

HEWLETT  PACKARD

A Guide To Time Shared **BASIC**

**HP Computer Museum**  
**[www.hpmuseum.net](http://www.hpmuseum.net)**

**For research and education purposes only.**



A GUIDE TO TIME SHARED BASIC  
For Reference and Self-Instruction

HEWLETT  PACKARD

Software Publications  
Cupertino, California  
95014

Part Number 02000-90002

August, 1969

Copyright © 1969 by  
HEWLETT-PACKARD COMPANY  
Printed in the United States of America

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

SAMPLE	EXPLANATION
PLEASE LOG IN	Black, all capitals in examples indicates computer-output information.
And then...	Mixed upper and lower case black is used for regular text.
2Ø PRINT X,Y	Red, all capitals indicates a statement or command typed by the programmer.
<u>line number</u> PRINT X,Y	Black lower case italics indicates a generalized form. Red underlining indicates an essential part of a general form; each underlined item is a separate, essential element.
<u>return</u> <u>linefeed</u> <u>esc</u> <u>ctrl</u> <u>alt-mode</u> <u>break</u>	Represents the teleprinter keys: Return, Linefeed, Escape, Control, Alt-Mode, and Break.
Note: <i>Both X and ...</i>	Mixed upper and lower case italics is used for notes.
LISTING A PROGRAM	Oversize black is used for page headings.

Please examine the sample on the next page.

## PAGE FORMAT

The page format is as uniform as possible. This sample shows how positioning and typeface relate to content.

## SECTION TITLE

## FEATURE EXPLAINED ON THIS PAGE

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 in Section IX
3. The index tabs to locate the appropriate section.

# CONTENTS

Preface.....	iii
Conventions Used in this Text.....	iv
Page Format.....	v
How to Use this Book.....	vii

## I, AN INTRODUCTION TO TIME SHARED BASIC

How to Read this Section.....	1-1
What is Time Sharing.....	1-1
Communicating with a Computer.....	1-2
<b>EXAMPLES OF BASIC STATEMENTS</b>	
Statement.....	1-3
Line Number.....	1-3
Statement Type.....	1-3
Operands.....	1-3
Program.....	1-3
<b>BEFORE GOING ON-LINE</b>	
Mistakes and corrections.....	1-5
Deleting or changing a line.....	1-6
Listing a program.....	1-7
Connection to the computer.....	1-8
Checking the connection.....	1-9
I.D. Code and Password.....	1-9
Log In and Log Out.....	1-10
Mistakes During Log In.....	1-11
Entering the Sample Program.....	1-12
How to Obtain A Diagnostic Message.....	1-13
Running the Sample Program.....	1-14
Stopping a Program: The <i>break</i> Key.....	1-15
How the Program Works.....	1-16

## II, THE ESSENTIALS OF BASIC

How to Read this Section.....	2-1
<b>VOCABULARY</b>	
Number.....	2-2
"E" notation.....	2-2
Simple Variable.....	2-3
Expression.....	2-4
Arithmetic evaluation.....	2-4
<b>OPERATORS</b>	
Assignment.....	2-5
Arithmetic.....	2-6
Relational.....	2-7
Min-max.....	2-8
AND.....	2-9
OR.....	2-10
NOT.....	2-11
Order of Precedence.....	2-12

Continued on next page.

STATEMENTS.....	2-13
LET.....	2-14
REM.....	2-15
GO TO.....	2-16
IF...THEN .....	2-17
FOR...NEXT .....	2-18
NESTING FOR...NEXT LOOPS .....	2-20
READ, DATA, RESTORE.....	2-21
INPUT.....	2-24
PRINT.....	2-26
END, STOP.....	2-28
SAMPLE PROGRAM.....	2-29
COMMANDS .....	2-33
HELLO.....	2-34
BYE.....	2-35
ECHO.....	2-36
RUN.....	2-37
LIST.....	2-38
SCRATCH.....	2-39
RENUMBER.....	2-40
<u>BREAK</u> .....	2-41
PUNCH.....	2-42
TAPE.....	2-43
KEY.....	2-44
TIME.....	2-45

### III. ADVANCED BASIC

How to Read this Section.....	3-1
VOCABULARY.....	3-2
STORING AND DELETING PROGRAMS.....	3-6
LEN.....	3-7
NAME.....	3-8
SAVE.....	3-9
GET.....	3-10
KILL.....	3-11
UTILITY COMMANDS .....	3-12
APPEND.....	3-12
DELETE.....	3-13
LIBRARY.....	3-14
CATALOG.....	3-15
SUBROUTINES AND FUNCTIONS.....	3-16
GOSUB...RETURN .....	3-17
MULTIBRANCH GOSUB'S.....	3-18
NESTING GOSUB'S.....	3-19
FOR...NEXT with STEP .....	3-20
DEF FN.....	3-21
STANDARD FUNCTIONS .....	3-22
GENERAL MATHEMATICAL.....	3-22
TRIGONOMETRIC.....	3-23
TAB and SGN .....	3-24
TYP .....	3-25
LEN .....	3-26

IV. FILES	
VOCABULARY.....	4-2
OPENING AND CLOSING FILES	
OPEN- .....	4-4
KILL- .....	4-5
SERIAL FILES	
FILES.....	4-6
PRINT #.....	4-7
READ #.....	4-8
IF END #...THEN .....	4-9
STRUCTURE OF A FILE.....	4-10
SINGLE FILE.....	4-11
SERIAL FILES.....	4-12
FILE STRUCTURE-SUMMARY.....	4-13
RANDOM FILE ACCESS	
PRINT #..., END .....	4-14
PRINT #...,... .....	4-15
PRINT # to Reset a Pointer.....	4-16
READ #.....	4-17
READ # to Reset a Pointer.....	4-18

---

## V. MATRICES

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

---

## VI. STRINGS

VOCABULARY.....	6-2
INITIALIZING	
The String DIM Statement.....	6-4
The String ASSIGNMENT Statement.....	6-5
STRING ACCESS	
INPUT.....	6-6
PRINT...\$ .....	6-7
READ...\$ .....	6-8

String IF.....	6-9
The LEN FUNCTION.....	6-10
FILE ACCESS	
DATA.....	6-11
PRINT # .....	6-12
READ # .....	6-13

VII. LOGICAL OPERATIONS

RELATIONAL OPERATORS.....	7-2
BOOLEAN OPERATORS.....	7-4
Some Examples.....	7-6

VIII. FOR THE PROFESSIONAL

Syntax Requirements of TSB.....	8-2
String Evaluation By ASCII Codes.....	8-7
Memory Allocation by a TSB User.....	8-8
General Information.....	8-9
Comments.....	8-9

APPENDICES

A. Preparing a Paper Tape Offline.....	A-1
B. The X-ON, X-OFF Feature.....	B-1
C. Sample Programs	
Listing a File.....	C-1
Interest Rates.....	C-2
An Electronic Calendar.....	C-4
H-P Football.....	C-8
D. Diagnostic Messages.....	D-1
E. An Instant Guide to Time Shared BASIC	
Special Characters.....	E-1
Operators.....	E-2
Statements.....	E-3
Commands.....	E-5
Functions.....	E-7
Strings.....	E-8
Matrices.....	E-9
Files.....	E-11
F. Index.....	F-1



# SECTION I

## AN INTRODUCTION TO TIME SHARED BASIC

### HOW TO READ THIS SECTION

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 we use to communicate with people. To communicate with the computer we must 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 next page show how to interpret these rules. Notice how the statements are written. What they do will be explained later.



## EXAMPLES OF BASIC STATEMENTS

THIS IS A BASIC STATEMENT:

10 INPUT A,B,C,D,E

*Remember:*

- a) *It is contained in one line with a maximum of 72 characters, one teleprinter line. Blank spaces count as characters.*
- b) *A statement may also be called a line.*

EACH BASIC STATEMENT BEGINS WITH A NUMBER

(in this example, 20):

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

*Remember:*

- a) *The number is chosen by the user.*
- b) *It is an integer from 1 to 9999 inclusive.*
- c) *Each line has a different number.*
- d) *The computer puts the statements in numerical order, (by line number); it is best to leave an interval of five or ten when choosing line numbers (you may wish to insert additional statements between the ones you have already numbered).*
- e) *Statements are acted upon in the sequence given by their statement numbers.*

THEN AN INSTRUCTION OR STATEMENT TYPE IS GIVEN

(in this example, PRINT):

30 PRINT S

NEXT, IF THE INSTRUCTION REQUIRES FURTHER  
DETAILS, OPERANDS (NUMERIC DETAILS) ARE  
SUPPLIED (in this example, 10; in statement 30,S):

40 GO TO 10

*Remember:*

- a) *The operands specify what the instruction acts upon, for example, what is INPUT or PRINTED.*

The sequence of BASIC statements in these four frames is called a program. Note that the last line in the program is an END statement. This informs the computer that the program is finished.

