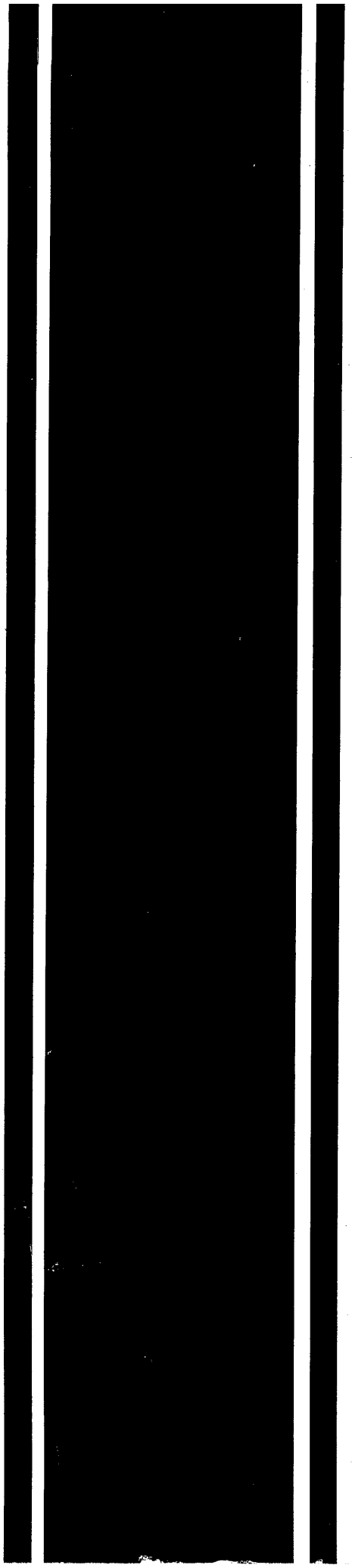


HEWLETT hp PACKARD

A GUIDE TO TIME SHARED BASIC

A GUIDE TO TIME SHARED BASIC

For Reference and Self-Instruction

Software Publications
Cupertino, California
95014

HP 02000-90002

August, 1969

First printing, Aug. 1969
Second printing, Feb. 1970

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.
line number PRINT X,Y	Black lower case italics indicates a general form, derived from BASIC syntax requirements (Sect. VIII). Red underlining indicates an essential part of a general form; each underlined item is a separate, essential element.
return *linefeed* *esc* *ctrl* *alt-mode* *break*	Represents the terminal keys: Return, Linefeed, Escape, Control, Alt-Mode, and Break.
Note: Both X and...	Mixed upper and lower case italics is used for notes.
LISTING A PROGRAM	Oversize black is used for page headings.
O	The letter "O"
Ø	Zeroes are slashed.

Please examine the sample on the next page.

PAGE FORMAT

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

EXAMPLES: Several sample statements or commands

GENERAL FORM:

(Each essential element underlined in red.)

PURPOSE

A clear and concise explanation of the purpose or function.

COMMENTS

A series of several items containing:

- Pertinent information
- Additional explanation or examples
- Helpful hints.

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

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

Section No. ___ Page No. ___ (Revision Date)

HOW TO USE THIS BOOK

If your purpose is:	Read:
Quickly acquiring a minimum working knowledge of Time Shared BASIC:	Sections I and II.
Acquiring a good working knowledge of Time Shared BASIC:	Sections I, II, III, IV, V, VI, in that order.
Learning the complete Time Shared BASIC system:	The entire book, in sequence.
Reference only:	1. Contents 2. The index, Appendix "F" 3. The index tabs to locate the appropriate section.

CONTENTS

CONTENTS CONTINUED

CONTENTS CONTINUED

CONTENTS CONTINUED

CONTENTS CONTINUED

CONTENTS CONTINUED

SECTION I
AN INTRODUCTION TO TIME SHARED BASIC

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

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

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

WHAT IS TIME SHARING?

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

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

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

COMMUNICATING WITH A COMPUTER

THE BASIC LANGUAGE

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

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

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

```
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
```

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

EXAMPLES OF BASIC STATEMENTS

This is a BASIC *statement:*

```
1Ø INPUT A,B,C,D,E
```

COMMENTS

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

A *statement* may also be called a line.

STATEMENT NUMBERS

Each BASIC statement begins with a *statement number* (in this example, 2Ø):

```
2Ø LET S=(A+B+C+D+E)/5
```

COMMENTS

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

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

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

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

Continued on the next page.

INSTRUCTIONS (STATEMENT TYPES)

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

```
3Ø PRINT S
```

COMMENTS

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

Continued on the next page.

OPERANDS

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

```
40 GO TO 10
```

COMMENTS

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

Continued on the next page.

A PROGRAM

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

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

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

COMMENTS

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

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

Continued on the next page.

THE FORMAT OF STATEMENTS

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

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

COMMENTS

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

Continued on the next page.

(Spot check)

Be sure you are familiar with these terms before continuing:

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

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

BEFORE GOING ON-LINE

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

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

PRESS RETURN AFTER EACH STATEMENT

The *return* key must be pressed after each statement.

Examples:

```
1Ø INPUT A,B,C,D,E return
2Ø LET S=(A+B+C+D+E)/5 return
3Ø PRINT S return
4Ø GO TO 1Ø return
5Ø END return
```

COMMENTS

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

BACKSPACE

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

Typing:	2Ø LR←ET S=1Ø *return*
is equivalent to typing:	2Ø LET S=1Ø *return*
And typing:	3Ø LET← ← ← PRINT S *return*
is equivalent to typing:	30 PRINT S *return*

COMMENTS

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

DELETING OR CHANGING A STATEMENT

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

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

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

Pressing the *esc* key deletes the statement being typed:

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

To delete statement 5 in the sequence:

```
5 LET S = Ø
1Ø INPUT A,B,C,D,E,
2Ø LET S = (A+B+C+D+E)/5
```

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

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

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

The old statement is replaced by the new one.

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

For example, typing:

```
5
```

or:

```
RUN
```

has no effect on the original statement 5.

LISTING A PROGRAM

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

NOTE: The program has already been entered.

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

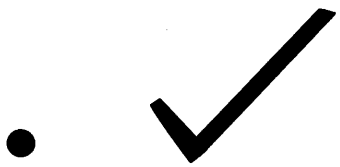
The computer skips three lines, separating the listing from previously printed information.

linefeed indicates that the listing is complete.

