</tr>
<tr>
<td>910 READ A(9,9)</td>
<td></td>
</tr>
<tr>
<td>920 READ C(X); P$; R7</td>
<td></td>
</tr>
<tr>
<td>930 READ C(X,Y)</td>
<td></td>
</tr>
<tr>
<td>940 READ Z(X,Y), P(R2, S5), X(4)</td>
<td></td>
</tr>
</tbody>
</table>

### 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:

\[
\text{statement number MAT matrix variable} = \text{matrix variable} - \text{matrix variable}
\]

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:

<table>
<thead>
<tr>
<th>Original</th>
<th>Transposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  3</td>
<td>1  4  7</td>
</tr>
<tr>
<td>4  5  6</td>
<td>2  5  8</td>
</tr>
<tr>
<td>7  8  9</td>
<td>3  6  9</td>
</tr>
</tbody>
</table>

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.
MATRIX INVERSION

EXAMPLES:

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

GENERAL FORM:

statement number MAT matrix variable = INV ( matrix variable )

PURPOSE

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

COMMENTS

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

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

EXAMPLES:
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

When printing on a specified file record, remember that each record holds a maximum of 32 numbers. Attempting a MAT PRINT of a matrix having more than 32 elements generates an error diagnostic, terminating the program.
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 32 numbers may be stored on a file record.

5-20 869
This section explains how to manipulate strings with BASIC statements. There is little difference in the form of statements manipulating strings and those used with numeric variables. One important difference however, is the use of subscripts to reference strings and substrings.

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 definitions of "string" and "substring", skip to "The String DIM Statement."
TERM: STRING

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

COMMENTS

Special purpose characters such as +, esc (or alt-mode) and quotation marks may not be used as string characters.

Apostrophes (single quotes) and control characters are legal string characters.

String variables must be a single letter (A to Z) followed by a $, for example: A$,Z$,X$. 
**TERM: SUBSTRING**

**DEFINED IN TSB AS:** A certain character or characters contained within a string.

**COMMENTS**

A substring is referenced by subscripts placed after the string variable. For example, if the string $Z$ = ABCDEFGH, the statement:

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

prints the substring:

```
BCDEF
```

Two subscripts specify the first and last characters of the substring.

Using single subscript, as:

```
310 PRINT Z$(3)
```

prints the substring:

```
CDEFGH
```

The single subscript identifies the first character of the substring; all characters after it are considered to be part of the substring.

Both strings and substrings may be used with relational operators.

A substring may be a single character; using string $Z$ above, substring $Z$(2,2) = B.

A substring may also be defined as a null string (no value, as distinguished from a blank space which has a value.) This is done by making the second subscript one less than the first, as: $A$(6,5). This is the only case in which a smaller second subscript is acceptable.
THE STRING DIM STATEMENT

EXAMPLES:

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

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

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

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

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

Matrix dimension specifications may be used in the same DIM statement as string dimensions (example statement 45 above).
THE STRING ASSIGNMENT STATEMENT

NOTE: These strings have been mentioned in a DIM statement

EXAMPLES:

```
200 LET A$ = "TEXT OF STRING"
210 B$ = "*** TEXT !!!"
220 LET C$ = A$(1,4)
230 D$ = B$(4)
240 F$(3,8)=N$
```

GENERAL FORM:

```
statement number LET string variable = " string value "
```

or

```
statement number LET string variable = string or substring variable
```

or

```
statement number string variable = " string value "
```

or

```
statement number string variable = string or substring variable
```

PURPOSE

Establishes a value for a string; the value may be a literal value in quotation marks, or a string or substring value.

COMMENTS

Strings contain a maximum of 72 characters, enclosed by quotation marks. Strings having more than 1 character must be mentioned in a DIM statement.

Special purpose characters, such as + or (esc or alt-mode) may not be string characters.

If the assigned value is longer than the string length, the assigned value is truncated at the appropriate point.
THE STRING INPUT STATEMENT

NOTE: These string variables have been mentioned in a DIM statement.

EXAMPLES:
50 INPUT R$
55 INPUT A$, B$, C9, D10
60 INPUT A$(1,5)
65 INPUT B$(3)

GENERAL FORM:
statement number INPUT string or substring variable...

PURPOSE

Allows string values to be entered from the teleprinter.

COMMENTS

Placing a single string variable in an INPUT statement allows the string value to be entered without enclosing it in quotation marks.

If multiple string variables are used in an INPUT statement, each string value must be enclosed in quotation marks, and the values separated by commas. The same convention is true for substring values. Mixed string and numeric values must also be separated by commas.

If a substring subscript extends beyond the boundaries of the input string, the appropriate number of blanks are appended.

Numeric variables may be used in the same INPUT statement as string variables (example statement 55 above).
PRINTING STRINGS

EXAMPLES:

105 PRINT A$
110 PRINT A$, B$, Z$
115 PRINT C$(8) "END OF STRING" B3, B4, A9
120 PRINT C$(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.

0C and N0 generate a return and linefeed respectively when printed as string characters.
READING STRINGS

EXAMPLES:

300 READ C$
305 READ X$, Y$, Z$
310 READ Y$(5), A,B,C5,N$
315 READ Y$(1,4)

GENERAL FORM:

statement number READ string or substring variable , string or substring variable , ...

PURPOSE

Causes the value of a specified string or substring variable to be read from a DATA statement.

COMMENTS

A string variable (to be assigned more than 1 character) must be mentioned in a DIM statement before attempting to READ its value.

String or substring values read from a DATA statement must be enclosed in quotation marks, and separated by commas. See "Strings in DATA Statements" in this section.

Only the number of characters specified in the DIM statement may be assigned to a string. Blanks are appended to substrings extending beyond the string dimensions.

Mixed string and numeric values may be read (example statement 310 above); see "The TYP Function", Section III for description of a data type check which may be used with DATA statements.
STRING IF

EXAMPLES:
340 IF C$<D$ THEN 800
350 IF C$>=D$ THEN 900
360 IF C$#D$ THEN 1000
370 IF N$(3,5)<R$(9) THEN 500
380 IF A$(10)="END" THEN 400

GENERAL FORM:
statement no. IF string variable relational oper. string var. THEN statement no.

PURPOSE

Compares two strings. If the specified condition is true, control is transferred to the specified statement.

COMMENTS

Strings are compared one character at a time, from left to right; the first difference determines the relation. If one string ends before a difference is found, the shorter string is considered the smaller one.

Characters are compared by their A.S.C.I.I. representation. See Section VII, "String Evaluation by ASCII Codes" for details.

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

6-9 869
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 ) ...
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:

\[
\text{statement number DATA } \" \text{string text}\" , \" \text{string text}\" \ldots
\]

PURPOSE

Specifies data in a program (string values may also be used as data).

COMMENTS

String values must be enclosed by quotation marks and separated by commas.

String and numeric values may be mixed in a single DATA statement. They must be separated by commas (example statement 520 above).

Strings up to 72 characters long may be stored in a DATA statement.

See "The TYP Function", Section III, for description of a data type (string, numeric) check which may be used with DATA statements.
PRINTING STRINGS ON FILES

EXAMPLES:
350 PRINT #5; "THIS IS A STRING."
355 PRINT #8; C$, B$, X$, Y$, Z$
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:

\[
1 + \frac{\text{number of characters in string}}{2} \quad \text{if the number of characters is even;}
\]

\[
1 + \frac{\text{number of characters in string} + 1}{2} \quad \text{if the number of characters is odd.}
\]

A maximum of 124 string characters may be stored on 1 file record.

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

6-12 869
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$, VS(2,7), R$(9)

GENERAL FORM:

statement no.  READ# file no. formula , record no. formula ; string or substring variable...
          or
statement no.  READ# file no. formula ; string or substring variable...

PURPOSE

Reads string and substring values from a file.

COMMENTS

String and numeric values may be mixed in a file and in a READ# statement; they must be separated by commas.

See "The TYP Function", Section III, for description of a data type check.
LOGICAL VALUES AND NUMERIC VALUES

When using the logical capability of Time Shared BASIC, be sure to distinguish between logical values and the numeric values produced by logical evaluation.

The logical value of an expression is determined by definitions established in the user's program.

The numeric values produced by logical evaluation are assigned by Time Shared BASIC. The user may not assign these values.

Logical value is the value of an expression or statement, using the criteria:

any nonzero expression value = "true"
any expression value of zero = "false"

When an expression or statement is logically evaluated, it is assigned one of two numeric values, either:

1, meaning the expression or statement is "true",
or
Ø, meaning the expression or statement is "false".
RELATIONAL OPERATORS

There are two ways to use the relational operators in logical evaluations:
1. As a simple check on the numeric value of an expression.

<table>
<thead>
<tr>
<th>EXAMPLES:</th>
<th>150 IF B=7 THEN 600</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200 IF A9#27.65 THEN 700</td>
</tr>
<tr>
<td></td>
<td>300 IF (Z/10)&gt;=0 THEN 800</td>
</tr>
</tbody>
</table>

When a statement is evaluated, if the "IF" condition is currently true (for example, in statement 150, if B = 7), then control is transferred to the specified statement.

Note that the numeric value produced by the logical evaluation is unimportant when the relational operators are used in this way. The user is concerned only with the presence or absence of the condition indicated in the IF statement.

Continued on the next page.
RELATIONAL OPERATORS CONTINUED

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

EXAMPLES:

610 LET X=27
615 PRINT X=27
620 PRINT X#27
630 PRINT X>=27

The example PRINT statements give the numeric values produced by logical evaluation. For instance, statement 615 is interpreted by TSB as "Print 1 if X equals 27, 0 if X does not equal 27." There are only two logical alternatives; 1 is used to represent "true", and 0 "false".

The numeric value of the logical evaluation is dependent on, but distinct from, the value of the expression. In the example above, X equals 27, but the numeric value of the logical expression X=27 is 1, since it describes a "true" condition.
BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.
1. As logical checks on the value of an expression or expressions.

   EXAMPLES:
   510 IF A1 OR B THEN 670
   520 IF B3 AND C9 THEN 680
   530 IF NOT C9 THEN 690
   540 IF X THEN 700

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

   Similarly, statement 540 is interpreted: "if X is true (has a nonzero 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 nonzero 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
   500 PRINT B AND C
   510 PRINT C OR B
   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".