50 END

*Remember:*

- a) *The last statement in a BASIC program must be an END statement.*

Continued on next page.

## EXAMPLES OF BASIC STATEMENTS, CONTINUED

One point not obvious from inspection of the sample program is the free format feature of BASIC. Blank spaces have no effect (as long as the statement is contained in 72 characters). For example, these three lines are equivalent:

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

When possible, leave a space between words and numbers for better legibility.

---

## BEFORE CONTINUING

Be sure you are familiar with these terms, introduced in the previous section:

instruction (or statement type)  
line number  
operand  
program  
statement

## BEFORE GOING ON-LINE

The following four frames explain how to correct mistakes and list what has been entered. Since you will probably have to make several corrections in your first attempts to use the teleprinter on-line (connected to the computer), you should be familiar with these features.

Note that the return key must be pressed after each command or statement.

## MISTAKES AND CORRECTIONS

The character ← (shift 0) acts as a backspace, deleting the immediately preceding character. Typing multiple ←'s deletes as many preceding characters as ←'s.

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

## DELETING OR CHANGING A LINE

To delete the statement being typed, press the esc or alt-mode key. This causes a \ to be printed, and deletes the entire line.

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

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

Pressing esc

deletes the line being typed: 20 LET S = esc

*NOTE: The computer responds with a \ when esc is typed.*

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

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

type: 5 return

Or, to change statement 5 in the above sequence, type: 5 LET S = 5 return

The old statement is replaced by the new one.

Typing an esc (or alt-mode) before a return aborts the replacement line.

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

will have 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 produces a listing of all lines accepted by the computer.

*NOTE: The program has already been entered.*

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

linefeed indicates that the listing is complete.

LIST return

linefeed

linefeed

linefeed

1Ø INPUT A,B,C,D,E

2Ø LET S = (A+B+C+D+E)/5

3Ø PRINT S

4Ø GO TO 1Ø

5Ø END

linefeed

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

A list of the same sample program will produce these lines:

LIST-3Ø return

linefeed

linefeed

linefeed

3Ø PRINT S

4Ø GO TO 1Ø

5Ø END

linefeed

## CONNECTION TO THE COMPUTER

To enter a program into the computer you must 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 a 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.



## SAMPLE LOG IN AND LOG OUT

To LOG IN, type: HELLO-H200,SE<sup>C</sup>C<sup>C</sup>R<sup>C</sup>E<sup>C</sup>T 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.

linefeed  
linefeed  
linefeed  
READY  
linefeed

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 instance, if a user forgets the hyphen while entering the HELLO command:

```
HELLO H200,SECCCRCECT 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,SECCCRCET return
```

*Note: The final E in the password should have been E<sup>C</sup>.*

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

Sample Prog.

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,SE<sup>C</sup>C<sup>R</sup>C<sup>E</sup>T return  
SYSTEM 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. You may give either of two 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:

```
3Ø PRINT S return
```

*NOTE: PRINT has been misspelled.*

The computer responds:

```
ERROR
```

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

```
ERROR: return
```

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

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

To correct the statement, he retypes it in the proper form:

```
3Ø 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:

```
3Ø PRINT S return
```

```
ERROR return
```

```
3Ø PRINT S
```

Appendix "D" contains a list of TSB diagnostic messages.

## RUNNING THE SAMPLE PROGRAM

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

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

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

```
RUN return linefeed
linefeed
linefeed
linefeed
```

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

```
? 95.6,87.3,80.5,90,82.8 return
```

The answer is printed:

```
84.24 return linefeed
```

? indicates that five more numbers are expected:

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

*NOTE: A ?? message indicates that more input is required by the program.*

The answer is printed:

```
2.1 return linefeed
```

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

```
? CC return
```

The program is finished:

```
DONE
```

Log off:

```
BYE return
```

Time used is printed:

```
003 MINUTES OF TERMINAL TIME
```

STOPPING A PROGRAM: THE break KEY

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

break Key

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

break

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

STOP

after break is pressed.

Remember that:

C<sup>c</sup> return

and not break is used to terminate input loops.

COMMENTS

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

## HOW THE PROGRAM WORKS

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

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

After line 10 is executed, the variables and their assigned values, typed in by the user, are stored. For example, using the values entered by the user in the previous example, this information is stored: A = -12.5; B = -50.6; C = -32; D = 45; E = 6.60

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 the user's teleprinter:

```
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 instructions 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<sup>C</sup> 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<sup>C</sup> 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.

Vocabulary

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:  $1.000000E+06$  is equal to 1,000,000 and is read "1 times 10 to the sixth power ( $1 \times 10^6$ ).

$1.020000E+04$  is equal to 10,200

$1.020000E-04$  is equal to .000102

COMMENTS

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

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

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



TERM: SIMPLE VARIABLE

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

EXAMPLES: A0 B  
M5 C2  
Z9 D

Vocabulary

COMMENTS

Variables are used to represent numeric values.

For instance, in the statement:

10 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

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

Asmt. Oper.

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 1Ø and 5Ø above.

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

## ARITHMETIC OPERATORS

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

Arith. Oper.

PURPOSE

Represents an arithmetic operation, as:

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

COMMENTS

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

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

The order of precedence (hierarchy) is:

$\uparrow$
* /
+ -

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

## RELATIONAL OPERATORS

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

Rel. Oper.

PURPOSE

Determines the logical relationship between two expressions, as

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

COMMENTS

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

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

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

## MIN AND MAX OPERATORS

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

PURPOSE

Selects the larger or smaller value of two expressions.

COMMENTS

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

## THE AND OPERATOR

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

AND Oper.

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 Ø.

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

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

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

THE OR OPERATOR

SYMBOL: OR

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

OR Oper.

PURPOSE

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

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

COMMENTS

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

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

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

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



## THE NOT OPERATOR

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

NOT Oper.

PURPOSE

Logically evaluates the complement of a given expression.

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

COMMENTS

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

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

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

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

## ORDER OF PRECEDENCE (OF EXECUTION)

The order of performing operations is:

↑ *highest precedence*

NOT

\* /

+ -

MIN MAX

*Relational Operators*

AND

OR *lowest precedence*

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

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

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

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

Parentheses override the order of precedence  
in all cases.

## STATEMENTS

Be sure you know the difference between statements and commands.

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

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

Here are some examples mentioned in Section I:

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

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



# THE ESSENTIALS OF BASIC

## REM

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

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

### PURPOSE

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

### COMMENTS

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

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

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

## GO TO AND MULTIBRANCH GO TO

EXAMPLES:           10 LET X = 20  
                           :  
                           40 GO TO 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  $X \leq 10$ , the message in statement 60 is printed.

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

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

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

## FOR...NEXT

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

---

*Sample Program - Variable Number Of Loops*

```

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

```

## GENERAL FORM:

statement number FOR simple variable = initial value TO final value

or

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

:

statement number NEXT simple variable

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



THE ESSENTIALS OF BASIC  
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.