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

A list of the same sample program produces these lines:

```
LIST-30 return
linefeed
linefeed
linefeed
30 PRINT S
40 GO TO 10
50 END
linefeed
```



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

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

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

CONNECTION TO THE COMPUTER

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

IF YOUR TERMINAL EQUIPMENT IS A TELEPRINTER WITH

ACOUSTIC COUPLER AND TELEPHONE:

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

HALF DUPLEX COUPLER AND TELEPHONE

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

DATA SET:

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

DIRECT CONNECTION TO THE COMPUTER:

Turn the teleprinter control knob to the LINE position.

CHECKING THE CONNECTION

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

EXAMPLE: 3 *return*

The computer then responds with the message:

PLEASE LOG IN *return linefeed*

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

This step is optional

YOUR ID CODE AND PASSWORD

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

CONTROL CHARACTERS

Control characters are non-printing. They are represented with a superscript "C" to indicate that they are control characters. By using these non-printing characters, you may keep your password a secret. For example, on the teleprinter the password $SE^{C}C^{C}R^{C}E^{C}T$ prints as:

ST

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

SAMPLE LOG IN AND LOG OUT

H2ØØ is used as a sample identification code.

User H2ØØ for example, logs in by typing: *HELLO- is a command, not a statement. Commands are orders to the computer which are acted upon (executed) immediately. Unlike statements, commands do not require line numbers.*	HELLO-H2ØØ,*password* *return*
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.	ØØ1 MINUTES OF TERMINAL TIME

MISTAKES DURING LOG IN

If you make a mistake while logging in, the computer responds with a message informing you that something is wrong. For example, if user H2ØØ forgets the hyphen while entering the HELLO command:

HELLO H2ØØ,*password* *return*

the computer responds with the message:

ILLEGAL FORMAT *return* *linefeed*

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

If user H2ØØ enters his password incorrectly:

HELLO-H2ØØ,*password* *return*

the response is:

ILLEGAL ACCESS *return* *linefeed*

and the user tries again.

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

ENTERING THE SAMPLE PROGRAM

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

NOTE: Connection to the computer is made.

Log in: HELLO-H2ØØ,*password* *return*
OPERATOR'S MESSAGE TO USER

or

READY *return* *linefeed*

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

1Ø INPUT A,B,C,D,E *return*
linefeed
2Ø LET S = (A+B+C+D+E)/5 *return*
linefeed
3Ø PRINT S *return*
linefeed
4Ø GO TO 1Ø *return*
linefeed
5Ø END *return*
linefeed

Now the program is ready to run.

HOW TO OBTAIN A DIAGNOSTIC MESSAGE

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

If the user types:	3Ø PRIMT S *return* *NOTE: PRINT has been misspelled.*
The computer responds:	ERROR
The user then types in a colon (or any other character) followed by a *return*. This causes the diagnostic to be printed on the same line. The resulting output looks like this:	ERROR: *return* ERROR: NO STATEMENT TYPE FOUND *NOTE: PRIMT has not been recognized as a legal statement type, and the line was not accepted.*
To correct the statement, retype it in the proper form:	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 PRIMT S *return* ERROR *return* 3Ø PRINT S

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

RUNNING THE SAMPLE PROGRAM

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

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

The computer responds with four *linefeed's* indicating that the command is being executed.	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*.	? 95.6,87.3,8Ø.5,9Ø,82.8 *return*
The answer is printed:	87.24 *return* *linefeed*
NOTE: This program continues executing indefinitely, unless terminated by the user. To stop the program, type a C^C return (control "C") when more input is requested:	?-12.5,-5Ø.6,-32,45.6,6Ø *return* 2.1 *return* *linefeed* ? C^C *return*
The program is finished:	DONE
Log off:	BYE *return*
Time used is printed:	ØØ3 MINUTES OF TERMINAL TIME

STOPPING A PROGRAM: THE *break* KEY

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

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

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

Remember that: C^C *return*
and not *break* is used to terminate input loops (when the computer is expecting a number to be typed in).

COMMENTS

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

HOW THE PROGRAM WORKS

Line 1Ø 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 varables.

```
1Ø INPUT A,B,C,D,E
```

After line 1Ø is executed, the variables and their assigned values, typed in by the user, are stored. For example, using the values entered by the user in the previous example, this information is stored: A = -12.5; B = -50.6; C = -32; D = 45.6; E = 60

Line 2Ø 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:

```
2Ø LET S = (A+B+C+D+E)/5
```

Line 3Ø instructs the computer to output the value of S to user's terminal:

```
3Ø PRINT S
```

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

Line 4Ø tells the computer to go to line 1Ø and execute whatever instruction is there:

```
4Ø GO TO 1Ø
```

NOTE: A "loop" is formed by lines 1Ø to 4Ø. 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^C WHEN INPUT IS REQUESTED BY A "?" IS THE ONLY WAY TO BREAK AN INPUT LOOP WITHOUT DISCONNECTING THE TERMINAL DEVICE. Other, more controlled loops are explained later. Line 5Ø is not executed until the loop is broken by entering a C^C when input is requested.

Line 5Ø informs the computer that the program is finished:

```
5Ø END
```

SECTION II

THE ESSENTIALS OF BASIC

HOW TO READ THIS SECTION

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

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

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

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

TERM: NUMBER

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

TERM: E NOTATION

DEFINED IN TSB AS:	A means of expressing numbers having more than six decimal digits, in the form of a decimal number raised to some power of 10.
EXAMPLES:	1.00000E+06 is equal to 1,000,000 and is read "1 times 10 to the sixth power ($1x10^6$).
	1.02000E+04 is equal to 10,200
	1.02000E-04 is equal to .000102

COMMENTS

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

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

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

TERM: SIMPLE VARIABLE

DEFINED IN TSB AS:	A letter (from A to Z); or a letter immediately followed by a number (from Ø to 9).	
EXAMPLES:	AØ	B
	M5	C2
	Z9	D

COMMENTS

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

```
1Ø LET M5 = 96.7
```

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

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

TERM: EXPRESSION

DEFINED IN TSB AS:	A combination of variables, constants and operators which has a numeric value.
EXAMPLES:	(P + 5)/27 (where P has previously been assigned a numeric value.)
	Q - (N + 4) (where Q and N have previously been assigned numeric values.)

TERM: ARITHMETIC EVALUATION

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

THE ASSIGNMENT OPERATOR

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 *variable* = *expression*

PURPOSE

Assigns an arithmetic or logical value to a variable.