   Note that the criteria for determining the logical values are:
   true = any nonzero expression value
   false = an expression value of 0.

   The numeric value 1 or 0 is assigned accordingly.
SOME EXAMPLES

These examples show some of the possibilities for combining logical operators in a statement.

It is advisable to use parentheses wherever possible when combining logical operators.

EXAMPLES:

310 IF (A9 MIN B7)<0 OR (A9 MAX B7)>100 THEN 900
310 PRINT (A>B) AND (X<Y)
320 LET C = NOT D
330 IF (C7 OR D4) AND (X2 OR Y3) THEN 930
340 IF (A1 AND B2) AND (X2 AND Y3) THEN 940

The numerical value of "true" or "false" may be used in algebraic operations. For example, this sequence counts the number of zero values in a file:

90 LET X = 0
100 FOR I = 1 TO N
110 READ #1; A
120 LET X = X+(A=0)
130 NEXT I
140 PRINT N; "VALUES WERE READ."
150 PRINT X; "WERE ZEROES."
160 PRINT (N-X); "WERE NONZERO."

Note that X is increased by 1 or 0 each time A is read; when A = 0, the expression A = 0 is true, and X is increased by 1.
SECTION VIII
FOR THE PROFESSIONAL

This section contains the most precise reference authority --
the syntax requirements of Time Shared BASIC. The syntax
requirements are explicit and unambiguous. They may be used
in all cases to clarify descriptions of BASIC language
features presented in other sections.

The other subsections give technical information of interest
to the sophisticated user.
SYNTAX REQUIREMENTS OF TSB

LEGEND

::= 'is defined as...'
| "or"
\< > enclose an element of Time Shared BASIC

LANGUAGE RULES

1. Exponents have 1 or 2 digit integers only.
2. A <parameter> primary appears only in the defining formula of a <DEF statement>.
3. A <sequence number> must lie between 1 and 9999 inclusive.
4. An array bound must lie between 1 and 9999 inclusive; a string variable bound must lie between 1 and 72 inclusive.
5. The character string for a <REM statement> may include the character ".
6. An array may not be transposed into itself, nor may it be both an operand and the result of a matrix multiplication.

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

Continued on the next page.
SYNTAX REQUIREMENTS OF TSB

<constant> ::= <number>|+<number>|<number>-<number>|<literal string>
<number> ::= <decimal number>|<decimal number><exponent part>
<decimal number> ::= <integer>|<integer>.<integer>|<integer>.<integer>.<integer>|<integer>
<integer> ::= <digit><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>
<character> ::= any ASCII character except null, line feed, return, x-off, alt-mode, escape, +, "", and rubout

<variable> ::= <simple variable>|<subscripted variable>
<simple variable> ::= <letter>|<letter><digit>
<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<subscripted variable> ::= <letter>(<sublist>)
<sublist> ::= <expression>|<expression>,<expression>
<string variable> ::= <string simple variable>|<string simple variable>(<sublist>)
<string simple variable> ::= <letter>$
<expression> ::= <conjunction>|<expression>OR<conjunction>
$conjunction$ ::= <relation>|<conjunction>AND<relation>
$relation$ ::= <minmax>|<minmax><relational operator><minmax>
$minmax$ ::= <sum>|<minmax>M<sum>|<minmax>M<sum>
=sum>|<sum>+<term>|<sum>-<term>
<term> ::= <subterm>|<term>*<subterm>|<term>/<subterm>
<subterm> ::= <denial>|<signed factor>

8-3 869
SYNTAX REQUIREMENTS OF TSB, CONTINUED

<denial> ::= <factor> | NOT <factor>
<signed factor> ::= +<factor> | -<factor>
<factor> ::= <primary> | <factor>+<primary>
<primary> ::= <variable> | <number> | <functional> | <parameter> (rule 2) | (<expression>)
<relational operator> ::= < | <= | >= | > |<>|
<parameter> ::= <letter> | <letter><digit>
<functional> ::= <function identifier>(<expression>) | <pre-defined function>(<expression>) | LEN (<string simple variable>)
<function identifier> ::= FN <letter>
<pre-defined function> ::= SIN | COS | TAN | ATN | EXP | LOG | ABS | SQR | INT | RND | SGN | TYP
<source string> ::= <string variable> | <literal string>
<destination string> ::= <string variable>
<file reference> ::= #<file formula> | #<file formula>, <record formula>
<file formula> ::= <expression>
<record formula> ::= <expression>
<array identifier> ::= <letter>
<sequence number> ::= <integer> (see rule 3)
<program statement> ::= <sequence number> <BASIC statement> carriage return
<BASIC statement> ::= <LET statement> | <IF statement> | <GOTO statement> | <GOSUB statement> | <RETURN statement> | <FOR statement> | <NEXT statement> | <STOP statement> | <END statement> | <DATA statement> | <READ statement> | <INPUT statement> | <PRINT statement> | <RESTORE statement> | <DIM statement> | <DEF statement> | <FILES statement> | <REM statement> | <MAT statement>

<LET statement> ::= LET <leftpart><expression> | LET <destination string>=<source string> | <leftpart><expression> | <destination string>=<source string>
<leftpart> ::= <variable>= | <leftpart><variable>=
<IF statement> ::= IF<decision expression>THEN<sequence number> | IF END #<file formula>THEN<sequence number>
<decision expression> ::= <expression> | <comparison string 1><relational operator><comparison string 2>
<comparison string 1> ::= <string variable>
<comparison string 2> ::= <string variable> | <literal string>

8-4 869
SYNTAX REQUIREMENTS OF TSB, CONTINUED

<GOTO statement> ::= GOTO <sequence number> |
                  GOTO <expression>OF <sequence list>
<sequence list> ::= <sequence number> | <sequence list>, <sequence number>
<GOSUB statement> ::= GOSUB <sequence number> |
                     GOSUB <expression>OF <sequence list>
<RETURN statement> ::= RETURN
<FOR statement> ::= FOR <for variable> = <initial value> TO <final value> |
                  FOR <for variable> = <initial value> TO <final value>
                  STEP <step size>
<for variable> ::= <simple variable>
<initial value> ::= <expression>
<final value> ::= <expression>
<step size> ::= <expression>
<NEXT statement> ::= NEXT <for variable>
<STOP statement> ::= STOP
<END statement> ::= END
.DATA statement ::= DATA <constant> | <DATA statement>, <constant>
<READ statement> ::= READ <variable list> | READ <file reference>
                  READ <file reference>; <variable list>
<variable list> ::= <read variable> | <variable list>, <read variable>
<read variable> ::= <variable> | <destination string>
<INPUT statement> ::= INPUT <variable list>
<PRINT statement> ::= <type statement> | <file write statement>
                   PRINT <file reference>
$type statement ::= <print 1> | <print 2>
<print 1> ::= PRINT | <print 2>, <print 2>; <print 3>
<print 2> ::= <print 1> | <print expression> | <print 3>
<print 3> ::= <type statement> | <literal string>
<print expression> ::= <expression> | TAB (<expression>) | <source string>
<file write statement> ::= PRINT <file reference> ; <write expression>
                           <file write statement> , <write expression>
                           <file write statement> ; <write expression>
                           <file write statement> ; <literal string>
                           <file write statement> <literal string>
                           <write expression>
<write expression> ::= <expression> | END | <source string>
<RESTORE statement> ::= RESTORE | RESTORE <sequence number>
SYNTAX REQUIREMENTS OF TSB, CONTINUED

<DIM statement> ::= DIM<dimspec>|<DIM statement>,<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<file name>|<FILES statement>,<file name>
<file name> ::= a string of 1 to 6 printing characters
<REM statement> ::= REM<character string> (see rule 5)
<MAT statement> ::= <MAT READ statement>|<MAT INPUT statement>|<MAT PRINT statement>|<MAT initialization statement>|<MAT assignment statement>

<MAT READ statement> ::= MAT READ<actual array>|MAT READ<file reference>;<actual array>|MAT READ statement>,<actual array>
<actual array> ::= <array identifier>|<array identifier>(<dimensions>)
<dimensions> ::= <expression>|<expression>,<expression>

<MAT INPUT statement> ::= MAT INPUT<actual array>|MAT INPUT statement>,<actual array>

<MAT PRINT statement> ::= <MAT PRINT 1>|<MAT PRINT 2>

<MAT PRINT 1> ::= MAT PRINT<array identifier>|MAT PRINT<file reference>;<array identifier>|MAT PRINT 2;<array identifier>

<MAT PRINT2> ::= <MAT PRINT 1>,<MAT PRINT 1>

<MAT initialization statement> ::= MAT<array identifier>=<initialization function>|MAT<array identifier>=<initialization function>(<dimensions>)
<initialization function> ::= ZER|CON|IDN

<MAT assignment statement>(rule 6) ::= MAT<array identifier>=<array identifier>|MAT<array identifier>=<array identifier><mat operator><array identifier>|MAT<array identifier>=INV(<array identifier>)|MAT<array identifier>=TRN(<array identifier>)|MAT<array identifier>=(<expression>)*<array identifier>

<mat operator> ::= +|-|*
STRING EVALUATION BY ASCII CODES


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

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

<table>
<thead>
<tr>
<th>Lowest:</th>
<th>bell</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>5</td>
</tr>
<tr>
<td>#</td>
<td>6</td>
</tr>
<tr>
<td>$</td>
<td>7</td>
</tr>
<tr>
<td>%</td>
<td>8</td>
</tr>
<tr>
<td>&amp;</td>
<td>9</td>
</tr>
<tr>
<td>'</td>
<td>:</td>
</tr>
<tr>
<td>(</td>
<td>;</td>
</tr>
<tr>
<td>)</td>
<td>(</td>
</tr>
<tr>
<td>*</td>
<td>)</td>
</tr>
<tr>
<td>+</td>
<td>*</td>
</tr>
<tr>
<td>,</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>,</td>
</tr>
<tr>
<td>.</td>
<td>-</td>
</tr>
<tr>
<td>/</td>
<td>.</td>
</tr>
<tr>
<td>Ø</td>
<td>/</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Quotation marks are used to delimit strings, and may not be used within a string.
MEMORY ALLOCATION BY A USER

Approximate number of 2-character words per user: 5,440
System overhead (approx.): 320
Space available for user allocation: 5,120 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 64 words for buffer space.

e) An approximate estimate of space required for a program is:
   11 words per BASIC statement
   +2X(number of matrix elements dimensioned)
   +1/2X(number of string characters used)

* This is variable "system overhead"; it is not included in word counts produced by the LEN command.
HOW TO PREPARE A PAPER TAPE OFF-LINE

To prepare a paper tape for input:

1. Turn teleprinter control knob to "LOCAL".

2. Press the "ON" button (on tape punch).

3. Press the "HERE IS" key; or press $\text{Ctrl C}$ (control shift "p") several times to put leading holes on the tape.

4. Type program as usual, following each line with \texttt{return linefeed}.

5. Press "HERE IS"; or press $\text{Ctrl C}$ 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 \texttt{esc}, +, and repeating the same line number may be punched on tape; \texttt{esc} must be followed by \texttt{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.
THE X-ON, X-OFF FEATURE

Terminals equipped with the X-ON, X-OFF feature may be used to input data from a paper tape while a program is running.