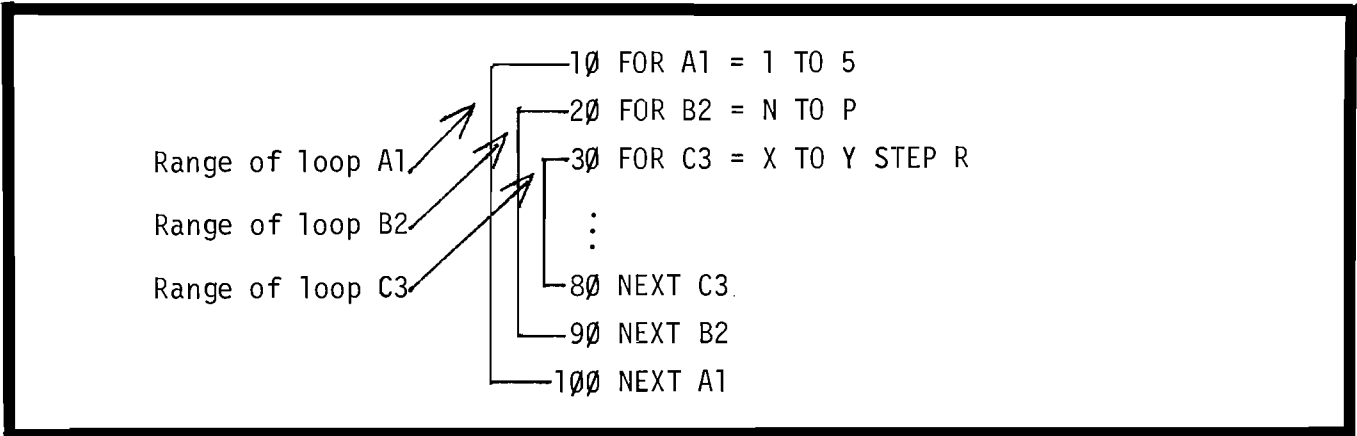
Try running the sample program if you are not sure what happens when FOR...NEXT loops are used in a program.

FOR...NEXT

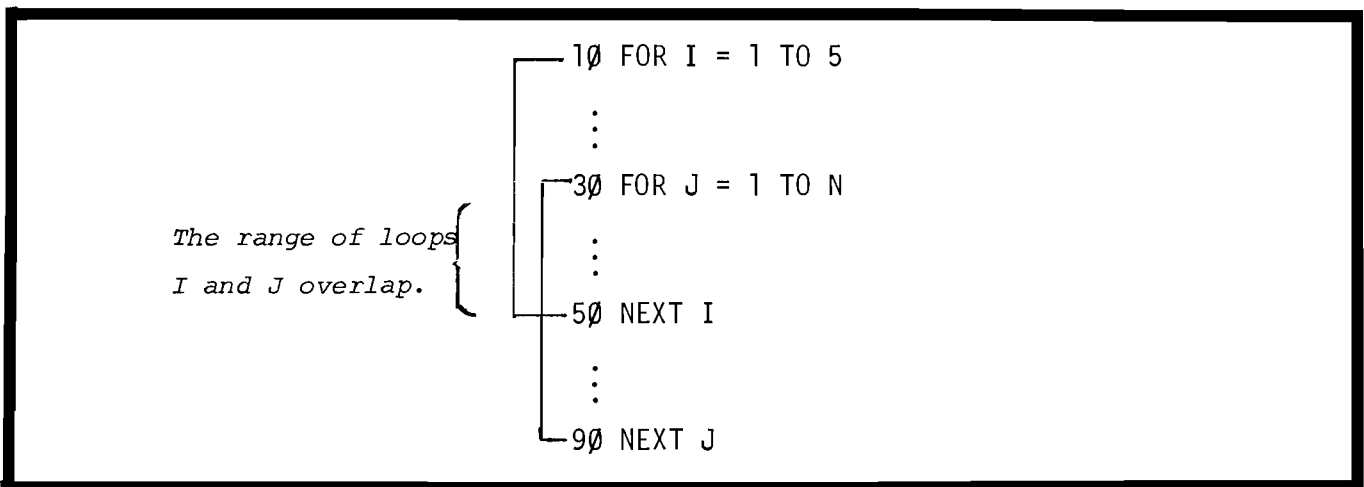
## NESTING FOR...NEXT LOOPS

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

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



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



THE ESSENTIALS OF BASIC  
READ, DATA AND RESTORE

Sample Program using READ and DATA

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

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

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

## READ, DATA AND RESTORE, CONTINUED

## Sample Program Using READ, DATA and RESTORE

```

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

```

## GENERAL FORM:

statement number READ variable , variable ,...

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

statement number RESTORE

statement number RESTORE statement number

PURPOSE

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

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

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

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

READ, DATA AND RESTORE, CONTINUED

COMMENTS

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

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

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

DATA statements do not execute; they merely specify data.

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

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

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

## INPUT

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

## Sample Program Using INPUT

```

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

```

## ----- RESULTS -----

RUN

?1 return

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

WHAT VALUE SHOULD BE ASSIGNED TO R?27

1	2	3	4	5	6	7	R=27
---	---	---	---	---	---	---	------

?1.5 return

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

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

WHAT VALUE SHOULD BE ASSIGNED TO R?-99

1.5	2.5	3.5	4.5	6	7.2
-----	-----	-----	-----	---	-----

8.1 R=-99

DONE

GENERAL FORM:

statement number INPUT variable , variable ,...

PURPOSE

Assigns a value input from the teleprinter to a variable.

Continued on next page.

## INPUT, CONTINUED

COMMENTS

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

Only one question mark will be printed for each INPUT statement. The statement:

1Ø INPUT A, B2, C5, D, E, F, G

*and*

2Ø INPUT X

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

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

Relevant Diagnostics:

? indicates that input is required.

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

??? indicates that TSB cannot decipher your input.

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

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

## PRINT

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

```

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

```

## ----- RESULTS -----

RUN return

```

10          10          10          20          20
1          20.05
NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE
VALUE IN THE SAME STATEMENT.
'A' DIVIDED BY 'E9' = .5
11111          22222          33333          44444          55555
66666
111112222233333444445555566666
DONE

```

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

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

PRINT

PURPOSE

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

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

COMMENTS

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

## END AND STOP

## EXAMPLES:

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

## GENERAL FORM:

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

PURPOSE

Terminates execution of the program and returns control to TSB.

COMMENTS

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

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

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

## SAMPLE PROGRAM

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

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

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

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

## SAMPLE PROGRAM

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

## SAMPLE PROGRAM, CONTINUED

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

## RUNNING THE SAMPLE PROGRAM

RUN return

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY.

IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS.

PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.

HOW MANY NUMBERS DO YOU WANT TO AVERAGE? 5 return

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

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

NOW, LET'S BEGIN

? 99 return

? 87.6 return

? 92.7 return

? 79.5 return

? 84 return

5 NUMBERS WERE INPUT.

THEIR SUM IS: 442.8

THEIR AVERAGE IS: 88.56

DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?

TYPE 1 IF YES, 0 IF NO

BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.

YOUR REPLY? 2 return

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

YOUR REPLY? 1 return

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

DONE

## COMMANDS

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

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

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

```
HEL_H200,SECCRECT 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 to write your own programs.

## HELLO

EXAMPLE: HELLO-D007,POS<sup>C</sup>T return

or

HEL-D007,POS<sup>C</sup>T return

GENERAL FORM: HELLO- *IDcode* , *password*

or

HEL- *IDcode* , *password*

### PURPOSE

The command used to log in to the TSB system.

### COMMENTS

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

HELLO



BYE

EXAMPLE:           BYE return  
                      009 MINUTES OF TERMINAL TIME

GENERAL FORM:     BYE

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-

ECHO

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

PURPOSE

Allows use of a half duplex teleprinter terminal.

COMMENTS

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

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

May be abbreviated to its first three letters.

## RUN

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

RUN

PURPOSE

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

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

COMMENTS

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

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

C<sup>c</sup> return



## SCRATCH

EXAMPLE:	SCRATCH <u>return</u> or SCR <u>return</u>
GENERAL FORM:	<u>SCRATCH</u> or <u>SCR</u>

SCRATCH

### 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

RENUMBER

EXAMPLES:

RENUMBER returnREN return

REN-100

REN-10, 1 returnREN-20, 50 return

GENERAL FORM:

REN

or

REN=number assigned to first statement

or

REN=number assigned to first statement , interval between new statement numbersPURPOSE

Renumbers statements in a program.

COMMENTS

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

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

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

Remember that text contained in REM and PRINT statements is 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:

C<sup>c</sup> return

which produces the DONE message.

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

## PUNCH

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 will be 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 <u>return</u>
	TAP <u>return</u>
GENERAL FORM:	<u>TAPE</u>
	or
	<u>TAP</u>

TAPE

### PURPOSE

Informs the computer that following input is from paper tape.

### COMMENTS

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

TSB responds to the TAPE command with a linefeed.

KEY

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

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

KEY may not be abbreviated.

Any command followed by a *return* will have 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

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.



## SECTION III

### ADVANCED BASIC

This section describes 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 divided into columns (vertical) and rows (horizontal).

Arrays may have one or two dimensions. For example,

1.Ø  
2.1  
3.2  
4.3

is a one dimensional array, while

6 , 5 , 4  
3 , 2 , 1  
Ø , 9 , 8

is a two dimensional array.

Array elements are referenced by their column and row position. For instance, if the examples above were arrays A and Z respectively, 2.1 would be A(2); similarly, Ø 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.



ADVANCED BASIC

TERM: WORD

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

COMMENTS

Numbers require two words of storage each.

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".

## STORING AND DELETING PROGRAMS

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

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

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

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

## LENGTH

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

PURPOSE

Prints the number of two-character words in the program currently being accessed from the teleprinter. 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.



## SAVE

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

PURPOSE

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

COMMENTS

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

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

KILL-SAMPLE return (Deletes the stored version.)

linefeed

NAME-SAMPLE return (Names the current program)

linefeed

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

linefeed

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

## GET- AND GET-\$

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

PURPOSE

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

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

COMMENTS

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

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

KILL-

EXAMPLE:	KILL-PROG12 <u>return</u>
	KIL-EXMPLE <u>return</u>
	KIL-FILE10 <u>return</u>
GENERAL FORM:	<u>KILL- program or file to be deleted</u>
	or
	<u>KIL- program or file to be deleted</u>

PURPOSE

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

COMMENTS

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

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

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

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

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

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

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

KILL

## APPEND-

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

PURPOSE

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

COMMENTS

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

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

DELETE

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*

BINOPO 0594	CDETER 0706	CSHFLO 1598	CURFIT 1618	DIFFEQ 0133	DIVID 1367
FNCTS 0652	GEOMEN 0199	IN4 5440	IN5 5440	INVHIL 0250	LINFIT 0492
ROMINT 0299	SQE 0209	STAT11 0568	TAB 2098	YELLOW 0227	Z123 0413

## GENERAL FORM:

LIBRARY*or*LIB

LIBRARY

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.

## 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.

## SUBROUTINES AND FUNCTIONS

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

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

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

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

## GOSUB...RETURN