COMMENTS

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

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: ↑ * / + -

EXAMPLES:

```
4Ø LET N1 = X-5
5Ø LET C2 = N↑3
6Ø LET A = (B-C)/4
7Ø LET X = ((P↑2)-(Y*X))/N+Q
```

PURPOSE

Represents an arithmetic operation, as:

exponentiate:	↑
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:

↑
* /
+ -

with ↑ 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:	1ØØ IF A=B THEN 9ØØ
	11Ø IF A+B >C THEN 91Ø
	12Ø IF A+B < C+E THEN 92Ø
	13Ø IF C>= D*E THEN 93Ø
	14Ø IF C9<= G*H THEN 94Ø
	15Ø IF P2#C9 THEN 95Ø
	16Ø IF J <> K THEN 95Ø

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 Ø 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:     1Ø LET A=A9=P2=P5=C2=X=7.5
              2Ø LET B5=D8=Q1=Q4=Y=B=12.Ø
               :
              8Ø PRINT (A MIN 1Ø)
              9Ø LET B=(A MIN 1Ø)+1ØØ
             1ØØ IF (A MIN B5) > (C2 MIN D8) THEN 1Ø
             11Ø PRINT (X MAX Y)
             12Ø IF (A9 MAX B) <= 5 THEN 15Ø
```

PURPOSE

Selects the larger or smaller value of two expressions.

COMMENTS

In the examples above, statement 11Ø selects and prints the larger value: since X = 7.5 and Y = 12.Ø, the value of Y is printed. The evaluation is made first, then the statement type (PRINT) is executed.

THE AND OPERATOR

SYMBOL: AND

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


PURPOSE

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

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

COMMENTS

The numeric value of "true" is 1, of "false" is 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:
```
1ØØ IF A>1 OR B<5 THEN 5ØØ
11Ø PRINT C OR D
12Ø LET D = X OR Y
13Ø IF (X AND Y) OR (P AND Q) THEN 6ØØ
```

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" = Ø.

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:	3Ø LET X = Y = Ø 35 IF NOT A THEN 3ØØ 45 IF (NOT C) AND A THEN 4ØØ 55 LET B5 = NOT P 65 PRINT NOT (X AND Y) 7Ø IF NOT (A=B) THEN 5ØØ

PURPOSE

Logically evaluates the complement of a given expression.

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

COMMENTS

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

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

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

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

ORDER OF PRECEDENCE OF EXECUTION

The order of performing operations is:

↑	*highest precedence*
NOT	
* /	
+ -	
MIN MAX	
Relational Operators	
AND	
OR	*lowest precedence*

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 + (6x7)

7/14*2/5 is evaluated as: $\frac{(7/14)x2}{5}$

A MIN B MAX C MIN D is evaluated as:
((A MIN B) MAX C) MIN D

Parentheses override the order of precedence in all cases.

STATEMENTS

Be sure you know the difference between statements and commands.

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

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

Here are some examples mentioned in Section I:

Statements	Commands
LET	HELLO
PRINT	BYE
INPUT	LIST

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

THE ASSIGNMENT STATEMENT

EXAMPLES:

```
1Ø LET A = 5.Ø2
2Ø X = Y7 = Z = Ø
3Ø B9 = 5* (X↑2)
4Ø LET D = (3*C2↑N)/(A*(N/2))
```

GENERAL FORM:

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

or

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

PURPOSE

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

COMMENTS

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

The assignment statement must contain:

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

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

REM

EXAMPLES:

```
10 REM--THIS IS AN EXAMPLE
20 REM: OF REM STATEMENTS
30 REM-----/////*****!!!!!
40 REM. STATEMENTS ARE NOT EXECUTED BY TSB
```

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

PURPOSE

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

COMMENTS

Must be preceeded 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 erros.

IF...THEN

SAMPLE PROGRAM:

```
1Ø LET N = 1Ø
2Ø READ X
3Ø IF X <=N THEN 6Ø
4Ø PRINT "X IS 1Ø OR OVER"
5Ø GO TO 8Ø
6Ø PRINT "X IS LESS THAN 1Ø"
7Ø GO TO 2Ø
8Ø 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<=1Ø, the message in statement 6Ø is printed.

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

If the specified condition for transfer is not true, then the program will continue executing in sequence. In the example above, if X>=1Ø, the message in statement 4Ø will be printed.

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

FOR...NEXT

EXAMPLES:

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

Sample Program - Variable Number Of Loops

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

GENERAL FORM:

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

or

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

(Statements to be repeated)

⋮

statement number NEXT *simple variable*

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

FOR...,,NEXT, CONTINUED

PURPOSE

Allows repetition of a group of statements within a program.

COMMENTS

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

How the loop works:

The simple variable is assigned the value of the *initial value*; the value of the simple variable is increased by 1 (or by the *step value*) each time the loop executes. When the value of the *simple variable* passes the *final value*, control is transferred to the statement following the "NEXT" statement.

STEP and *step value* are optional.

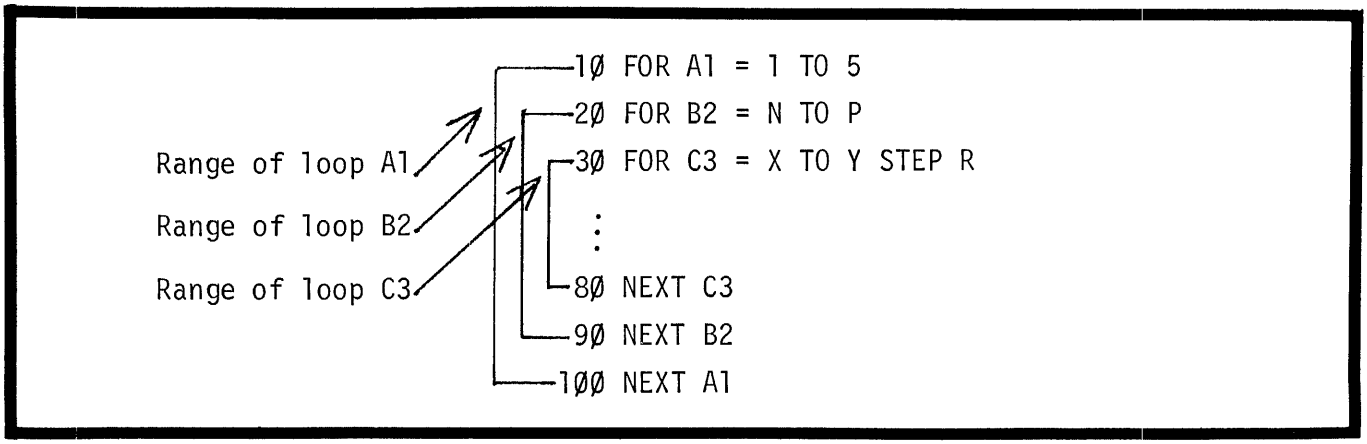
For further details on the STEP feature, see "FOR...NEXT with STEP" in Section III.