Data is punched on paper tape in this format:

- line of data items separated by commas
- x-off
- return
- linefeed

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

COMMENTS

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

See Appendix A, "Preparing A Paper Tape Offline," for instructions on editing a paper tape.
APPENDIX C

SAMPLE PROGRAM  LISTING A FILE

This program shows the use of the multi-branch GOTO and the TYP function.

9002 REM LISTS THE CONTENTS OF A FILE.
9003 LET N=1
9004 DIM A$[72]
9005 LET I=0
9006 LET I=I+1
9007 READ #N,I
9008 PRINT "**FILE#N"
9009 PRINT "RECORD#I"
9010 GOTO TYP(-N) OF 9011,9014,9017,9006
9011 READ #N;A
9012 PRINT A;
9013 GOTO 9010
9014 READ #N;A$
9015 PRINT A$
9016 GOTO 9010
9017 PRINT "**END OF FILE"N"**
9018 LET N=N+1
9019 GOTO 9005
9020 STOP
9999 END
This program calculates the interest rate of a loan. Note the checks included to guide the user -- they are skipped over if the data remains within pre-defined limits.

9003 PRINT "* TRUE ANNUAL INTEREST RATE *"
9004 PRINT
9010 PRINT "THIS PROGRAM CALCULATES THE TRUE ANNUAL INTEREST RATE"
9020 PRINT "ON AN INSTALLMENT LOAN"
9030 PRINT
9040 PRINT
9050 PRINT "IF YOU NEED INSTRUCTIONS TYPE 1, OTHERWISE TYPE 0: "
9060 INPUT X
9070 IF X=0 THEN 9120
9080 PRINT "TO USE THIS PROGRAM IT IS NECESSARY FOR YOU TO SUPPLY"
9090 PRINT "FOUR VARIABLES: A = AMOUNT OF LOAN (IN $), P = AMOUNT OF"
9100 PRINT "PAYMENT ($), N = THE TOTAL NUMBER OF PAYMENTS DUE, AND K = NUMBER"
9110 PRINT "OF PAYMENTS DUE IN ONE YEAR."
9115 PRINT
9120 PRINT "WHAT ARE A,P,N,K ";
9130 INPUT A,P,N,K
9140 PRINT
9150 IF N=1 THEN 9550
9160 IF P*N >= A THEN 9220
9170 PRINT
9180 PRINT "THATS NOT REASONABLE; THE PAYMENTS ADD UP TO LESS THAN THE AMOUNT"
9190 PRINT "OWED. TRY AGAIN."
9200 PRINT
9210 GOTO 9120
9220 LET R=0
9230 LET D=100
9240 GOSUB 9330
9250 IF P=P1 THEN 9430
SAMPLE PROGRAM- INTEREST RATES, CONTINUED

9260 IF P>P1 THEN 9290
9270 LET R=R-D
9280 GOTO 9300
9290 LET R=R+D
9300 LET D=D/2
9310 IF D<.0001 THEN 9430
9320 GOTO 9240
9330 LET R1=R/(100*K)
9340 LET Q=1+R1
9350 IF N*LOG(Q)/LOG(10) <= 75 THEN 9380
9360 LET P1=A*R1
9370 RETURN
9380 IF Q>1 THEN 9410
9390 LET P1=A/N
9400 RETURN
9410 LET P1=A*Q*N*R1/(Q*N-1)
9420 RETURN
9430 LET R=.01*INT(.5+100*R)
9440 IF R<199.5 THEN 9500
9450 PRINT
9460 PRINT "ARE YOU SURE THE DATA IS CORRECT? THE INTEREST RATE IS OVER"
9470 PRINT "200 PERCENT. TRY AGAIN."
9480 PRINT
9490 GOTO 9120
9500 PRINT "THE TRUE ANNUAL INTEREST RATE = ";R
9510 PRINT
9520 PRINT
9530 PRINT "ANOTHER CASE?? TYPE 'N' TO QUIT, 'Y' TO TRY AGAIN";
9532 INPUT Q$
9534 IF Q$="N" THEN 9999
9540 GOTO 9120
9550 LET R=(P/A-1)*K
9560 LET R=100*R
9570 GOTO 9430
9999 END

C-3 869
This program depends on a series of IF... THEN statements to find a day of the week. Note the use of the INT function, and the choice given in statements 9080 - 9082 in which only a "NO" reply is significant.

9002 PRINT "THIS PROGRAM DETERMINES THE DAY OF THE WEEK"
9003 PRINT "ON WHICH A GIVEN DATE FALLS."
9004 DIM A$(5)
9005 LET W=W1=0
9006 DIM F[12],L[12]
9007 MAT READ F
9008 MAT READ L
9009 GOTO 9076
9010 IF Y<1 THEN 9027
9011 IF M>12 THEN 9027
9012 IF M<1 THEN 9027
9013 IF D<1 THEN 9027
9014 IF Y>1752 THEN 9034
9015 IF Y<1582 THEN 9029
9016 IF Y=1752 THEN 9023
9017 IF Y=1582 THEN 9019
9018 GOTO 9031
9019 IF M<10 THEN 9029
9020 IF M>10 THEN 9031
9021 IF D<15 THEN 9029
9022 GOTO 9031
9023 IF M<9 THEN 9031
9024 IF M>9 THEN 9034
9025 IF D<14 THEN 9031
9026 GOTO 9034
9027 PRINT "UNACCEPTABLE DATA -- TRY AGAIN."
9028 GOTO 9078
9029 LET G1=0
9030 GOTO 9032
9031 LET G1=1
9032 LET J1=1
9033 GOTO 9036
9034 LET G1=1
9035 LET J1=0
9036 IF J1 <> 1 THEN 9054
9037 LET L1=0
9038 LET A=Y+INT((Y+3)/4)
9039 IF Y <> INT(Y/4)*4 THEN 9044
9040 LET L1=1
9041 IF M<3 THEN 9044
9042 LET L=1
9043 GOTO 9045
9044 LET L=0
9045 LET Z=A+D+L+F[M]+5
9046 LET Z=Z-INT(Z/7)*7
9047 LET Q=L[M]
9048 IF M <> 2 THEN 9050
9049 LET Q=Q+L1
9050 IF D>Q THEN 9027
9051 PRINT "OLD STYLE CALENDAR: ";
9052 LET W=1
9053 GOSUB 9089
9054 IF G1 <> 1 THEN 9080
9055 LET L1=0
9056 LET Y=Y-400*INT(Y/400)
9057 LET A=Y+INT((Y+3)/4)-INT((Y-1)/100)
9058 IF Y <> INT(Y/4)*4 THEN 9065
9059 IF Y=0 THEN 9061
9060 IF Y=100*INT(Y/100) THEN 9065
9061 LET L1=1
9062 IF M<3 THEN 9065
9063 LET L=1
SAMPLE PROGRAM  ELECTRONIC CALENDAR, CONTINUED

9064 GOTO 9066
9065 LET L=0
9066 LET Z=A+D+L+F[M]
9067 LET Z=Z-INT(Z/7)*7
9068 LET Q=L[M]
9069 IF M <> 2 THEN 9071
9070 LET Q=Q+L1
9071 IF D>Q THEN 9027
9072 IF W=0 THEN 9074
9073 PRINT "NEW STYLE CALENDAR: ";
9074 GOSUB 9089
9075 GOTO 9080
9076 PRINT "ENTER MONTH NUMBER, DATE, AND YEAR."
9077 PRINT
9078 INPUT M,D,Y
9079 IF W1=0 THEN 9010
9080 PRINT "IS THERE ANOTHER DATE YOU WANT TO KNOW";
9081 INPUT A$
9082 IF A$="NO" THEN 9999
9083 PRINT
9084 PRINT "ENTER DATE: ";
9085 LET W=0
9086 LET W1=1
9087 INPUT M,D,Y
9088 GOTO 9010
9089 GOTO Z+1 OF 9090,9092,9094,9096,9098,9100,9102
9090 PRINT "FRIDAY"
9091 RETURN
9092 PRINT "SATURDAY"
9093 RETURN
9094 PRINT "SUNDAY"
9095 RETURN

C-6 869
SAMPLE PROGRAM AN ELECTRONIC CALENDAR, CONTINUED

9096 PRINT "MONDAY"
9097 RETURN
9098 PRINT "TUESDAY"
9099 RETURN
9100 PRINT "WEDNESDAY"
9101 RETURN
9102 PRINT "THURSDAY"
9103 RETURN
9900 DATA 0,3,3,6,1,4,6,2,5,0,3,5
9901 DATA 31,28,31,30,31,30,31,31,31,30,31,30,31
9999 END
SAMPLE PROGRAM

H-P FOOTBALL

This program is a football game, based on the random number generator. Notice how the GO TO's and GOSUB' save repeating statements. Also note that the coin toss allows the user to specify a number, then generates as many random numbers, and uses the final number to determine the result.