```

EXAMPLE:      50 READ A2
              60 IF A2<100 THEN 80
              70 GOSUB 400
              :
              380 STOP (STOP, 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: statement number GOSUB statement number starting subroutine
              :
              statement number RETURN

```

PURPOSE

GOSUB transfers control to the specified statement number.

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

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

COMMENTS

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

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

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

## 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 will be 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
                :

```

## GENERAL FORM:

statement number GOSUB expression OF sequence of statement numbers

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 -1  
                       80 FOR P = 1 TO N STEP R  
                       90 FOR X = N TO W STEP (N+2-V)  
                       :  
                       :

## GENERAL FORM:

statement number FOR simple variable = expression TO expression STEP expression

PURPOSE

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

COMMENTS

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

100 FOR N = 1 TO 2 STEP .01

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

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

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



## DEF FN

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

GENERAL FORM:

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

### PURPOSE

Allows the programmer to define functions.

### COMMENTS

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

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

1Ø DEF FNA (Y) = FNB (X)

2Ø DEF FNB (X) = FNA (Y)

will cause infinite looping.

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

## GENERAL MATHEMATICAL FUNCTIONS

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

PURPOSE

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

- ABS (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  $\leq$  the expression;
- LOG (expression), the logarithm of the positively valued expression to the base  $e$ ;
- RND (expression), a random number between 1 and Ø; 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, Z  
                       540 PRINT TAB (X+2) "HEADING"; R5

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

The function forms are:

TAB ( expression indicating column number )

SGN ( expression )

PURPOSE

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

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

## THE TYP FUNCTION

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

PURPOSE

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

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

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

COMMENTS

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

## THE LEN FUNCTION

EXAMPLES:           58Ø IF LEN (B\$) >= 21 THEN 9999  
                      8ØØ IF LEN (C\$) = R THEN 1ØØØ  
                      85Ø PRINT LEN (N\$)  
                      88Ø 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.

## FILES

### 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.



## FILES

### 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.

### TERMS: 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.

## FILES

### 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

#### 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

## FILES

### KILL-

EXAMPLE:	KILL-NAMEXX <u>return</u>
	KIL-EXMPLE <u>return</u>
	KIL-FILE1Ø <u>return</u>
GENERAL FORM:	<u>KILL-file to be deleted</u>
	<u>KIL-file to be deleted</u>

### 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.

KILL-

## FILES

## FILES

EXAMPLE: 1Ø FILES MATH, DATA, 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,

```
1ØØ PRINT #2;A
```

would print the value of A on the file named DATA.

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.

## FILES

### 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.

## FILES

### READ #

EXAMPLES:                   65 READ#5; A,B,C  
                              7Ø READ#3; B\$  
                              8Ø READ#N; A, B\$, C(5,3)  
                              9Ø 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.

READ#

## FILES

### 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.

## 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:

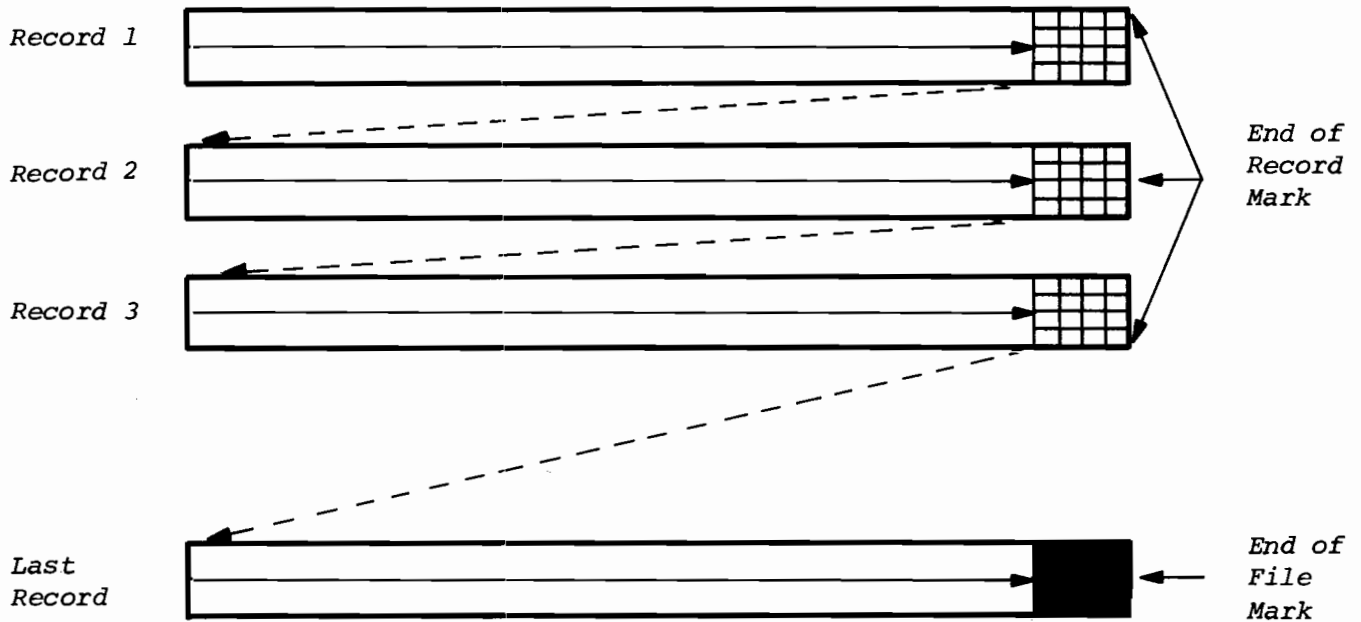
- a) File space may be wasted by creating files larger than necessary.
- b) Time may be wasted if a file is too small: it terminates the program, and must be enlarged, and the program re-run.

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.