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

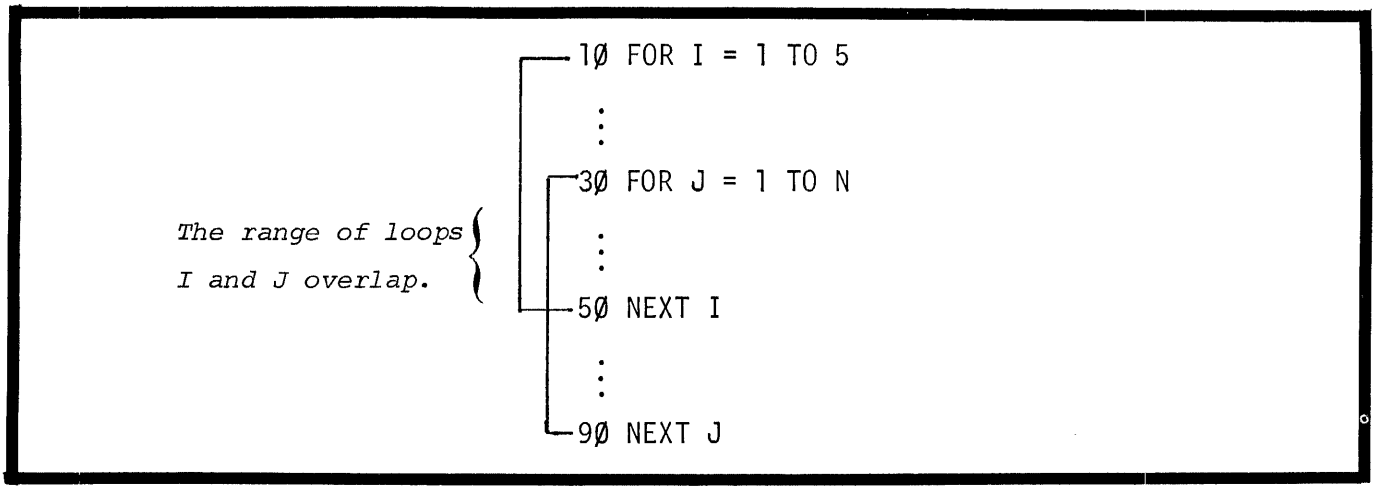
NESTING FOR...NEXT LOOPS

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

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



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



READ, DATA AND RESTORE

Sample Program using READ and DATA

```
15  FOR I=1 TO 5
2Ø  READ A
4Ø  LET X=A↑2
45  PRINT A;" SQUARED =";X
5Ø  NEXT I
55  DATA 5.24,6.75,30.8,72.65,89.72
6Ø  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

```
 2Ø FOR I=1 TO 5
 3Ø READ A
 4Ø LET X=A↑2
 5Ø PRINT A; "SQUARED =";X
 6Ø NEXT I
 8Ø RESTORE
1ØØ FOR J=1 TO 5
11Ø READ B
12Ø LET Y=B↑4
13Ø PRINT B; "TO THE FOURTH POWER =";Y
14Ø NEXT J
15Ø DATA 5.24,6.75,30.8,72.65,89.72
16Ø 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
1Ø  INPUT A
2Ø  INPUT A1,B2,C3,ZØ,Z9,E5
3Ø  PRINT "WHAT VALUE SHOULD BE ASSIGNED TO R";
4Ø  INPUT R
5Ø  PRINT A;A1;B2;C3;ZØ;Z9;E5;"R=";R
6Ø  NEXT M
7Ø  END
```

---------- RESULTS ----------

```
RUN
?1 return
?2,3,4,5,6,7 return
WHAT VALUE SHOULD BE ASSIGNED TO R?27 return

   1    2    3    4    5    6    7    R=27
?1.5 return
?2.5,3.5,4.5,6.,7.2 return
??8.1 return    ?? indicates that more input is expected
WHAT VALUE SHOULD BE ASSIGNED TO R?-99
  1.5     2.5     3.5     4.5     6     7.2
  8.1     R=-99

DONE
```

GENERAL FORM:

statement number INPUT *variable* , *variable* ,...

PURPOSE

Assigns a value input from the teleprinter to a variable.

Continued on next page.

INPUT CONTINUED

COMMENTS

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

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

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

and

```
20 INPUT X
```

each cause a single "?" to be printed. Note that the "?" generated by statement 10 requires seven input items, separated by commas, while the "?" generated by statement 20 requires only a single input item.

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

Relevant Diagnostics:

```
  ?  indicates that input is required.
 ??  indicates that more input is needed to satisfy an INPUT statement.
???  indicates that TSB cannot decipher your input.
ENTRA INPUT-WARNING ONLY indicates that a) extra input was
      entered;  b) it has been disregarded; and  c) the program
      is continuing execution.
```

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

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
1111122222333334444455555566666
DONE
```

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

Continued on next page.

PRINT, CONTINUED

GENERAL FORM:

statement number PRINT *expression* , *expression* , ...

or

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

or

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

or

statement number PRINT *any combination of text and/or expressions*

or

statement number PRINT

PURPOSE

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

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

COMMENTS

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

END AND STOP

EXAMPLES:

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

GENERAL FORM:

any statement number STOP

any statement number END

Highest statement number in program END

PURPOSE

Terminates execution of the program and returns control to TSB.

COMMENTS

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

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

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

SAMPLE PROGRAM

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

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

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

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

SAMPLE PROGRAM


```
1Ø  REMARK: "REMARK" OR "REM" IS USED TO INDICATE REMARKS OR COMMENTS
2Ø  REMARK:  THE USER WANTS TO INCLUDE IN THE TEXT OF HIS PROGRAM.
3Ø  REM:  THE COMPUTER LISTS AND PUNCHES THE "REM" LINE, BUT DOES NOT
4Ø  REM:  EXECUTE IT.
5Ø  REM:  "PRINT" USED ALONE GENERATES A "RETURN" "LINEFEED"
6Ø  PRINT
7Ø  PRINT "THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY."
8Ø  PRINT
9Ø  PRINT "IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS."
1ØØ PRINT
11Ø PRINT "PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY."
12Ø PRINT
13Ø PRINT
14Ø REM:  FIRST, ALL VARIABLES USED IN THE PROGRAM ARE INITIALIZED
15Ø REM:  TO ZERO (THEIR VALUE IS SET AT ZERO.)
16Ø LET A=N=R1=S=Ø
18Ø REM:  NOW THE USER WILL BE GIVEN A CHANCE TO SPECIFY HOW MANY
19Ø REM:  NUMBERS HE WANTS TO AVERAGE.
2ØØ PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE";
21Ø INPUT N
22Ø PRINT
23Ø PRINT "O.K., TYPE IN ONE OF THE ";N;"NUMBERS AFTER EACH QUES. MARK."
24Ø PRINT "DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER."
25Ø PRINT
26Ø PRINT "NOW, LET'S BEGIN"
27Ø PRINT
28Ø PRINT
3ØØ REM:  "N" IS NOW USED TO SET UP A "FOR-NEXT" LOOP WHICH WILL READ
31Ø REM:  1 TO "N" NUMBERS AND KEEP A RUNNING TOTAL.
32Ø FOR I=1 TO N
33Ø INPUT A
34Ø LET S=S+A
35Ø NEXT I
36Ø REM:  "I" IS A VARIABLE USED AS A COUNTER FOR THE NUMBER OF TIMES
```