10 PRINT "WELCOME TO THE CUPERTINO DIVISION FOOTBALL CHAMPIONSHIP GAME".
20 PRINT "THE DIVISION PLAYOFF IS BETWEEN THE HEWLETT HORNETS"
30 PRINT "AND THE PACKARD PANTHERS."
40 PRINT
50 PRINT "WE'LL NEED SOME HELP. WILL YOU CALL THE PLAYS FOR HEWLETT";
60 DIM W$[12]
70 INPUT W$
80 PRINT "FINE. THE COMPUTER WILL CALL THE PLAYS FOR PACKARD."
90 PRINT
100 PRINT "OK, COACH- FIRST, LET'S GET ACQUAINTED. WHAT'S YOUR NAME";
110 INPUT W$
120 PRINT "OK "W$", TYPE ONE OF THE PLAY NUMBERS FOLLOWED BY A RETURN"
130 PRINT "THE PLAY NUMBERS ARE:";
140 PRINT "1 = SIMPLE RUN; 2 = TRICKY RUN; 3 = SHORT PASS;"
150 PRINT "4 = LONG PASS; 5 = PUNT; 6 = QUICK KICK; 7 = PLACE KICK."
160 PRINT
170 LET Pl=51
180 LET Ql=0
190 LET T=0
200 LET S[1]=0
210 LET S[3]=0
220 PRINT "TOSS OF THE COIN-TYPE A NUMBER FROM 1 TO 300 (THEN RETURN)."
230 INPUT Z1
240 FOR I=1 TO Z1
250 LET X=RND(Q1)
260 NEXT I
270 IF RND(Q1)>1/2 THEN 300
280 PRINT "PACKARD WON THE TOSS."
SAMPLE PROGRAMS CONTINUED

H-P FOOTBALL

290 GOTO 1580
300 PRINT "HEWLETT WON THE TOSS."
310 PRINT "HEWLETT'S BALL ON ITS OWN 20."
320 LET P=1
330 LET X=20
340 LET X1=20
350 LET D=1
360 GOTO 1720
370 PRINT "CALL IT, "W$"."
380 INPUT Z
390 LET R=RND(Q1)
400 LET R=R*(.97+P*.03)
410 LET T=T+1
420 IF T<P1 THEN 540
430 PRINT P1-1; "PLAYS HAVE BEEN MADE. DO YOU WISH TO STOP NOW?"
440 PRINT "TYPE 1 FOR YES, Ø FOR NO.""YOUR REPLY";
450 INPUT D1
460 GOTO D1+1 OF 520, 490
470 GOTO 430
480 PRINT
490 PRINT "END OF GAME ***"
500 PRINT "FINAL SCORE: HEWLETT";S[3];" PACKARD";S[1]
510 STOP
520 PRINT "20 MORE PLAYS WILL BE ALLOWED."
530 LET P1=P1+20
540 LET R1=RND(Q1)
550 LET F=0
560 IF Z>4 THEN 620
570 IF Z=1 THEN 740
580 IF Z=2 THEN 790
590 PRINT "PASS PLAY-----";
600 IF Z=3 THEN 860
610 GOTO 1010
620 REM PUNT
630 LET Y=INT(100*(R-.5)+3+35)
640 IF Z=7 THEN 2330
650 IF D=4 THEN 670
660 LET Y=INT(Y*1.3)
670 PRINT "PUNT GOOD FOR"Y "YARDS"
680 IF D<4 THEN 720
690 LET Y1=INT(R1+2*20)+(1-P)*INT(R+2*30)
700 PRINT "RUN BACK FOR "Y1 "YARDS"
710 LET Y=Y-Y1
720 LET F=-1
730 GOTO 1180
740 REM SIMPLE RUN
750 PRINT "RUNNING PLAY--";
760 LET Y=INT(24*(R-.5)+3+3)
770 IF R1<.05 THEN 830

C-9 869
SAMPLE PROGRAMS CONTINUED

H-P FOOTBALL

780 GOTO 1070
790 REM TRICKY RUN
800 PRINT "RUNNING PLAY--";
810 LET Y=INT(20*R-5)
820 IF R1>.1 THEN 1070
830 LET F=-1
840 PRINT "*** FUMBLE AFTER ";
850 GOTO 1070
860 REM SHORT PASS
870 IF R<.05 THEN 920
880 IF R<.15 THEN 980
890 IF R<.55 THEN 950
900 PRINT "COMPLETE. ";
910 GOTO 1070
920 PRINT "INTERCEPTED. ";
930 LET F=-1
940 GOTO 1180
950 PRINT "INCOMPLETE. ";
960 LET Y=0
970 GOTO 1070
980 PRINT "PASSER TACKLED. ";
990 LET Y=-INT(10*R1)
1000 GOTO 1070
1010 REM LONG PASS
1020 LET Y=INT(160*(R1-.5)+30)
1030 IF R<.1 THEN 920
1040 IF R<.25 THEN 980
1050 IF R<.7 THEN 950
1060 GOTO 900
1070 REM RESULT OF PLAY
1080 LET X2=X+P*Y
1090 IF X2 >= 100 THEN 1260
1100 IF X2 <= 0 THEN 1760
1110 IF Y<0 THEN 1150
1120 IF Y=0 THEN 1170
1130 PRINT "GAIN OF"Y"YARDS"
1140 GOTO 1180
1150 PRINT "LOSS OF"-Y"YARDS"
1160 GOTO 1180
1170 PRINT "NO GAIN"
1180 LET X=X+P*Y
1190 IF X <= 0 THEN 1760
1200 IF X>=0 THEN 1230
1210 PRINT "BALL ON HEWLETT'S"X"YARD LINE. ";
1220 GOTO 1440
1230 IF X >= 100 THEN 1260
1240 PRINT "BALL ON PACKARD'S"100-X;"YARD LINE, ";
1250 GOTO 1440
1260 IF P<0 THEN 1340

C-10 869
SAMPLE PROGRAMS CONTINUED

H-P FOOTBALL

1270 IF F<0 THEN 1320
1280 PRINT "TOUCHDOWN!!!"
1290 LET P=-1
1300 GOSUB 2250
1310 GOTO 1580
1320 PRINT "TOUCHBACK FOR PACKARD."
1330 GOTO 1580
1340 IF F<0 THEN 1410
1350 PRINT "SAFETY!"
1360 GOSUB 2210
1370 PRINT "TOUCHDOWN FOR HEWLETT!!!"
1380 LET X=40
1390 LET P=1
1400 GOTO 1610
1410 PRINT "TOUCHDOWN HEWLETT!!!"
1420 GOSUB 2250
1430 GOTO 1580
1440 LET D=D+1
1450 IF F >= 0 THEN 1540
1460 IF P>0 THEN 1510
1470 PRINT
1480 PRINT "HEWLETT'S BALL."
1490 LET P=-1
1500 PRINT
1510 PRINT "PACKARD'S BALL."
1520 PRINT "PACKARD'S BALL ON ITS OWN 20."
1530 GOTO 1600
1540 IF P*(X-X1) >= 10 THEN 1610
1550 IF D<5 THEN 1720
1560 IF P<0 THEN 1470
1570 GOTO 1510
1580 LET X1=80
1590 PRINT "PACKARD'S BALL ON ITS OWN 20."
1600 LET P=-1
1610 LET D=1
1620 PRINT "FIRST DOWN."
1630 IF P<0 THEN 1670
1640 IF X<90 THEN 1700
1650 LET X1=90
1660 GOTO 1730
1670 IF X>10 THEN 1700
1680 LET X1=10
1690 GOTO 1730
1700 LET X1=X
1710 GOTO 1730
1720 PRINT "DOWN"D;"AND"10+P*(S1-S);"YARDS TO GO."
1730 PRINT
1740 IF P>0 THEN 370
1750 GOTO 1940
SAMPLE PROGRAMS CONTINUED

H-P FOOTBALL

1760 IF F<0 THEN 1880
1770 IF P<0 THEN 1820
1780 PRINT "TOUCHDOWN!!"
1790 LET P=1
1800 GOSUB 2250
1810 GOTO 310
1820 PRINT "SAFETY!!"
1830 GOSUB 2210
1840 PRINT "PACKARD GETS THE BALL ON ITS OWN 40."
1850 LET X=60
1860 LET P=-1
1870 GOTO 1610
1880 IF P>0 THEN 1910
1890 PRINT "TOUCHBACK FOR HEWLETT."
1900 GOTO 310
1910 PRINT "TOUCHDOWN PACKARD!!"
1920 GOSUB 2250
1930 GOTO 310
1950 LET P=-1
1960 IF D>1 THEN 2020
1970 IF RND(Q1)>1/3 THEN 2000
1980 LET Z=3
1990 GOTO 2190
2000 LET Z=1
2010 GOTO 2190
2020 IF D<4 THEN 2090
2030 IF X <= 30 THEN 2060
2040 LET Z=5
2050 GOTO 2190
2060 IF 10+X-X1<3 THEN 1970
2070 LET Z=7
2080 GOTO 2190
2090 IF 10+X-X1<5 THEN 1970
2100 IF X>X1 THEN 2160
2110 IF RND(Q1)>1/2 THEN 2140
2120 LET Z=2
2130 GOTO 2190
2140 LET Z=4
2150 GOTO 2190
2160 IF RND(Q1)>1/4 THEN 2180
2170 GOTO 2120
2180 GOTO 2140
2190 GOTO 390
2200 REM KEEP SCORE
2210 LET S'[2-P]=S'[2-P]+7
2220 PRINT "SCORE: HEWLETT "S[3];"PACKARD"S[1]
2230 PRINT
2240 RETURN
2250 IF RND(Q1)>.8 THEN 2290

C-12 869
SAMPLE PROGRAMS CONTINUED

H-P FOOTBALL

2260 PRINT "KICK IS GOOD"
2280 GOTO 2220
2290 PRINT "KICK IS OFF TO THE SIDE"
2310 GOTO 2210
2320 PRINT
2330 REM FIELD GOAL
2340 PRINT "PLACE KICK"
2350 LET F=-1
2360 IF R>.15 THEN 2390
2370 PRINT "KICK IS BLOCKED***"
2380 GOTO 1180
2390 IF P<0 THEN 2500
2400 IF X+Y >= 110 THEN 2460
2410 IF X+Y<0 THEN 2440
2420 PRINT "KICK IS OFF TO THE SIDE"
2430 GOTO 1320
2440 PRINT "KICK IS OFF TO THE SIDE"
2450 GOTO 1180
2460 PRINT "FIELD GOAL!!!"
2480 GOSUB 2220
2490 GOTO 1580
2500 IF X-Y <= -10 THEN 2540
2510 IF X-Y>20 THEN 2440
2520 PRINT "KICK IS OFF TO THE SIDE."
2530 GOTO 1890
2540 PRINT "FIELD GOAL!!!"
2560 GOSUB 2220
2570 GOTO 310
2580 END