# FILES

## STRUCTURE AND STORAGE PATTERN



### COMMENTS

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

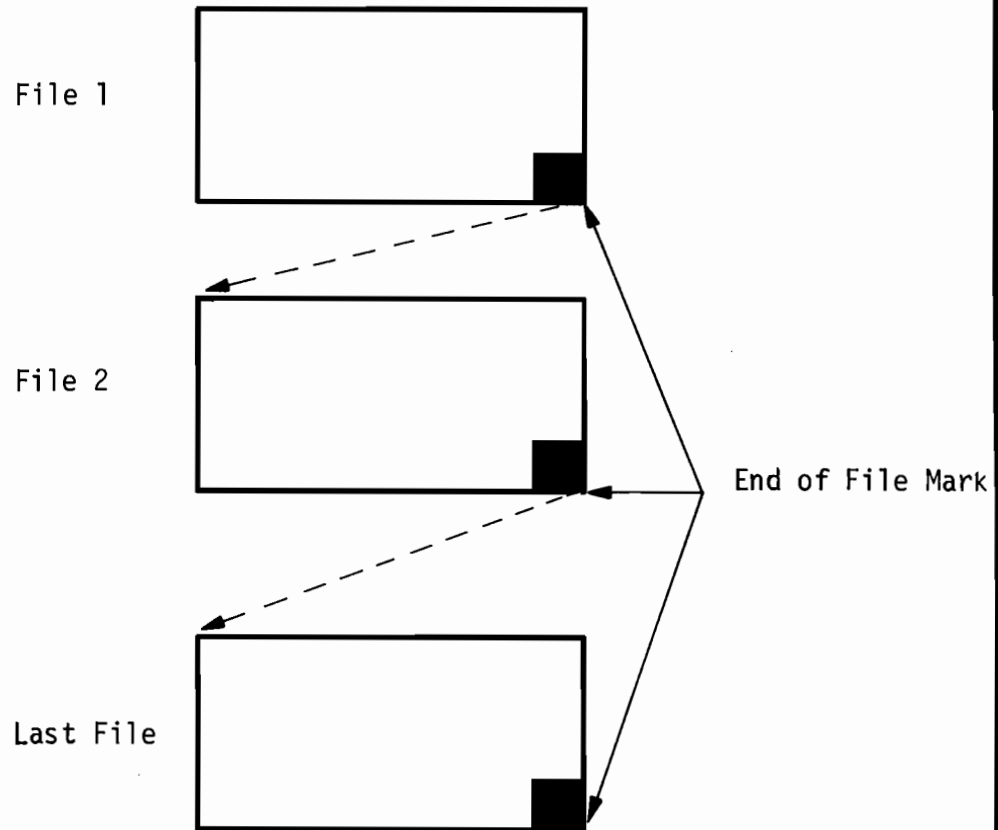
Each record contains 64 words of storage space.

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.)



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 100

```

fills files sequentially, moving to the next file when the current file is filled.

## FILES

### 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:

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

if the number of characters is odd.

or

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

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.

## FILES

### 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; 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.

## FILES

### 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#

## 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 will be 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.

# FILES

## 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.

READ#

### COMMENTS

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

Non-integers will be 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.

## READ# TO RESET A POINTER

EXAMPLES:                   41Ø READ #2, 3  
                               42Ø READ #N, 1  
                               43Ø READ #(N-P), 5  
                               44Ø 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 will be 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 TSB, and assumes that the programmer has some knowledge of matrix theory.

---

#### TERM: MATRIX (ARRAY)

DEFINED IN TSB 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:

1	2	3
4	5	6
7	8	9

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

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

## MATRICES

### DIM

EXAMPLES:                   11Ø DIM A (5Ø), B(2Ø,2Ø)  
                              12Ø DIM Z (5,2Ø)  
                              13Ø DIM S (5,25)  
                              14Ø DIM R (4,4)

#### GENERAL FORM:

statement number DIM matrix variable ( integer ) ...

or

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

#### PURPOSE

Reserves working space in the TSB system for a matrix within the limits set by the original DIM statement.

#### COMMENTS

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

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

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

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

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

## MAT...ZER

## EXAMPLES:

305 MAT A = ZER  
 310 MAT Z = ZER (N)  
 315 MAT X = ZER (30, 10)  
 320 MAT R = ZER (N, P)

## GENERAL FORM:

statement number MAT matrix variable = ZER

or

statement number MAT matrix variable = ZER ( expression )

or

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

PURPOSE

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

COMMENTS

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

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

## MATRICES

### MAT...CON

#### EXAMPLES:

205 MAT C = CON

210 MAT A = CON (N,N)

220 MAT Z = CON (5,20)

230 MAT Y = CON (50)

#### GENERAL FORM:

statement number MAT matrix variable = CON

or

statement number MAT matrix variable = CON ( expression )

or

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

#### PURPOSE

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

#### COMMENTS

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

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

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

## MATRICES

### 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 will be 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.

INPUT

## MAT INPUT

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

## GENERAL FORM:

statement number MAT INPUT matrix variable

or

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

or

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

PURPOSE

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

COMMENTS

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

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

Elements being input must be separated by commas.

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

Only one ? will be 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.

## PRINT

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 will be 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.

## MATRICES

### 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.

MAT PRINT



## MATRICES

### READ

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

#### GENERAL FORM:

statement number READ matrix variable ( expression )

or

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

#### PURPOSE

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

#### COMMENTS

Expressions (subscripts) should evaluate to integers. Non-integers will be 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.

## MATRICES

### 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.

## MATRICES

### MATRIX ADDITION

EXAMPLES:

31Ø MAT C = B + A

32Ø MAT X = X + Y

33Ø MAT P = N + M

GENERAL FORM:

statement number MAT matrix variable = matrix variable + matrix variable

#### PURPOSE

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

#### COMMENTS

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

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

MAT ADD.

## MATRICES

### MATRIX SUBTRACTION

EXAMPLES:                   55Ø MAT C = A - B  
                              56Ø MAT B = B - Z  
                              57Ø MAT X = X - A

GENERAL FORM:

statement number MAT matrix variable = matrix variable - matrix variable

#### PURPOSE

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

#### COMMENTS

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

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

# MATRICES

## MATRIX MULTIPLICATION

EXAMPLES:                    93Ø MAT Z = B \* C  
                                 94Ø MAT X = A \* A  
                                 95Ø MAT C = Z \* B

GENERAL FORM:

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

### PURPOSE

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



### COMMENTS

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

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

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

MAT MULT.

## MATRICES

### SCALAR MULTIPLICATION

EXAMPLES:

11Ø MAT A = (5) \* B

115 MAT C = (1Ø) \* C

12Ø MAT C = (N/3) \* X

13Ø 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.

SCALAR MULT.

## MATRICES

### 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.

MAT COPY

## MATRICES

### 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



## MATRICES

### MATRIX TRANSPOSITION

EXAMPLES:                    959 MAT Z = TRN (A)  
                                 969 MAT X = TRN (B)  
                                 979 MAT Z = TRN (C)

GENERAL FORM:

statement number MAT matrix variable = TRN ( matrix variable )

#### PURPOSE

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

#### COMMENTS

Sample transposition:

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

Note that the dimensions of the resulting matrix must be the reverse of the original matrix. For instance, if A has dimensions of 6,5 and MAT C = TRN (A), C must have dimensions of 5,6.

MAT Trans.

## MATRICES

### MATRIX INVERSION

EXAMPLES:            38Ø MAT A = INV(B)  
                      39Ø MAT C = INV(A)  
                      4ØØ 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 4ØØ, above.

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

## MATRICES

### MAT PRINT#

EXAMPLES:                   52Ø MAT PRINT #5; A  
                              53Ø MAT PRINT #6, 3; B  
                              54Ø MAT PRINT #4,M; A  
                              55Ø MAT PRINT #N,M; A

#### GENERAL FORM:

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

or

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

#### PURPOSE

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

#### COMMENTS

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 PRINT

## MATRICES

### MAT READ#

#### EXAMPLES:

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

#### GENERAL FORM:

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

or

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

or

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

or

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

MAT READ#

#### 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.

## SECTION VI

### STRINGS

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."

## STRINGS

### TERM: STRING

DEFINED IN	Ø to 72 teleprinter characters,
TSB AS:	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\$ = ABCDEF, 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:

```
CDEF
```

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.



## 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 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\$  
                              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.

## PRINTING STRINGS

## EXAMPLES:

```

100 LET B3 = 642
105 PRINT A$
110 PRINT A$, B$, Z$
115 PRINT C$(8) "END OF STRING" B3
120 PRINT C$(1,7)
130 PRINT "THE TOTAL IS: ";X5

```

## GENERAL FORM:

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

PURPOSE

Causes the current value of the specified string or substring variable to be output to the teleprinter.

COMMENTS

String and numeric values may be mixed in a PRINT statement.

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.

O<sup>C</sup> and N<sup>C</sup> generate a return and linefeed respectively when printed as string characters.

## STRINGS

### 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 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; see "The TYP Function", Section III for description of a data type check which may be used with DATA statements.

## STRINGS

### 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.

## THE LEN FUNCTION

## EXAMPLE:

```

469 PRINT LEN (A$)
479 PRINT LEN (X$)
489 PRINT "TEXT"; LEN(A$); B$, C
499 IF LEN (P$) #5 THEN 600
509 IF LEN (P$) = 5 THEN 609
519 IF LEN (P$) = 5 OR LEN (P$) = 10 THEN 10
529 LET X$(LEN(X$)+1) = "ADDITIONAL SUBSTRING"
:
600 STOP
609 PRINT "STRING LENGTH = "; LEN (P$)

```

## GENERAL FORM:

statement number statement type LEN ( string variable ) ...

PURPOSE

Supplies the current (logical) length of the specified string, in number of characters.

COMMENTS

DIM merely specifies a maximum string length. The LEN function allows the user to check the actual number of characters currently assigned to a string variable.

Note that LEN is a directly executable command (See Section III), while LEN (...\$) is a pre-defined function used only as an operand in a statement. The LEN command gives the working program length; the LEN function gives the current length of a string.

## STRINGS IN DATA STATEMENTS

EXAMPLES:                   500 DATA "NOW IS THE TIME."  
                              510 DATA "HOW", "ARE", "YOU,"  
                              520 DATA 5.172, "NAME?", 6.47,5071

## GENERAL FORM:

*statement number* DATA " *string text* " , " *string text* " ...

PURPOSE

Specifies data in a program (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.

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$, Y$, D$
360 PRINT #7,3; X$, P$, "TEXT", 27.5,R7
365 PRINT #N,R; P$, N, A(5,5), "TEXT"

```

## GENERAL FORM:

statement number PRINT file number , record number formula ; string variable ...

or

statement number PRINT file number formula , record number formula ; " string text "...

or

statement number PRINT file number formula ; string variable or substring variable...

PURPOSE

Prints string or substring variables on a file.

COMMENTS

String and numeric variables may be mixed in a single file or record within a file (example statement 360 above).

The formula for determining the number of 2-character words required for storage of a string on a file is:

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

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

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.



## STRINGS

### READING STRINGS FROM FILES

#### EXAMPLES:

71Ø READ #1, 5; A\$, B\$

715 READ #2; C\$, A1, B2, X

72Ø READ #3,6; C\$(5),X\$(4,7),Y\$

73Ø READ #N,P; C\$, V\$(2,7), R\$(9)

#### GENERAL FORM:

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

or

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

#### PURPOSE

Reads string and substring values  
from a file.

#### COMMENTS

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

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

STR. READ#



## SECTION VII

### LOGICAL OPERATIONS

#### LOGICAL VALUES AND NUMERIC VALUES

A distinction should be made between logical values and the numeric values produced by logical evaluation, when using the logical capability of Time Shared BASIC.

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

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

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

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

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

1, meaning the expression or statement is "true",

or

0, meaning the expression or statement is "false".

## LOGICAL OPERATIONS

### RELATIONAL OPERATORS

There are two ways to use the relational operators in logical evaluations:

1. As a simple check on the numeric value of an expression.

EXAMPLES:	150 IF B=7 THEN 600
	200 IF A#27.65 THEN 700
	300 IF (Z/10)>=0 THEN 800

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

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

Continued on the next page.

## LOGICAL OPERATIONS

### 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.

## LOGICAL OPERATIONS

### BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.

1. As logical checks on the value of an expression or expressions.

EXAMPLES:

```
51Ø IF A1 OR B THEN 67Ø
52Ø IF B3 AND C9 THEN 68Ø
53Ø IF NOT C9 THEN 69Ø
54Ø IF X THEN 7ØØ
```

Statement 51Ø is interpreted: "if either A1 is true (has a nonzero value) or B is true (has a nonzero value) then transfer control to statement 67Ø."

Similarly, statement 54Ø is interpreted: "if X is true (has a nonzero value) then transfer control to statement 7ØØ."

The Boolean operators evaluate expressions for their logical values only; these are "true" = any non-zero value, "false" = zero. For example, if B3 = 9 and C9 = -5, statement 52Ø would evaluate to "true", since both B3 and C9 have a nonzero value.

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

EXAMPLES:

```
49Ø LET B = C = 7
5ØØ PRINT B AND C
51Ø PRINT C OR B
52Ø PRINT NOT B
```

Statements 5ØØ - 52Ø returns a numeric value of either: 1, indicating that the statement has a logical value of "true", or Ø, indicating a logical value of "false".

Note that the criteria for determining the logical values are:

true = any nonzero expression value

false = an expression value of Ø.

The numeric value 1 or Ø is assigned accordingly.

## LOGICAL OPERATIONS

### 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:          31Ø IF (A9 MIN B7)<Ø OR (A9 MAX B7)>1ØØ THEN 9ØØ
                   31Ø PRINT (A>B) AND (X<Y)
                   32Ø LET C = NOT D
                   33Ø IF (C7 OR D4) AND (X2 OR Y3) THEN 93Ø
                   34Ø IF (A1 AND B2) AND (X2 AND Y3) THEN 94Ø
```

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:

```
9Ø LET X = Ø
1ØØ FOR I = 1 TO N
11Ø READ #1; A
12Ø LET X = X+(A=Ø)
13Ø NEXT I
14Ø PRINT N; "VALUES WERE READ."
15Ø PRINT X; "WERE ZEROES."
16Ø PRINT (N-X); "WERE NONZERO."
```

Note that X is increased by 1 or Ø each time A is read; when A = Ø, the expression A = Ø 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 any ambiguity in descriptions of BASIC language features presented in other sections.

The other subsections give technical information of interest to the sophisticated user.

FOR THE PROFESSIONAL  
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> <literal string>
<number>	::= <decimal number> <decimal number><exponent part>
<decimal number>	::= <integer> <integer>. <integer>.<integer> .<integer>
<integer>	::= <digit> <integer><digit>
<digit>	::= 0 1 2 3 4 5 6 7 8 9
<exponent part>	::= E<integer> E+<integer> E-integer ( <i>see rule 1</i> )
<literal string>	::= "<character string>"
<character string>	::= <character> <character string><character>
<character>	::= any ASCII character except null, line feed, return, x-off, 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>MIN<sum> <minmax>MAX<sum>
<sum>	::= <term> <sum>+<term> <sum>-<term>
<term>	::= <factor> <subterm>*<factor> <subterm>/<factor>
<subterm>	::= <denial> <signed factor>



## 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

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

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

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

*Lowest:*    bell

space	5	I
!	6	J
#	7	K
\$	8	L
%	9	M
&	:	N
'	;	O
(	<	P
)	=	Q
*	>	R
+	?	S
,	@	T
-	A	U
.	B	V
/	C	W
0	D	X
1	E	Y
2	F	Z
3	G	[
4	H	\
5		]
		↑ <i>Highest</i>



ASCII Code

Quotation marks are used to delimit strings, and may not be used within a string.

## MEMORY ALLOCATION BY A TSB 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 @<sup>C</sup> (control shift "P") several times to put leading holes on the tape.
4. Type program as usual, following each line with return linefeed.
5. Press "HERE IS"; or press @<sup>C</sup> several times to put trailing holes on the tape.
6. Press the "OFF" button on the tape punch.

COMMENTS

1. The standard on-line editing features, such as esc, ←, and repeating the same line number may be punched on tape; esc must be followed by return linefeed.
2. 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

Teleprinters 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.



## 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
```

## SAMPLE PROGRAM-INTEREST RATES

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
```

## SAMPLE PROGRAM-AN ELECTRONIC CALENDAR

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
```



## SAMPLE PROGRAM-AN ELECTRONIC CALENDAR, CONTINUED

```
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-AN 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
```

## 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 31,28,31,30,31,30,31,31,30,31,30,31  
9999 END
```

## SAMPLE PROGRAMS

H-P FOOTBALL

This program is a football game, based on the random number generator. Notice how the GO TO's and GOSUB's 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 P1=51
180 LET Q1=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, 0 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

```

## 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)+3+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>50 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

```

## 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 GOTO 1610
1510 PRINT
1520 PRINT "PACKARD'S BALL."
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 X=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

```



## SAMPLE PROGRAMS, CONTINUED

H-P FOOTBALL

```
2260 PRINT "KICK IS GOOD"
2270 LET S[2-P]=S[2-P]+7
2280 GOTO 2220
2290 PRINT "KICK IS OFF TO THE SIDE"
2300 LET S[2-P]=S[2-P]+6
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<80 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!!!"
2470 LET S[3]=S[3]+3
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!!!"
2550 LET S[1]=S[1]+3
2560 GOSUB 2220
2570 GOTO 310
2580 END
```