SAMPLE PROGRAM CONTINUED


```
37Ø REM:  THE TASK SPECIFIED IN THE "FOR-NEXT" LOOP IS PERFORMED.
38Ø REM:  "I" INCREASES BY 1 EACH TIME THE LOOP IS EXECUTED.
39Ø REM:  "A" IS THE VARIABLE USED TO REPRESENT THE NUMBER TO BE
4ØØ REM:  AVERAGED.  THE VALUE OF "A" CHANGES EACH TIME THE
41Ø REM:  USER INPUTS A NUMBER.
42Ø REM:  "S" WAS CHOSEN AS THE VARIABLE TO REPRESENT THE SUM
43Ø REM:  OF ALL NUMBERS TO BE AVERAGED.
44Ø REM:  AFTER THE LOOP IS EXECUTED "N" TIMES, THE PROGRAM CONTINUES.
46Ø REM:  A SUMMARY IS PRINTED FOR THE USER.
47Ø PRINT
48Ø PRINT
49Ø PRINT N; "NUMBERS WERE INPUT."
5ØØ PRINT
51Ø PRINT "THEIR SUM IS:";S
52Ø PRINT
53Ø PRINT "THEIR AVERAGE IS:";S/N
54Ø PRINT
55Ø PRINT
57Ø REM:  NOW THE USER WILL BE GIVEN THE OPTION OF QUITTING OR
58Ø REM:  RESTARTING THE PROGRAM.
59Ø PRINT "DO YOU WANT TO AVERAGE ANOTHER GROUP OF NUMBERS?"
6ØØ PRINT
61Ø PRINT "TYPE 1 IF YES, Ø IF NO"
62Ø PRINT "BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER."
63Ø PRINT
64Ø PRINT "YOUR REPLY";
65Ø INPUT R1
66Ø IF R1=1 THEN 12Ø
67Ø REM:  THE FOLLOWING LINES ANTICIPATE A MISTAKE IN THE REPLY.
68Ø IF R1#Ø THEN 7ØØ
69Ø GO TO 72Ø
7ØØ PRINT "TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø IF NO."
71Ø GO TO 64Ø
72Ø END
```

RUNNING THE SAMPLE PROGRAM


```
RUN return

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY.
IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS.
PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.
HOW MANY NUMBERS DO YOU WANT TO AVERAGE?
O.K.,TYPE IN ONE OF THE 5 NUMBERS AFTER EACH QUES. MARK.
DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER.
NOW, LET'S BEGIN
? 99 return
? 87.6 return
? 92.7 return
? 79.5 return
? 84 return
5  NUMBERS WERE INPUT.
THEIR SUM IS:  442.8
THEIR AVERAGE IS:  88.56
DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?
TYPE 1 IF YES, Ø IF NO
BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.
YOUR REPLY?  2 return
TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø IF NO.
YOUR REPLY?  1 return
HOW MANY NUMBERS DO YOU WISH TO AVERAGE?  C^C return
DONE
```

COMMANDS

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

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

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

HEL-H2ØØ,SE^{C}C^{C}R^{C}E^{C}T *return*

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

HELLO -

EXAMPLE:	HELLO-DØØ7,POS^{C}T *return* *or* HEL-DØØ7,POS^{C}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.

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

BYE

EXAMPLE: BYE *return*
ØØ9 MINUTES OF TERMINAL TIME

GENERAL FORM: BYE

RYF

PURPOSE

The command used to log out of the TSB system.

COMMENTS

Causes the amount of terminal time used to be printed.

Breaks a telephone connection to the computer.

ECHO-

EXAMPLES:	ECHO-OFF *return* ECHO-ON *return*
GENERAL FORM:	ECHO-ON *or* ECHO-OFF

PURPOSE

Allows use of half duplex terminal.

COMMENTS

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

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

May be abbreviated to its first three letters.

RUN

EXAMPLE:	RUN *return* *or* RUN- 3ØØ *return*
GENERAL FORM:	RUN RUN- *statement number*

PURPOSE

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

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

COMMENTS

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

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

C^{C} *return*

LIST

EXAMPLE:	LIST *return* *or* LIST-1ØØ *return*
GENERAL FORM:	LIST LIST- *statement number*

PURPOSE

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

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

COMMENTS

A listing may be stopped by pressing the *break* key.

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

May be abbreviated to its first three letters.

SCRATCH

EXAMPLE:	SCRATCH *return* *or* SCR *return*
GENERAL FORM:	SCRATCH *or* SCR

PURPOSE

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

COMMENTS

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

RENUMBER

RENUMBER

EXAMPLES:

RENUMBER *return*

REN *return*

REN-1ØØ

REN-1Ø, 1 *return*

REN-2Ø, 5Ø *return*

GENERAL FORM:

REN

or

REN-*number assigned to first statement*

or

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

PURPOSE

Renumbers statements in a Program.

COMMENTS

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

If no first statement number is specified, renumbering begins at statement 1Ø, in intervals of 1Ø.

If no interval is specified, the new numbers are spaced at intervals of 1Ø, from the beginning statement.

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

BREAK

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

PURPOSE

Terminates a program being run.

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

COMMENTS

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

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

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

C^C *return*

which produces the DONE message.

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

PUNCH

EXAMPLES: PUNCH *return*
PUN *return*
PUN-65 *return*
PUN-5 *return*

GENERAL FORM: PUNCH
or
PUN
or
PUN-*statement number at which PUNCHing is to begin*

PURPOSE

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

COMMENTS

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

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

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

TAPE

EXAMPLES:	TAPE *return* TAP *return*
GENERAL FORM:	TAPE *or* TAP

PURPOSE

Informs the computer that following input is from paper tape.

COMMENTS

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

TSB responds to the TAPE command with a *linefeed*.