C-13 869
<table>
<thead>
<tr>
<th>Diagnostic Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument of Sin or Tan too big</td>
</tr>
<tr>
<td>Array of unknown dimensions</td>
</tr>
<tr>
<td>Array too large</td>
</tr>
<tr>
<td>Bad format in files statement</td>
</tr>
<tr>
<td>Bad input, retype from item</td>
</tr>
<tr>
<td>Characters after command end</td>
</tr>
<tr>
<td>Characters after statement end</td>
</tr>
<tr>
<td>Data of wrong type</td>
</tr>
<tr>
<td>Dimensions not compatible</td>
</tr>
<tr>
<td>Divide by zero - warning only</td>
</tr>
<tr>
<td>End-of-file/end of record</td>
</tr>
<tr>
<td>Exp overflow - warning only</td>
</tr>
<tr>
<td>Extra input - warning only</td>
</tr>
<tr>
<td>ExTRANEOUS list delimiter</td>
</tr>
<tr>
<td>Function defined twice</td>
</tr>
<tr>
<td>GOSUBS nested ten deep</td>
</tr>
<tr>
<td>Illegal exponent</td>
</tr>
<tr>
<td>Illegal or missing integer</td>
</tr>
<tr>
<td>Illegal read variable</td>
</tr>
<tr>
<td>Illegal symbol follows 'MAT'</td>
</tr>
<tr>
<td>Last input ignored, retype it</td>
</tr>
<tr>
<td>Last statement not 'END'</td>
</tr>
<tr>
<td>Log of negative argument</td>
</tr>
<tr>
<td>Log of zero - warning only</td>
</tr>
<tr>
<td>Matrix cannot be on both sides</td>
</tr>
<tr>
<td>Matrix not square</td>
</tr>
<tr>
<td>Missing assignment operator</td>
</tr>
<tr>
<td>Missing left parenthesis</td>
</tr>
<tr>
<td>Missing or bad array variable</td>
</tr>
<tr>
<td>Missing or bad file reference</td>
</tr>
<tr>
<td>Missing or bad function name</td>
</tr>
<tr>
<td>Missing or bad list delimiter</td>
</tr>
<tr>
<td>Missing or bad simple variable</td>
</tr>
<tr>
<td>Missing or bad string operand</td>
</tr>
<tr>
<td>Missing or illegal data item</td>
</tr>
<tr>
<td>Missing or illegal 'OF'</td>
</tr>
<tr>
<td>Missing or illegal 'STEP'</td>
</tr>
<tr>
<td>Missing or illegal subscript</td>
</tr>
<tr>
<td>Missing or illegal 'THEN'</td>
</tr>
<tr>
<td>Missing or illegal 'TO'</td>
</tr>
<tr>
<td>Missing or protected file</td>
</tr>
<tr>
<td>Missing relational operator</td>
</tr>
<tr>
<td>Missing right parenthesis</td>
</tr>
<tr>
<td>Nearly singular matrix</td>
</tr>
<tr>
<td>Diagnostic Messages Printed While Entering a Program Refer Only to the First Error Found in a Line.</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Diagnostic messages printed while entering a program refer only to the first error found in a line.</td>
</tr>
</tbody>
</table>

? (Input is required to continue execution.)

?? (More input is required to continue execution.)

??? (Input is unintelligible.)
APPENDIX E

SPECIAL CHARACTERS

Note: Superscript "C" indicates a control character (Press ctrl and character simultaneously.)

<table>
<thead>
<tr>
<th>KEY</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt-mode</td>
<td>Deletes a line being typed. (Same as esc).</td>
</tr>
<tr>
<td>break</td>
<td>Terminates a running program, listing, or punching.</td>
</tr>
<tr>
<td>C^C</td>
<td>Terminates an input loop (C^C return); causes a jump to the END statement.</td>
</tr>
<tr>
<td>esc</td>
<td>Deletes a line being typed (same as alt-mode).</td>
</tr>
<tr>
<td>linefeed</td>
<td>Causes the teleprinter to advance one line.</td>
</tr>
<tr>
<td>N^C</td>
<td>Generates a linefeed when used in a PRINT statement.</td>
</tr>
<tr>
<td>O^C</td>
<td>Generates a return when used in a PRINT statement.</td>
</tr>
</tbody>
</table>
| return  | 1. Must follow every command or statement.  
|         | 2. Causes the teleprinter typeface to return to the first print position.  
|         | 3. TSB responds with a linefeed. |
| ^       | Backspace. Deletes as many preceding characters as ^'s are typed in. |
## OPERATORS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>100 A=B=C=Ø</td>
<td>Assignment operator; assigns a value to a variable;</td>
</tr>
<tr>
<td>✱</td>
<td>110 LET A = Ø</td>
<td>May also be used with LET.</td>
</tr>
<tr>
<td>*</td>
<td>120 PRINT X+2</td>
<td>Exponentiate (as in $X^2$).</td>
</tr>
<tr>
<td>/</td>
<td>130 LET C5 = (A*B)*N2</td>
<td>Multiply</td>
</tr>
<tr>
<td>+</td>
<td>140 PRINT T5/4</td>
<td>Divide</td>
</tr>
<tr>
<td>-</td>
<td>150 LET P = R1 +10</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td>160 X3 = R3 - P</td>
<td>Subtract</td>
</tr>
</tbody>
</table>

**NOTE:** The numeric values used in logical evaluation are: "true" = any nonzero number; "false" = Ø.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>170 IF D=E THEN 600</td>
<td>expression equals expression</td>
</tr>
<tr>
<td>≠</td>
<td>180 IF (D+E)#(2*D)THEN 710</td>
<td>expression does not equal expression</td>
</tr>
<tr>
<td>&lt;</td>
<td>190 IF X&gt;10 THEN 620</td>
<td>expression is greater than expression</td>
</tr>
<tr>
<td>&gt;</td>
<td>200 IF R8&lt;P7 THEN 640</td>
<td>expression is less than expression</td>
</tr>
<tr>
<td>≥</td>
<td>210 IF R8&gt;=P7 THEN 710</td>
<td>expression is greater than or equal to expression</td>
</tr>
<tr>
<td>≤</td>
<td>220 IF X2&lt;=10 THEN 650</td>
<td>expression is less than or equal to expression</td>
</tr>
</tbody>
</table>

**AND**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>230 IF G2 AND H5 THEN 900</td>
<td>expression1 AND expression2 must both be &quot;true&quot; for statement to be &quot;true&quot;</td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>240 IF G2 OR H5 THEN 910</td>
<td>If either expression1 OR expression2 is &quot;true&quot;, statement is &quot;true&quot;</td>
</tr>
</tbody>
</table>

**NOT**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>250 IF NOT G5 THEN 950</td>
<td>Statement is &quot;true&quot; when expression (NOT G5) is &quot;false&quot;.</td>
</tr>
</tbody>
</table>

**MAX**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>260 LET B = A2 MAX C3</td>
<td>Evaluates for the larger of the two expressions</td>
</tr>
</tbody>
</table>

**MIN**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE/MEANING/TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>270 LET B1 = A7 MIN A9</td>
<td>Evaluates for the smaller of the two expressions</td>
</tr>
<tr>
<td>Name</td>
<td>Example</td>
<td>Purpose</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATA</td>
<td>360 DATA 99,106.7, &quot;HI!&quot;,16.2</td>
<td>Specifies data; read from left to right.</td>
</tr>
<tr>
<td>DIM</td>
<td>310 DIM A(72)</td>
<td>Specifies maximum string or matrix size.</td>
</tr>
<tr>
<td>END</td>
<td>400 END</td>
<td>Terminates the program; the last statement in a program must be an END statement.</td>
</tr>
<tr>
<td>FOR...NEXT</td>
<td>350 FOR J=1 TO N STEP 3</td>
<td>Executes statements between FOR and NEXT the specified number of times (a loop), and in increments of the size indicated after STEP; STEP and STEP SIZE may be omitted.</td>
</tr>
<tr>
<td>GO TO</td>
<td>330 GO TO 900</td>
<td>Transfers control (jumps) to specified statement number.</td>
</tr>
<tr>
<td>GO TO...OF</td>
<td>412 GO TO n OF 100,10.20</td>
<td>Transfers control to the nth statement of the statements listed after &quot;OF&quot;.</td>
</tr>
<tr>
<td>GOSUB</td>
<td>420 GOSUB 800</td>
<td>Begins executing the subroutine at specified statement (see RETURN).</td>
</tr>
<tr>
<td>GOSUB...OF</td>
<td>415 GOSUB n OF 100,10,20</td>
<td>Begins executing the subroutine n of the subroutines listed after &quot;OF&quot; (See RETURN).</td>
</tr>
<tr>
<td>IF...THEN</td>
<td>340 IF A#10 THEN 350</td>
<td>Logical test of specified condition; transfers control if &quot;true&quot;.</td>
</tr>
<tr>
<td>INPUT</td>
<td>390 INPUT X$,Y2,B4</td>
<td>Allows data to be entered from teletype while a program is running.</td>
</tr>
<tr>
<td>LET</td>
<td>300 LET A=B=C=0</td>
<td>Assigns variable a value; LET is optional.</td>
</tr>
<tr>
<td>NEXT</td>
<td>355 NEXT J</td>
<td>Marks the boundary of the FOR loop.</td>
</tr>
<tr>
<td>READ</td>
<td>360 READ A,B,C</td>
<td>Reads information from DATA statement.</td>
</tr>
<tr>
<td>REM</td>
<td>320 REM--ANY TEXT**!!</td>
<td>Inserts non-executable remarks in a program.</td>
</tr>
<tr>
<td>PRINT</td>
<td>356 PRINT A,B,C$</td>
<td>Prints the specified values; 5 fields per line when commas are used as separators.</td>
</tr>
<tr>
<td></td>
<td>357 PRINT X;Y;Z$;P;Q;R(5)</td>
<td>Prints the specified values; 12 fields per line when semicolons are used as separators.</td>
</tr>
<tr>
<td></td>
<td>358 PRINT</td>
<td>Causes the teleprinter to advance one line.</td>
</tr>
<tr>
<td></td>
<td>395 PRINT#</td>
<td>See &quot;Files&quot; in this section.</td>
</tr>
</tbody>
</table>
## STATEMENTS, CONTINUED

<table>
<thead>
<tr>
<th>NAME</th>
<th>EXAMPLE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE</td>
<td>380 RESTORE</td>
<td>Permits re-reading data without re-running the program.</td>
</tr>
<tr>
<td></td>
<td>385 RESTORE ( n )</td>
<td>Permits data to be re-read, beginning in statement ( n ).</td>
</tr>
<tr>
<td>RETURN</td>
<td>850 RETURN</td>
<td>Transfers control to statement following its GOSUB.</td>
</tr>
<tr>
<td>STOP</td>
<td>410 STOP</td>
<td>Terminates the program; may be used anywhere in program.</td>
</tr>
</tbody>
</table>
NOTE: Commands are executed immediately; they do not require statement numbers.