## DIAGNOSTIC MESSAGES

ARGUMENT OF SIN OR TAN TOO BIG  
ARRAY OF UNKNOWN DIMENSIONS  
ARRAY TOO LARGE  
BAD FORMAT IN FILES STATEMENT  
BAD INPUT, RETYPE FROM ITEM  
CHARACTERS AFTER COMMAND END  
CHARACTERS AFTER STATEMENT END  
DATA OF WRONG TYPE  
DIMENSIONS NOT COMPATIBLE  
DIVIDE BY ZERO - WARNING ONLY  
END-OF-FILE/END OF RECORD  
EXP OVERFLOW - WARNING ONLY  
EXTRA INPUT - WARNING ONLY  
EXTRANEOUS LIST DELIMITER  
FUNCTION DEFINED TWICE  
GOSUBS NESTED TEN DEEP  
ILLEGAL EXPONENT  
ILLEGAL OR MISSING INTEGER  
ILLEGAL READ VARIABLE  
ILLEGAL SYMBOL FOLLOWS 'MAT'  
LAST INPUT IGNORED, RETYPE IT  
LAST STATEMENT NOT 'END '

LOG OF NEGATIVE ARGUMENT  
LOG OF ZERO - WARNING ONLY  
MATRIX CANNOT BE ON BOTH SIDES  
MATRIX NOT SQUARE  
MISSING ASSIGNMENT OPERATOR  
MISSING LEFT PARENTHESIS  
MISSING OR BAD ARRAY VARIABLE  
MISSING OR BAD FILE REFERENCE  
MISSING OR BAD FUNCTION NAME  
MISSING OR BAD LIST DELIMITER  
MISSING OR BAD SIMPLE VARIABLE  
MISSING OR BAD STRING OPERAND  
MISSING OR ILLEGAL DATA ITEM  
MISSING OR ILLEGAL 'OF'  
MISSING OR ILLEGAL 'STEP'  
MISSING OR ILLEGAL SUBSCRIPT  
MISSING OR ILLEGAL 'THEN'  
MISSING OR ILLEGAL 'TO'  
MISSING OR PROTECTED FILE  
MISSING RELATIONAL OPERATOR  
MISSING RIGHT PARENTHESIS  
NEARLY SINGULAR MATRIX

## DIAGNOSTIC MESSAGES, CONTINUED

NEGATIVE NUMBER TO REAL POWER	SECOND FILES STATEMENT
NEGATIVE STRING LENGTH	72 CHARACTERS MAX FOR STRING
NEXT WITHOUT MATCHING FOR	SIGN WITHOUT NUMBER
NO '*' AFTER RIGHT PARENTHESIS	STATEMENT HAS EXCESSIVE LENGTH
NO LEGAL BINARY OPERATOR FOUND	STRING OVERFLOW
NO CLOSING QUOTE	STRING VARIABLE NOT LEGAL HERE
NON-CONTIGUOUS STRING CREATED	SQR OF NEGATIVE ARGUMENT
NON-EXISTENT FILE REQUESTED	SUBSCRIPT OUT OF BOUNDS
NO STATEMENT TYPE FOUND	UNDECIPHERABLE OPERAND
OUT OF DATA	UNDEFINED FUNCTION
OUT OF STORAGE	UNDEFINED STATEMENT REFERENCE
OVERFLOW - WARNING ONLY	UNDEFINED VALUE ACCESSED
OVER/UNDERFLOWS - WARNING ONLY	UNDERFLOW - WARNING ONLY
PARAMETER NOT STRING VARIABLE	UNMATCHED FOR
READ-ONLY FILES:	VARIABLE DIMENSIONED TWICE
REDIMENSIONED ARRAY TOO LARGE	WRITE TRIED ON READ-ONLY FILE
RETURN WITH NO PRIOR GOSUB	ZERO TO NEGATIVE POWER-WARNING
SAME FOR-VARIABLE NESTED	ZERO TO ZERO POWER

*Diagnostic messages printed while entering a program refer only to the first error found in a line.*

? (Input is required to continue execution.)

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

??? (Input is unintelligible.)

## SPECIAL CHARACTERS

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

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



## OPERATORS

<u>SYMBOL</u>	<u>SAMPLE STATEMENT</u>	<u>PURPOSE/MEANING/TYPE</u>
=	100 A=B=C=J	Assignment operator; assigns a value to a variable;
	110 LET A = 0	May also be used with LET.
↑	120 PRINT X↑2	Exponentiate (as in $X^2$ ).
*	130 LET C5 = (A*B)*N2	Multiply
/	140 PRINT T5/4	Divide
+	150 LET P = R1 +10	Add
-	160 X3 = R3 - P	Subtract
<i>NOTE: The numeric values used in logical evaluation are: "true" = any nonzero number; "false" = 0.</i>		
=	170 IF D=E THEN 600	<u>expression</u> "equals" <u>expression</u>
#	180 IF (D+E)#(2*D)THEN 710	<u>expression</u> "does not equal" <u>expression</u>
<>	180 IF(D+E)<>(2*D)THEN 700	<u>expression</u> "does not equal" <u>expression</u>
>	190 IF X>10 THEN 620	<u>expression</u> "is greater than" <u>expression</u>
<	200 IF R8<P7 THEN 640	<u>expression</u> "is less than" <u>expression</u>
>=	210 IF R8>=P7 THEN 710	<u>expression</u> "is greater than or equal to" <u>expression</u>
<=	220 IF X2<=10 THEN 650	<u>expression</u> "is less than or equal to" <u>expression</u>
AND	230 IF G2 AND H5 THEN 900	<u>expression1</u> AND <u>expression2</u> must both be "true" for statement to be "true"
OR	240 IF G2 OR H5 THEN 910	If either <u>expression1</u> OR <u>expression2</u> is "true", statement is "true"
NOT	250 IF NOT G5 THEN 950	Statement is "true" when <u>expression</u> (NOT G5) is "false".
MAX	260 LET B = A2 MAX C3	Evaluates for the larger of the two expressions
MIN	270 LET B1 = A7 MIN A9	Evaluates for the smaller of the two expressions

## STATEMENTS

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

## STATEMENTS , CONTINUED

<u>NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE</u>
RESTORE	380 RESTORE	Permits re-reading data without re-running the program.
	385 RESTORE <i>n</i>	Permits data to be re-read, beginning in statement <i>n</i> .
RETURN	850 RETURN	Transfers control to statement following its GOSUB.
STOP	410 STOP	Terminates the program; may be used anywhere in program.



## COMMANDS

*NOTE: Commands are executed immediately; they do not require statement numbers.*

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

Continued on next page.

## COMMANDS, CONTINUED

<u>FULL NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE</u>
LIST	LIS	Produces a listing of current program.
	LIS-150	Produces a listing, starting at specified statement.
NAME	NAM-SAMPLE	Assigns specified name to the current program; name may be 1 to 6 characters in length and must include only printing characters.
PUNCH	PUN	Punches current program to paper tape.
	PUN-50	Punches program to paper tape, beginning at specified statement.
RENUMBER	REN	Renumbers program from 10 in multiples of 10.
	REN-50	Renumbers program from specified statement number in multiples of 10.
	REN-60,y	Renumbers program from specified statement number in multiples of y.
RUN	RUN	Starts program execution.
	RUN-50	Starts program execution at specified statement.
SAVE	SAV	Saves the current program in user's library.
SCRATCH	SCR	Erases current program (but not program name.)
TAPE	TAP	Informs computer that following input is from paper tape.
TIME	TIM	Produces a listing of terminal and account time.