KEY

EXAMPLES: KEY *return*

GENERAL FORM: KEY

PURPOSE

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

COMMENTS

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

TIME

EXAMPLE: TIME *return*
CONSOLE TIME = Ø MINUTES. TOTAL TIME = ØØ MINUTES

GENERAL FORM: TIME

PURPOSE

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

COMMENTS

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

SECTION III

ADVANCED BASIC

This section describes more sophisticated capabilities of BASIC.

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

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

TERM: ROUTINE

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

PURPOSE

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

COMMENTS

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

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

Examples of routines and subroutines are given in this section.

TERM: ARRAY (OR MATRIX)

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

COMMENTS

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

Arrays may have one or two dimensions. For example,

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 row and column 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

```
1ØØ LET Y = SQR(X)
```

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

TERM: WORD

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

COMMENTS

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

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

TERM: RECORD

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

COMMENTS

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

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 *return*
LEN *return*
ØØØØ WORDS

GENERAL FORM: LEN *return*

PURPOSE

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

COMMENTS

Each user has a working "space" of approximately 51ØØ two character words. LEN is a useful check on total program length when writing long programs.

NAME-

EXAMPLE:	NAME-PROG.1 *return* NAME-**GO** *return* NAM-ADDER *return* NAM-MYPROG *return*
GENERAL FORM:	NAME-*Program name of 1 to 6 characters* *or* NAM-*Program name of 1 to 6 characters*

PURPOSE

Assigns a name to the program currently being accessed from the teleprinter.

COMMENTS

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

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

SAVE

EXAMPLES:	SAVE *return* SAV *return*
GENERAL FORM:	SAVE *or* SAV

PURPOSE

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

COMMENTS

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

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

KILL-SAMPLE *return* *linefeed*	(Deletes the stored version.)
NAME-SAMPLE *return* *linefeed*	(Names the current program)
SAVE *return* *linefeed*	(Saves the current program, named SAMPLE.)

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

GET- AND GET- $

EXAMPLES:
GET-PROGRM *return*
GET-MYPROG *return*
GET-$PUBLIC *return*
GET-$NAMES *return*

GENERAL FORM:
GET- *name of a program in user's library*
GET-$ *name of system library program*

PURPOSE

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

GET-$ 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 *return* KIL-EXMPLE *return* KIL-FILE1Ø *return*
GENERAL FORM:	KILL-*program or file to be deleted* *or* KIL-*program or file to be deleted*

PURPOSE

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

COMMENTS

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

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

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

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

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

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

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

APPEND-

EXAMPLES:	APPEND-MYPROG *return* APP-MYPROG *return* APPEND-$PUBLIC *return* APP-$SYSLIB *return*
GENERAL FORM:	APPEND-*program name* *or* APP-*program name* *or* APP-$*system library program name*

PURPOSE

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

COMMENTS

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

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

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

DELETE-

EXAMPLES: DELETE-27 *return*
DEL-27, 5Ø *return*

GENERAL FORM: DEL-*statement number at which deletion starts*
or
DEL-*statement no. at which deletion starts* , *statement no. at which deletion ends*

PURPOSE

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

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

COMMENTS

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

Deleted statements are not recoverable.

LIBRARY

EXAMPLES: LIBRARY *return*
LIB *return*

BINOPO Ø594	CDETER Ø7Ø6	CSHFLO 1598	CURFIT 1618	DIFFEQ Ø133	DIVID 1367
FNCTS Ø652	GEOMEN Ø199	IN4 544Ø	IN5 544Ø	INVHIL Ø25Ø	LINFIT Ø492
ROMINT Ø299	SQE Ø2Ø9	STAT11 Ø568	TAB 2Ø98	YELLOW Ø227	Z123 Ø413

GENERAL FORM: LIBRARY
or
LIB

PURPOSE

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

COMMENTS

Public library programs are available to users; typing:

GET-$ *program name return*

retrieves the specified program.

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

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

LIBRARY listings may be terminated with the *break* key.

CATALOG

EXAMPLES: CAT *return*

CATALOG *return*

PROG1 ØØ24 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:

```
5Ø READ A2
6Ø IF A2<1ØØ THEN 8Ø
7Ø GOSUB 4ØØ
 :
38Ø STOP (STOP, END, or GO TO's frequently precede
          the first statement of a subroutine, to
          prevent accidental entry.)
39Ø REM--THIS SUBROUTINE ASKS FOR A 1 OR Ø REPLY.
4ØØ PRINT "A2 IS>1ØØ"
41Ø PRINT "DO YOU WANT TO CONTINUE";
42Ø INPUT N
43Ø IF N #Ø THEN 45Ø
44Ø LET A2 = Ø
45Ø RETURN
 :
6ØØ 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 transferrred 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:
```
2Ø GOSUB 3 OF 1ØØ,2ØØ,3ØØ,4ØØ,5ØØ
6Ø GOSUB N+1 OF 2ØØ,21Ø,22Ø
7Ø GOSUB N OF 8Ø,18Ø,28Ø,38Ø,48Ø,58Ø
```

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 2Ø, above transfers control to the subroutine beginning with statement 3ØØ. The *expression* specifies which statement in the sequence of five statements is used as the starting one in the subroutine.

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

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

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

NESTING GOSUBS

EXAMPLES:

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


PURPOSE

Allows selective use of subroutines within subroutines.

COMMENTS

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

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

FOR...NEXT WITH STEP

EXAMPLES:

```
2Ø FOR I5 = 1 TO 2Ø STEP 2
4Ø FOR N2 = Ø TO -1Ø STEP -2
8Ø FOR P = 1 TO N STEP R
9Ø 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,

```
1ØØ FOR N = 1 TO 2 STEP .Ø1
```

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 4Ø above.

DEF FN

EXAMPLE:

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

GENERAL FORM:

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

PURPOSE

Allows the programmer to define functions.

COMMENTS

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

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

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

causes infinite looping.