<table>
<thead>
<tr>
<th>FULL NAME</th>
<th>EXAMPLE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>APP-PROG.1</td>
<td>Appends the named program to current program.</td>
</tr>
<tr>
<td>BYE</td>
<td>BYE</td>
<td>Log off.</td>
</tr>
<tr>
<td>CATALOG</td>
<td>CAT</td>
<td>Produces a listing of user library program names and length in two-character words.</td>
</tr>
<tr>
<td>DELETE</td>
<td>DEL-100</td>
<td>Deletes all statements after and including the specified one.</td>
</tr>
<tr>
<td></td>
<td>DEL-100,200</td>
<td>Deletes all statements between and including the specified ones.</td>
</tr>
<tr>
<td>ECHO</td>
<td>ECH-OFF</td>
<td>Permits use of a half duplex coupler; entered after logging in.</td>
</tr>
<tr>
<td></td>
<td>ECH-ON</td>
<td>Returns user to full duplex mode.</td>
</tr>
<tr>
<td>GET</td>
<td>GET-SAMPLE</td>
<td>Retrieves the specified program from the user's library and makes it the current program.</td>
</tr>
<tr>
<td>GET-$</td>
<td>GET-$PROG</td>
<td>Retrieves the named program from the system library.</td>
</tr>
<tr>
<td>HELLO-</td>
<td>HEL-D007,PDC</td>
<td>Log on. User needs I.D. code and Password.</td>
</tr>
<tr>
<td>KEY</td>
<td>KEY</td>
<td>Returns control to keyboard after TAPE inputs.</td>
</tr>
<tr>
<td>KILL</td>
<td>KIL-SAMPLE</td>
<td>Deletes the specified program from the user's library (does not modify the current program).</td>
</tr>
<tr>
<td>LENGTH</td>
<td>LEN</td>
<td>Produces a listing of the current program length in two-character words.</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>LIB</td>
<td>Produces a listing of system library program names, and size in two-character words.</td>
</tr>
</tbody>
</table>

Continued on next page.
### COMMANDS, CONTINUED

<table>
<thead>
<tr>
<th>FULL NAME</th>
<th>EXAMPLE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>LIS</td>
<td>Produces a listing of current program.</td>
</tr>
<tr>
<td></td>
<td>LIS-15Ø</td>
<td>Produces a listing, starting at specified statement.</td>
</tr>
<tr>
<td>NAME</td>
<td>NAM-SAMPLE</td>
<td>Assigns specified name to the current program; name may be 1 to 6 characters in length and must include only printing characters.</td>
</tr>
<tr>
<td>PUNCH</td>
<td>PUN</td>
<td>Punches current program to paper tape.</td>
</tr>
<tr>
<td></td>
<td>PUN-5Ø</td>
<td>Punches program to paper tape, beginning at specified statement.</td>
</tr>
<tr>
<td>RENUMBER</td>
<td>REN</td>
<td>Renumbers program from 10 in multiples of 10.</td>
</tr>
<tr>
<td></td>
<td>REN-5Ø</td>
<td>Renumbers program from specified statement number in multiples of 10.</td>
</tr>
<tr>
<td></td>
<td>REN-6Ø,y</td>
<td>Renumbers program from specified statement number in multiples of y.</td>
</tr>
<tr>
<td>RUN</td>
<td>RUN</td>
<td>Starts program execution.</td>
</tr>
<tr>
<td></td>
<td>RUN-5Ø</td>
<td>Starts program execution at specified statement.</td>
</tr>
<tr>
<td>SAVE</td>
<td>SAV</td>
<td>Saves the current program in user's library.</td>
</tr>
<tr>
<td>SCRATCH</td>
<td>SCR</td>
<td>Erases current program (but not program name.)</td>
</tr>
<tr>
<td>TAPE</td>
<td>TAP</td>
<td>Informs computer that following input is from paper tape.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIM</td>
<td>Produces a listing of terminal and account time.</td>
</tr>
</tbody>
</table>
## FUNCTIONS

**NOTE:** PRINT is used for examples only; other statement types may be used.

<table>
<thead>
<tr>
<th>FULL NAME</th>
<th>EXAMPLE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF FN</td>
<td>300 DEF FNA (X)=(M*X)+B</td>
<td>Allows the programmer to define functions; the function label (A) must be a letter from A to Z; the argument (X) must be mentioned in the function definition.</td>
</tr>
<tr>
<td>ABS (X)</td>
<td>310 PRINT ABS (X)</td>
<td>Gives the absolute value of the expression (X).</td>
</tr>
<tr>
<td>EXP (X)</td>
<td>320 PRINT EXP (X)</td>
<td>Gives the constant e raised to the power of the expression value (X); in this example, e^X.</td>
</tr>
<tr>
<td>INT (X)</td>
<td>330 PRINT INT (X)</td>
<td>Gives the largest integer ≤ the expression (X).</td>
</tr>
<tr>
<td>LOG (X)</td>
<td>340 PRINT LOG (X)</td>
<td>Gives the natural logarithm of an expression; expression must have a positive value.</td>
</tr>
<tr>
<td>RND (X)</td>
<td>350 PRINT RND (X)</td>
<td>Generates a random number between 0 and 1; the expression (X) is a dummy argument.</td>
</tr>
<tr>
<td>SQR (X)</td>
<td>360 PRINT SQR (X)</td>
<td>Gives the square root of the expression (X); expression must have a positive value.</td>
</tr>
<tr>
<td>SIN (X)</td>
<td>370 PRINT SIN (X)</td>
<td>Gives the sine of the expression (X); X is real and in radians.</td>
</tr>
<tr>
<td>COS (X)</td>
<td>380 PRINT COS (X)</td>
<td>Gives the cosine of the expression (X); X is real and in radians.</td>
</tr>
<tr>
<td>TAN (X)</td>
<td>390 PRINT TAN (X)</td>
<td>Gives the tangent of the expression (X); X is real and in radians.</td>
</tr>
<tr>
<td>ATN (X)</td>
<td>400 PRINT ATN (X)</td>
<td>Gives the arctangent of the expression (X); is real and in radians.</td>
</tr>
<tr>
<td>LEN (X)</td>
<td>410 PRINT LEN (A$)</td>
<td>Gives the current length of a string (A$), i.e., number of characters.</td>
</tr>
<tr>
<td>TAB (X)</td>
<td>420 PRINT TAB (X);A</td>
<td>Tabs to the specified position (X), then prints the specified value (A).</td>
</tr>
<tr>
<td>TYP (X)</td>
<td>430 PRINT TYP (X);</td>
<td>If argument (X) is negative, gives the type of data in a file as: 1=number; 2=string; 3=&quot;end of file&quot;; 4=&quot;end of record&quot;; or if argument (X) is positive, gives the type of data in a file as: 1=number; 2=string; 3=&quot;end of file&quot;. (For sequential access to files - skips over &quot;end of records&quot;).</td>
</tr>
<tr>
<td>SGN (X)</td>
<td>440 PRINT SGN (X)</td>
<td>Gives: +1 if X&gt;0, 0 if X=0, -1 if X&lt;0.</td>
</tr>
</tbody>
</table>

 powerhouse
STRINGS

NOTES: 1. A string is 1 to 72 teleprinter characters enclosed in quotes; it may be assigned to a string variable (an A to Z letter followed by a $).

2. Each string variable used in a program must be dimensioned (with a DIM statement), if it has a length of more than one character.

3. Substrings are described by subscripted string variables. For example, if A$ = "ABCDEF", A$ (2,2) = B, and A$ (1,4) = "ABCD".

3. The LEN function returns the current string length, for example:
100 PRINT LEN (A$).

<table>
<thead>
<tr>
<th>FULL NAME</th>
<th>EXAMPLE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM</td>
<td>10 DIM A$ (27)</td>
<td>Declares string length in characters.</td>
</tr>
<tr>
<td>LET</td>
<td>20 LET A$ = &quot;**TEXT 1&quot;</td>
<td>Assigns the character string in quotes to a string variable.</td>
</tr>
<tr>
<td>LEN</td>
<td>30 PRINT LEN (B$)</td>
<td>Gives the current length of the specified string.</td>
</tr>
<tr>
<td>=</td>
<td>105 IF A$=C$ THEN 600</td>
<td>String operators. They allow comparison of strings, and substrings, and transfer to a specified statement. Comparison is made in ASCII codes, character by character, left to right until a difference is found. If the strings are of unequal length, the shorter string is considered smaller if it is identical to the initial substring of the longer.</td>
</tr>
<tr>
<td>#</td>
<td>110 IF B$#X$ THEN 650</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>115 IF N$(2,2)&gt;B$(3,3) THEN 10</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>120 IF N$&lt;B$ THEN 999</td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td>125 IF P$ (5,8)=Y$(4,7)THEN 10</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td>130 IF X$&lt;=Z$ THEN 999</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>205 INPUT N$</td>
<td>Accepts the appropriate number of characters (followed by a return). The characters need not be in quotation marks if only one string is input.</td>
</tr>
<tr>
<td>INPUT</td>
<td>210 INPUT N$,X$,Y$</td>
<td>Inputs the specified strings; input must be in quotes and separated by commas.</td>
</tr>
<tr>
<td>READ</td>
<td>215 READ P$</td>
<td>Reads a string from a DATA statement; each string read must be enclosed in quotes.</td>
</tr>
<tr>
<td>READ#</td>
<td>220 READ#5; A$,B$</td>
<td>Reads strings from the specified file.</td>
</tr>
<tr>
<td>PRINT#</td>
<td>310 PRINT#2; A$,C$</td>
<td>Prints strings on a file.</td>
</tr>
</tbody>
</table>
### MATRICES, CONTINUED