## FUNCTIONS

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

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

## 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\$).

<u>FULL NAME</u>	<u>EXAMPLE</u>	<u>PURPOSE</u>
DIM	10 A\$ (27)	Declares string length in characters.
LET	20 A\$ = "**TEXT 1"	Assigns the character string in quotes to a string variable.
LEN	30 PRINT (B\$)	Gives the current length of the specified string.
=	105 IF A\$=C\$ THEN 600	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.
#	110 IF B\$#X\$ THEN 650	
>	115 IF N\$(2,2)>B\$(3,3) THEN 10	
<	120 IF N\$<B\$ THEN 999	
>=	125 IF P\$ (5,8)>=Y\$(4,7)THEN 10	
<=	130 IF X\$<=Z\$ THEN 999	
INPUT	205 INPUT N\$	Accepts the appropriate number of characters (followed by a <u>return</u> ). The characters need not be in quotation marks if only one string is input.
INPUT	210 INPUT N\$,X\$,Y\$	Inputs the specified strings; input must be in quotes and separated by commas.
READ	215 READ P\$	Reads a string from a DATA statement; each string read must be enclosed in quotes.
READ#	220 READ#5; A\$,B\$	Reads strings from the specified file.
PRINT#	310 PRINT#2; A\$,C\$	Prints strings on a file.

## MATRICES

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

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

Continued on the next page.

## MATRICES, CONTINUED

<u>NAME</u>	<u>SAMPLE STATEMENT</u>	<u>PURPOSE</u>
MAT +	100 MAT C = A + B	Matrix addition; A and B must be the same size.
MAT -	110 MAT C = A - B	Matrix subtraction; A,B, and C must be the same size.
MAT*	120 MAT C = A * B	Matrix multiplication; No. columns in A must equal No. rows in B.
MAT =	130 MAT A = B	Establishes equality of two matrices; assigns values of B to A.
MAT TRN	140 MAT B = TRN (A)	Transposes an $m$ by $n$ matrix to an $n$ by $m$ matrix.
MAT INV	150 MAT C = INV (B)	Inverts a square matrix into a square matrix of the same size; matrix may be inverted into itself.

## FILES

- 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:*
- $$1 + \frac{\text{number of characters in the string}}{2} \text{ if there are an even number of characters,}$$
- or  $1 + \frac{\text{number of characters in the string} + 1}{2} \text{ if the number of characters is odd.}$
- This formula gives the required storage space in 2-character words.*

<u>FULL NAME</u>	<u>EXAMPLE (Abbreviation)</u>	<u>PURPOSE</u>
OPEN-	OPE-MYFILE,85	Opens file; assigns specified name and number of 64-word records.
KILL-	KIL-MYFILE	Deletes specified file.
FILES	1Ø 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#	12Ø 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).
	13Ø 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.
	14Ø PRINT# 2; A,B,END	Prints value on specified file; inserts an "end of file" marker immediately after the printed value.
	16Ø 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#	17Ø 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.

# INDEX, CONTINUED

- \* ...2-6
- + ...2-6
- ...2-5, 2-7
- / ...2-6
- < ...2-7
- <= ...2-7
- > ...2-7
- >= ...2-7
- ABS ...3-22
- ABS FUNCTION ...3-22
- ADDITION ...2-6
- ALT-MODE ...1-6
- AND ...2-9
- AND OPERATOR ...2-9
- APPEND ...3-12
- ARITHMETIC EVALUATION ...2-4
- ARITHMETIC OPERATOR ...2-6
- ASCII CODES ...8-7
- ASSIGNMENT OPERATIO ...2-5
- ATN FUNCTION ...3-23
- BREAK ...1-15, 2-41
- BREAK KEY ...1-15, 2-41
- BYE ...1-10, 2-35
- CATALOG ...3-15
- COMBINING LOGICAL OPERATORS ...7-6
- COMMANDS ...2-34
- CONNECTION TO THE COMPUTER ...1-8
- CONTROL C ...1-14
- CONTROL N ...6-7, E-1
- CONTROL O ...6-7, E-1
- CONVENTIONS IS THIS BOOK ...IV
- COPYING A MATRIX ...5-15
- CORRECTING MISTAKES ...1-6
- COS ...3-23
- COSINE FUNCTION ...3-23
- DATA ...2-21, 6-10
- DATA ON PAPER TAPE ...B-1
- DATA TYPE ...3-25
- DEF FN ...3-21
- DEFINING FUNCTIONS ...3-21
- DELETE ...3-13
- DELETING A LINE ...1-6
- DIAGNOSTIC MESSAGES ...1-13, D-1
- DIM ...5-2, 6-4
- DIMENSIONING A MATRIX ...5-2
- DIVISION ...2-6
- ECHO ...2-36
- END OF FILE ...4-14
- END STATEMENT ...2-28
- ENTERING DATA FROM TAPE ...B-1
- ESC ...1-6
- ESCAPE KEY ...CONTROL CHARACTERS
- EXP ...3-22
- EXP FUNCTION ...3-22
- EXPONENTIATION ...2-6
- EXPRESSION ...2-4
- FILE STRUCTURE ...4-10
- FILES ...4-1
- FILES STATEMENT ...4-6
- FOR ...2-18
- FOR...NEXT WITH STEP ...3-20
- FORMAL LANGUAGE ...1-2
- FUNCTIONS ...3-16
- GET ...3-10
- GET-\$ ...3-10
- GO TO ...2-16
- GO TO...OF ...2-16
- GOSUB ...3-17
- GOSUB...OF ...3-19
- HELLO- ...2-34
- HIERARCHY OF OPERATORS ...2-12
- HOW TO USE THIS BOOK ...VII
- ID CODE ...1-9
- IDENTITY MATRIX ...5-16
- IF END# STATEMENT ...4-9
- IF...THEN ...2-17
- INPUT ...2-24, 5-5, 6-6
- INPUT OF MATRIX ELEMENTS ...5-5
- INT ...3-22
- INT FUNCTION ...3-22
- INVERTING A MATRIX ...5-18
- KEY ...2-44
- KILL ...3-11
- KILL- ...4-5
- LEN COMMAND ...3-7
- LEN FUNCTION ...6-9
- LET ...2-14, 6-5
- LIBRARY ...3-14
- LINE NUMBER ...1-3
- LINEFEED ...1-9, 1-12
- LIST ...1-7, 2-38
- LISTING A FILE ...C-1
- LISTING A PROGRAM ...1-7
- LOG ...3-22
- LOG FUNCTION ...3-22
- LOG IN AND LOG OUT ...1-10
- LOGICAL OPERATIONS ...7-2
- MAT CON ...5-4
- MAT PRINT ...5-8
- MAT READ ...5-10
- MAT ZER ...5-3
- MATHEMATICAL FUNCTIONS ...3-22

Continued on the next page.



# INDEX, CONTINUED

<p> MATRICES ...5-1  MATRIX ADDITION ...5-11  MATRIX EQUALITY ...5-15  MATRIX MULTIPLICATION ...5-13  MATRIX SUBTRACTION ...5-12  MATRIX TRANSPOSITION ...1-17  MAX ...2-8  MEMORY ALLOCATION ...8-8  MIN ...2-8  MIN-MAX OPERATORS ...2-8  MISTAKES AND CORRECTIONS ...1-5  MISTAKES DURING LOG IN ...1-11  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  OPEN- ...4-4  OPENING AND CLOSING FILES ...4-4FF  OPERANDS ...1-3  OPERATOR ...2-5,FF  OR ...2-10  OR OPERATOR ...2-10  ORDER OF PRECEDENCE (OPERS.) ...2-12  PAGE FORMAT ...V  POINTER, RESETTING ...4-16, 4-18  PREFACE ...III  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  READING, WRITING MATRICES ...5-5,FF </p>	<p> REM ...2-15  RENUMBER ...2-40  RESTORE ...2-21  RETURN ...1-9  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  SQRT ...3-22  STANDARD FUNCTIONS ...3-22,FF  STATEMENTS ...2-13  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  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  TRANSPOSING A MATRIX ...5-17  TRIGONOMETRIC FUNCTIONS ...3-23  TYP FUNCTION ...3-26  UTILITY COMMANDS ...3-12,FF  X-ON, X-OFF ...B-1 </p>
--	---

Continued on the next page.





READER COMMENT SHEET  
A Guide To Time Shared BASIC

TEAR ALONG LINE

FROM

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

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

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

FOLD ON DOTTED LINES AS SHOWN ON OTHER SIDE AND TAPE

TAPE

TAPE

FOLD

FOLD

**BUSINESS REPLY MAIL**

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

SUPERVISOR, SOFTWARE PUBLICATIONS  
HEWLETT-PACKARD COMPANY  
11000 Wolfe Road  
Cupertino, California  
95014

FIRST CLASS  
PERMIT NO. 141  
CUPERTINO  
CALIFORNIA



TEAR ALONG LINE

FOLD

FOLD

HEWLETT  PACKARD

02000-90002