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

GENERAL MATHEMATICAL FUNCTIONS

EXAMPLES:

```
642 PRINT EXP(N); ABS(N)
652 IF RND (Ø)>=.5 THEN 9ØØ
662 IF INT (R) # 5 THEN 91Ø
672 PRINT SQR (X); LOG (X)
```

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

PURPOSE

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

ABS (*expression*) the absolute value of the expression;

EXP (*expression*) the constant *e* raised to the power of the expression value (in statement 642 above, *e*↑N)

INT (*expression*) the largest integer ≤ the expression;

LOG (*expression*) the logarithm of the positively valued expression to the base *e*;

RND (*expression*) a random number between 1 and 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:

```
5ØØ PRINT SIN(X); COS(Y)
51Ø PRINT 3*SIN(B); TAN (C2)
52Ø PRINT ATN (22.3)
53Ø IF SIN (A2) <1 THEN 8ØØ
54Ø IF SIN (B3) = 1 AND SIN(X) <1 THEN 9Ø
```


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:

```
5ØØ IF SGN (X)> -1 THEN 8ØØ
51Ø LET Y = SGN(X)
52Ø PRINT TAB (5); A2; TAB (2Ø)"TEXT"
53Ø PRINT TAB (N),X,Y,Z2
54Ø 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 (Ø to 71).

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

THE TYP FUNCTION

EXAMPLES:	8ØØ IF TYP (3) = 2 THEN 1ØØØ 85Ø PRINT TYP (N) 9ØØ IF TYP (R) # X THEN 12ØØ
GENERAL FORM:	TYP may be used as an expression or as part of an expression; the function form is: TYP (*file number formula*)

PURPOSE

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

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

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

COMMENTS

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

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.

TERM: FILE

DEFINED IN TSB AS:

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

Smaller divisions within a file are called records and words.

File structure is explained later in this section.

TERM: END OF RECORD

DEFINED IN TSB AS:

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

TERM: END OF FILE

DEFINED IN TSB AS:

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

TERM: SERIAL AND RANDOM ACCESS

DEFINED IN TSB AS:

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

OPEN-

EXAMPLES:

OPEN-FILE27, 85 *return*
OPEN-SAMPLE, 128 *return*
OPEN-**FI**, 1Ø *return*

GENERAL FORM:

OPEN- *1 to 6 character file name* , *number of 64-word records in file*
OPE- *1 to 6 character file name* , *number of 64-word records in file*

PURPOSE

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

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

COMMENTS

The minimum number of records per file is 1.

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

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

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

** 90 to 128*

KILL-

EXAMPLE:	KILL-NAMEXX *return* KIL-EXMPLE *return* KIL-FILE1Ø *return*
GENERAL FORM:	KILL-*file to be deleted* KIL-*file to be deleted*

PURPOSE

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

COMMENTS

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

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

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

FILES

EXAMPLE: 1Ø MATH, SCORE, AND, SQRT, NAME5, $DATA

GENERAL FORM:

statement number FILES *maximum of 8 file names, separated by commas.*

PURPOSE

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

COMMENTS

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

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

```
1ØØ PRINT #2;A
```

prints the value of A on the file named SCORE.

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

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

PRINT

EXAMPLES:

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

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

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

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

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

READ

EXAMPLES:

```
65 READ#5; A,B,C
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.

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 preceeding pages, you may PRINT and READ information on files.

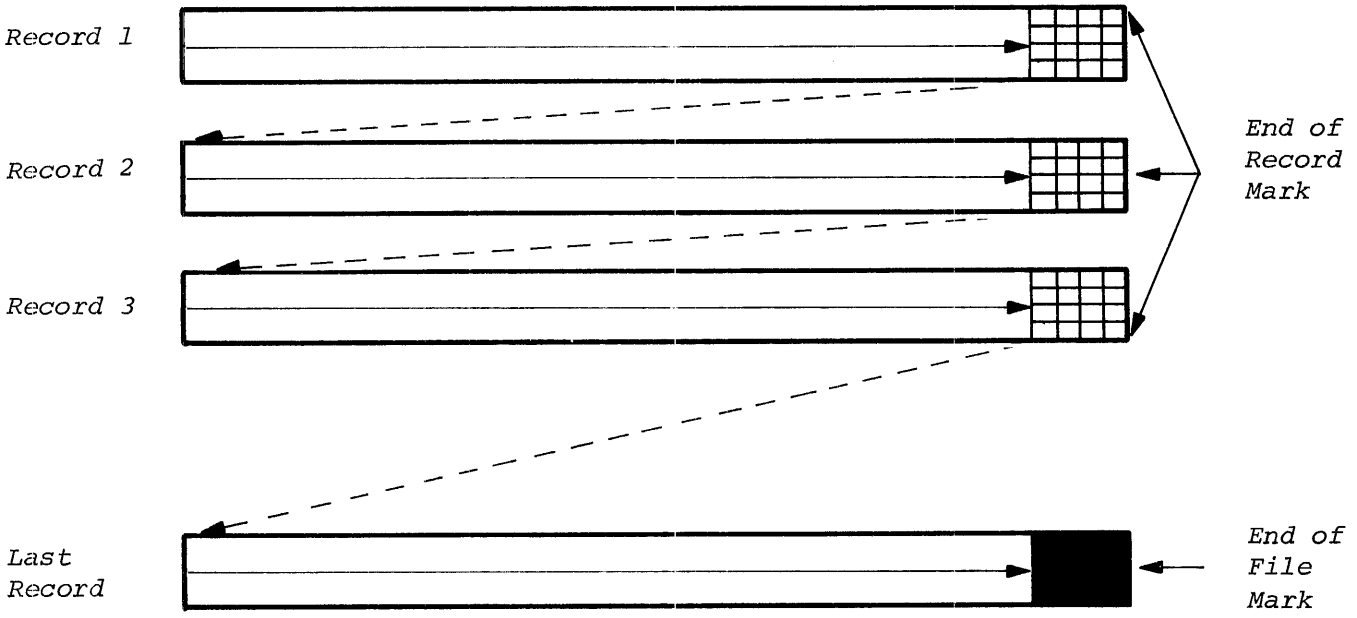
There are disadvantages to this method:

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.