**NOTES:**
1. Absolute maximum matrix size is 2500 elements.
2. Matrix variables must be a single letter from A to Z.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM</td>
<td>10 DIM A (10, 20)</td>
<td>Allocates space for a matrix of the specified dimensions.</td>
</tr>
<tr>
<td>MAT IDN</td>
<td>15 MAT X = IDN (m,n)</td>
<td>Establishes an identity matrix (with all ones down the diagonal). A new working size (m,n) may be specified;</td>
</tr>
<tr>
<td>MAT ZER</td>
<td>20 MAT B = ZER</td>
<td>Sets all elements of the specified matrix equal to 0.</td>
</tr>
<tr>
<td></td>
<td>25 MAT D = ZER (m,n)</td>
<td>A new working size (m,n) may be specified after ZER.</td>
</tr>
<tr>
<td>MAT CON</td>
<td>30 MAT C = CON</td>
<td>Sets all elements of the specified matrix equal to 1</td>
</tr>
<tr>
<td></td>
<td>35 MAT E = CON (m,n)</td>
<td>A new working size (m,n) may be specified after CON.</td>
</tr>
<tr>
<td>INPUT</td>
<td>40 INPUT A(5,5)</td>
<td>Allows input from the teleprinter of a specified matrix element.</td>
</tr>
<tr>
<td></td>
<td>45 MAT INPUT A(5,5)</td>
<td>Allows input of a matrix from the teleprinter; a new working size may be specified.</td>
</tr>
<tr>
<td>MAT PRINT</td>
<td>50 MAT PRINT A</td>
<td>Prints the specified matrix on the teleprinter.</td>
</tr>
<tr>
<td></td>
<td>55 PRINT A(X,Y)</td>
<td>Prints the specified element of a matrix on the teleprinter; element specifications X and Y may be any expression.</td>
</tr>
<tr>
<td></td>
<td>60 PRINT #2; A(1,5)</td>
<td>Prints matrix element on the specified file number.</td>
</tr>
<tr>
<td></td>
<td>65 MAT PRINT #2,3;A</td>
<td>Prints matrix on a specified file and record.</td>
</tr>
<tr>
<td>MAT READ</td>
<td>70 MAT READ A</td>
<td>Reads matrix from DATA statements.</td>
</tr>
<tr>
<td></td>
<td>75 MAT READ A(5,5)</td>
<td>Reads matrix of specified size from DATA statements.</td>
</tr>
<tr>
<td></td>
<td>80 READ A(X,Y)</td>
<td>Reads the specified matrix element from a DATA statement.</td>
</tr>
<tr>
<td></td>
<td>85 MAT READ #3; A</td>
<td>Reads matrix from the specified file.</td>
</tr>
<tr>
<td></td>
<td>90 MAT READ #3,5;A</td>
<td>Reads matrix from the specified record of a file.</td>
</tr>
</tbody>
</table>

Continued on the next page.
## MATRICES CONTINUED

<table>
<thead>
<tr>
<th>NAME</th>
<th>SAMPLE STATEMENT</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAT +</td>
<td>100 MAT C = A + B</td>
<td>Matrix addition; A and B must be the same size.</td>
</tr>
<tr>
<td>MAT -</td>
<td>110 MAT C = A − B</td>
<td>Matrix subtraction; A, B, and C must be the same size.</td>
</tr>
<tr>
<td>MAT*</td>
<td>120 MAT C = A ∗ B</td>
<td>Matrix multiplication; No. columns in A must equal No. rows in B.</td>
</tr>
<tr>
<td>MAT =</td>
<td>130 MAT A = B</td>
<td>Establishes equality of two matrices; assigns values of B to A.</td>
</tr>
<tr>
<td>MAT TRN</td>
<td>140 MAT B = TRN (A)</td>
<td>Transposes an m by n matrix to an n by m matrix.</td>
</tr>
<tr>
<td>MAT INV</td>
<td>150 MAT C = INV (B)</td>
<td>Inverts a square matrix into a square matrix of the same size; matrix may be inverted into itself.</td>
</tr>
</tbody>
</table>
NOTES: 1. STRUCTURE OF A FILE: 1 to 128 64-word records. Maximum size varies with systems; consult system operator. Files have logical "end of record" markers and "end of file" markers. Attempting to read an "end of file" or "end of record" marker will terminate the program unless an IF END# statement is used.

2. File names may be 1 to 6 printing characters.

3. The formula for allocating file space for strings is:
   
   \[ \text{number of characters in the string} \times \left( \frac{1 + \text{number of characters in the string}}{2} \right) \]

   This formula gives the required storage space in 2-character words.

---

**FULL NAME** | **EXAMPLE (Abbreviation)** | **PURPOSE**
---|---|---
OPEN- | OPE-MYFILE,85 | Opens file; assigns specified name and number of 64-word records.
KILL- | KIL-MYFILE | Deletes specified file.
FILES | 10 FILES FILE#1, SECOND,... | Tells the system which files to use (maximum of 8); used only once in a program. Files are assigned reference numbers (1 to 8) sequentially.
PRINT# | 120 PRINT# 1; A,B,C | Prints the specified values (A,B,C) on a specified file number (file reference numbers are assigned consecutively from the FILES statement).
 | 130 PRINT# X,Y; A,B,C | Prints the specified information on file number (X), record number (Y); X and Y are rounded to integer values.
 | 140 PRINT# 2; A,B,END | Prints value on specified file; inserts an "end of file" marker immediately after the printed value.
 | 160 PRINT# 3,5 | Sets the file pointer to the beginning of the specified file (3), and the specified record (5); erases the specified record.
READ# | 170 READ# 1; A,B2,C | Reads the specified values from a specified file (numbered consecutively by the system, from those given in the FILES statement).

Continued on the next page.
## FILES , CONTINUED

<table>
<thead>
<tr>
<th>FULL NAME</th>
<th>EXAMPLE (Abbreviation)</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ#</td>
<td>180 READ# 2,3; A,B</td>
<td>Reads specified values from a specified file (2) and a specified record in that file (3).</td>
</tr>
<tr>
<td></td>
<td>185 READ# 2,5</td>
<td>Resets the file pointer to the specified file record without erasing the record.</td>
</tr>
<tr>
<td>IF END#</td>
<td>190 IF END# 1 THEN 800</td>
<td>Transfers control to a specified statement number if the end of the specified file is encountered.</td>
</tr>
</tbody>
</table>
This index is produced with the help of the two programs listed below. The first accepts two strings (topic and page references), and writes them on a file. The second program reads the file, sorts the topics alphabetically, and prints them on the teleprinter.

```
10 DIM A$[40],B$[40]
20 FILES INDX2
70 PRINT "START NEW INPUT."
80 PRINT "T";
90 INPUT A$
100 PRINT "P";
110 INPUT B$
130 PRINT #1;A$,B$
140 GOTO 80
150 END

LIS
OK

1 IF END #1 THEN 200
5 DIM A$[40],B$[40],C$[40],L$[40],M$[40]
10 FILES INDX2, INDX3
20 LET C$=""
25 LET L$="ZZZZZZ"
30 READ #1;A$,B$
40 IF A$ < = C$ THEN 30
50 IF A$ > = L$ THEN 30
60 LET L$=A$
65 LET M$=B$
67 LET R=0
70 GOTO 30
200 IF R THEN 400
205 PRINT L$; "..."M$
210 PRINT #2;L$,M$
220 LET C$=L$
225 LET L$="ZZZZ"
230 LET R=1
235 READ #1,1
240 GOTO 30
400 END
```

Continued on the next page.
<table>
<thead>
<tr>
<th>INDEX, CONTINUED</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ...2-6</td>
</tr>
<tr>
<td>+ ...2-6</td>
</tr>
<tr>
<td>- ...2-5, 2-7</td>
</tr>
<tr>
<td>/ ...2-6</td>
</tr>
<tr>
<td>&lt; ...2-7</td>
</tr>
<tr>
<td>&lt;= ...2-7</td>
</tr>
<tr>
<td>&gt; ...2-7</td>
</tr>
<tr>
<td>&gt;= ...2-7</td>
</tr>
<tr>
<td>ABS ...3-22</td>
</tr>
<tr>
<td>ABS FUNCTION ...3-22</td>
</tr>
<tr>
<td>ACCESS, FILE ...6-11</td>
</tr>
<tr>
<td>ACCESS STRING ...6-6</td>
</tr>
<tr>
<td>ADDITION ...2-6</td>
</tr>
<tr>
<td>ADVANCED BASIC ...3-1, FF</td>
</tr>
<tr>
<td>ALLOCATION MEMORY ...8-8</td>
</tr>
<tr>
<td>ALT-MODE ...1-13</td>
</tr>
<tr>
<td>AND ...2-9</td>
</tr>
<tr>
<td>AND OPERATOR ...2-9</td>
</tr>
<tr>
<td>APPEND ...3-12</td>
</tr>
<tr>
<td>ARITHMETIC EVALUATION ...2-4</td>
</tr>
<tr>
<td>ARITHMETIC OPERATOR ...2-6</td>
</tr>
<tr>
<td>ASCII CODES ...8-7</td>
</tr>
<tr>
<td>ASSIGNMENT OPERATOR ...2-5</td>
</tr>
<tr>
<td>ATN FUNCTION ...3-23</td>
</tr>
<tr>
<td>BACKSPACE ...1-12</td>
</tr>
<tr>
<td>BREAK ...1-23,2-41</td>
</tr>
<tr>
<td>BREAK KEY ...1-23, 2-41</td>
</tr>
<tr>
<td>BYE ...2-35</td>
</tr>
<tr>
<td>CATALOG ...3-15</td>
</tr>
<tr>
<td>COMBINING LOGICAL OPERATORS ...7-6</td>
</tr>
<tr>
<td>COMMANDS ...2-34</td>
</tr>
<tr>
<td>CONNECTION TO THE COMPUTER ...1-16</td>
</tr>
<tr>
<td>CONTROL C ...1-17</td>
</tr>
<tr>
<td>CONTROL N ...6-7, E-1</td>
</tr>
<tr>
<td>CONTROL O ...6-7, E-1</td>
</tr>
<tr>
<td>CONVENTIONS IN THIS BOOK ...IV</td>
</tr>
<tr>
<td>COPYING A MATRIX ...5-15</td>
</tr>
<tr>
<td>CORRECTING MISTAKES ...1-13</td>
</tr>
<tr>
<td>COS ...3-23</td>
</tr>
<tr>
<td>COSINE FUNCTION ...3-23</td>
</tr>
<tr>
<td>DATA ...2-21, 6-10</td>
</tr>
<tr>
<td>DATA ON PAPER TAPE ...B-1</td>
</tr>
<tr>
<td>DATA TYPE ...3-2</td>
</tr>
<tr>
<td>DEF FN ...3-21</td>
</tr>
<tr>
<td>DEFINING FUNCTIONS ...3-21</td>
</tr>
<tr>
<td>DELETE ...3-13</td>
</tr>
<tr>
<td>DELETING A LINE ...1-13</td>
</tr>
<tr>
<td>DELETING PROGRAMS ...3-7</td>
</tr>
<tr>
<td>DIAGNOSTIC MESSAGES ...1-19, D-1</td>
</tr>
<tr>
<td>DIM ...5-2, 6-4</td>
</tr>
<tr>
<td>DIMENSIONING A MATRIX ...5-2</td>
</tr>
<tr>
<td>DIVISION ...2-6</td>
</tr>
<tr>
<td>ECHO ...2-36</td>
</tr>
<tr>
<td>END OF FILE ...4-14</td>
</tr>
<tr>
<td>END OF STATEMENT ...1-7, 2-28</td>
</tr>
<tr>
<td>E NOTATION ...2-2</td>
</tr>
<tr>
<td>ENTERING DATA FROM TAPE ...B-1</td>
</tr>
<tr>
<td>ESC ...1-13</td>
</tr>
<tr>
<td>ESCAPE KEY ...CONTROL CHARACTERS</td>
</tr>
<tr>
<td>EXP ...3-22</td>
</tr>
<tr>
<td>EXP FUNCTION ...3-22</td>
</tr>
<tr>
<td>EXPONENTIAL ...2-6</td>
</tr>
<tr>
<td>EXPRESSION ...2-4</td>
</tr>
<tr>
<td>FILE ACCESS ...5-19, 5-20</td>
</tr>
<tr>
<td>FILE STRUCTURE ...4-10</td>
</tr>
<tr>
<td>FILES ...4-1</td>
</tr>
<tr>
<td>FILES STATEMENT ...4-6</td>
</tr>
<tr>
<td>FOR ...2-18</td>
</tr>
<tr>
<td>FOR...NEXT WITH STEP ...3-20</td>
</tr>
<tr>
<td>FORMAL LANGUAGE ...1-2</td>
</tr>
<tr>
<td>FORMAT OF STATEMENTS ...1-8</td>
</tr>
<tr>
<td>FUNCTIONS ...3-16</td>
</tr>
<tr>
<td>GET ...3-10</td>
</tr>
<tr>
<td>GET-$ ...3-10</td>
</tr>
<tr>
<td>GO TO ...2-16</td>
</tr>
<tr>
<td>GO TO...0F ...2-16</td>
</tr>
<tr>
<td>GOSUB ...3-17</td>
</tr>
<tr>
<td>GOSUB...OF ...3-19</td>
</tr>
<tr>
<td>HELLO- ...2-34</td>
</tr>
<tr>
<td>HIERARCHY OF OPERATORS ...2-12</td>
</tr>
<tr>
<td>HOW TO USE THIS BOOK ...VII</td>
</tr>
<tr>
<td>ID CODE ...1-17</td>
</tr>
<tr>
<td>IDENTITY MATRIX ...5-16</td>
</tr>
<tr>
<td>IF END# STATEMENT ...4-9</td>
</tr>
<tr>
<td>IF...THEN ...2-17</td>
</tr>
<tr>
<td>INPUT ...2-24, 5-5, 6-6</td>
</tr>
<tr>
<td>INPUT OF MATRIX ELEMENTS ...5-5</td>
</tr>
<tr>
<td>INSTRUCTIONS ...1-5</td>
</tr>
<tr>
<td>INT ...3-22</td>
</tr>
<tr>
<td>INT FUNCTION ...3-22</td>
</tr>
<tr>
<td>INVERTING A MATRIX ...5-18</td>
</tr>
<tr>
<td>KEY ...2-44</td>
</tr>
<tr>
<td>KILL ...3-11</td>
</tr>
<tr>
<td>KILL- ...4-5</td>
</tr>
<tr>
<td>LEN COMMAND ...3-7</td>
</tr>
<tr>
<td>LEN FUNCTION ...3-26</td>
</tr>
<tr>
<td>LET ...2-14, 6-5</td>
</tr>
<tr>
<td>LIBRARY ...3-14</td>
</tr>
<tr>
<td>LINE NUMBER ...1-4</td>
</tr>
<tr>
<td>LINEFEED ...1-17, 1-20</td>
</tr>
<tr>
<td>LIST ...1-14, 2-38</td>
</tr>
<tr>
<td>LISTING A FILE ...C-1</td>
</tr>
<tr>
<td>LISTING A PROGRAM ...1-14</td>
</tr>
<tr>
<td>LOG ...3-22</td>
</tr>
<tr>
<td>LOG FUNCTION ...3-22</td>
</tr>
<tr>
<td>LOG IN AND LOG OUT ...1-18</td>
</tr>
<tr>
<td>LOGICAL OPERATIONS ...7-2</td>
</tr>
<tr>
<td>MAT CON ...5-4</td>
</tr>
<tr>
<td>MAT PRINT ...5-8</td>
</tr>
<tr>
<td>MAT READ ...5-10</td>
</tr>
</tbody>
</table>
INDEX, CONTINUED

MAT ZER ... 5-3
MATHEMATICAL FUNCTIONS ... 3-22
MATRICES ... 5-1
MATRIX ADDITION ... 5-11
MATRIX EQUALITY ... 5-15
MATRIX MULTIPLICATION ... 5-13
MATRIX SUBTRACTION ... 5-2
MATRIX TRANSPOSITION ... 5-17
MAX ... 2-8
MEMORY ALLOCATION ... 8-8
MIN ... 2-8
MIN-MAX OPERATORS ... 2-8
MISTAKES AND CORRECTIONS ... 1-19
MISTAKES DURING LOG IN ... 1-19
MULTIBRANCH GOSUB ... 3-18
MULTIBRANCH GOTO ... 2-1
MULTIPLICATION ... 2-6
NAME ... 3-8
NATURAL LANGUAGE ... 1-2
NESTING GOSUBS ... 3-19
NEXT ... 2-18, 3-20
NOT ... 2-11
NOT OPERATOR ... 2-11
NUMBER ... 2-2
NUMBER, LINE ... 1-4
NUMBER, STATEMENT ... 1-4
OPEN ... 4-4
OPENING AND CLOSING FILES ... 4-4, FF
OPERANDS ... 1-6
OPERATIONS, MAT ... 5-11, FF
OPERATORS, BOOLEAN ... 7-4
OPERATORS, RELATIONAL ... 2-5, FF, 7-4
OR ... 2-10
OR OPERATOR ... 2-10
ORDER OF PRECEDENCE (OPERS.) ... 2-12
PAGE FORMAT ... V
PASSWORD ... 1-17
POINTER, RESET ... 4-16, 4-18
PREFACE ... III
PREPARE TAPE OFF-LINE ... A-1
PRINT, MAT ... 5-8, 5-19
PRINT TO FILE ... 4-7
PRINT# ... END ... 4-14
PROGRAMMER-DEFINED FUNCTION ... 3-21
PUNCH ... 2-42
PUNCHING TAPE OFFLINE ... A-1
QUICK REFERENCE SECTION ... E-1, FF
RAISE TO A POWER ... 2-6
RANDOM FILE ACCESS ... 4-14, FF
READ ... 2-21, 4-8, 4-17, 6-8, 6-12

READ FROM FILE ... 4-8
READ, MAT ... 5-10, 5-20
READING, WRITING MATRICES ... 5-5, FF
RELATIONAL OPERATORS ... 2-7, 7-2
REM ... 2-15
RENUMBER ... 2-40
RESTORE ... 2-21
RETURN ... 1-11
RND ... 3-22
RUN ... 2-37
SAVE ... 3-9
SCALAR MULTIPLICATION ... 5-14
SCRATCH ... 2-39
SERIAL FILES ... 4-6, FF
SGN FUNCTION ... 3-24
SIMPLE VARIABLE ... 2-3
SIN ... 3-23
SPECIAL CHARACTERS ... E-1
SQR ... 3-22
STANDARD FUNCTIONS ... 3-22, FF
STATEMENTS ... 1-3, 2-13
STATEMENT TYPES ... 1-5
STOP ... 2-28
STORING, DELETING PROGRAMS ... 3-6
STRING ASSIGNMENT ... 6-5
STRING DIM ... 6-4
STRING EVALUATION (ASCII) ... 8-7
STRING IF ... 6-9
STRING INPUT ... 6-6
STRING LET ... 6-5
STRING PRINT TO FILE ... 6-12
STRING READ ... 6-8
STRING READ FROM FILE ... 6-11
STRINGS ... 6-1, 6-2
STRUCTURE FILES ... 4-10, FF
SUBROUTINES AND FUNCTIONS ... 3-16, FF
SUBSTRING ... 6-3
SUBTRACTION ... 2-6
SYNTAX REQUIREMENTS OF TSB ... 8-2
TAB ... 3-24
TAN FUNCTION ... 3-23
TAPE ... 2-43
TIME ... 2-45
TIME SHARING ... 1-1
TRANSPONING A MATRIX ... 5-17
TRIGONOMETRIC FUNCTIONS ... 3-23
TYP FUNCTION ... 3-26
UTILITY COMMANDS ... 3-12, FF
VARIABLE, SIMPLE ... 2-3
X-ON, X-OFF ... B1