STRUCTURE AND STORAGE PATTERN



COMMENTS

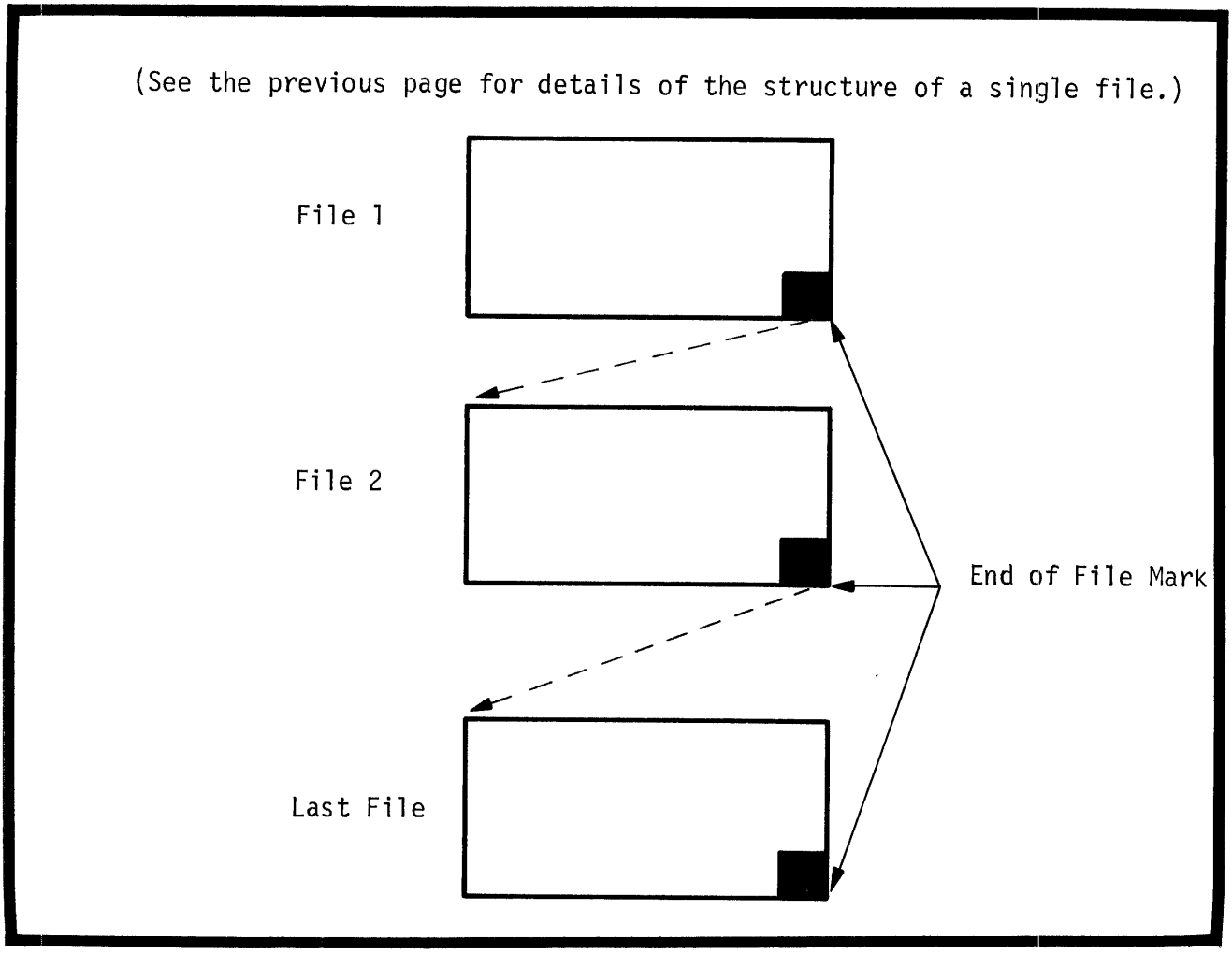
——▶ and – – –▶ indicate the order of record use when record numbers are not specified in PRINT and READ statements.

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



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

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

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

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

FILE STRUCTURE-SUMMARY

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

Each word = 2 teleprinter characters.

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

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

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

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.

PRINT ...,END

EXAMPLES:

```
 95 PRINT #N ; A,B2, END
1ØØ PRINT #(X+1); R3, S1, "TEXT", END
11Ø PRINT #2; G5; H$, P, END
```

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

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

PRINT#...,...

EXAMPLES:
```
165 PRINT #N,X; G2,H,I, "TEXT"
17Ø PRINT #1,3; X, Y4, Z
175 PRINT #(N+1), (X+2);F,P5
```

GENERAL FORM:

statement number PRINT# *file number formula* , *record number formula* ; *print list*

PURPOSE

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

COMMENTS

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

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

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

PRINT TO RESET A POINTER

EXAMPLES:

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

GENERAL FORM:

statement number PRINT# *file number formula* , *record number formula*

PURPOSE

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

Erases the contents of the specified record.

COMMENTS

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

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

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

READ#...,.

EXAMPLES:

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

GENERAL FORM:

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

PURPOSE

Reads data from a specified record of a file.

COMMENTS

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

Non-integers are rounded to the nearest integer value.

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

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

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

READ TO RESET A POINTER

EXAMPLES:

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

GENERAL FORM:

statement number READ# *file formula number*

or

statement number READ# *file number formula* , *record number formula*

PURPOSE

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

COMMENTS

A specified record may be reread without resetting the pointer.

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

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

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

SECTION V
MATRICES

This section explains matrix manipulation. It is intended to show the matrix capabilities of 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.

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

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

COMMENTS

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

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

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

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

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

MAT...ZER

EXAMPLES:

```
3Ø5 MAT A = ZER
31Ø MAT Z = ZER (N)
315 MAT X = ZER (3Ø, 1Ø)
32Ø 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 Ø; 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 Ø has a logical value of "false", MAT...ZER is useful in logical initialization.

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

MAT...CON

EXAMPLES:

```
2Ø5 MAT C = CON
21Ø MAT A = CON (N,N)
22Ø MAT Z = CON (5,2Ø)
23Ø MAT Y = CON (5Ø)
```

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 2Ø5.

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

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

INPUT

EXAMPLES:

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

GENERAL FORM:

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

or

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

PURPOSE

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

COMMENTS

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

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

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

MAT INPUT

EXAMPLES:

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

GENERAL FORM:

statement number MAT INPUT *matrix variable*

or

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

or

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

PURPOSE

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

COMMENTS

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

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

Elements being input must be separated by commas.

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

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

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

PRINTING MATRICES

EXAMPLES:

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

GENERAL FORM:

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

or

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

PURPOSE

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

COMMENTS

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

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

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

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

MAT PRINT

EXAMPLES:

```
5ØØ MAT PRINT A
5Ø5 MAT PRINT A;
515 MAT PRINT A,B,C
52Ø 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 5Ø5. Normal separation (",") prints 5 elements per row.

READ

EXAMPLES:

```
90Ø READ A(6)
91Ø READ A(9,9)
92Ø READ C(X); P$; R7
93Ø READ C(X,Y)
94Ø READ Z(X,Y), P(R2, S5), X(4)
```

GENERAL FORM:

statement number READ *matrix variable* (*expression*)

or

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

PURPOSE

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

COMMENTS

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

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

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

MAT READ

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

GENERAL FORM:

statement number MAT READ *matrix variable*

or

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

or

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

PURPOSE

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

COMMENTS

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

MATRIX ADDITION

EXAMPLES:

```
31Ø MAT C = B + A
32Ø MAT X = X + Y
33Ø MAT P = N + M
```

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* + *matrix variable*

PURPOSE

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

COMMENTS

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

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

MATRIX SUBTRACTION

EXAMPLES:

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

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* - *matrix variable*

PURPOSE

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

COMMENTS

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

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

MATRIX MULTIPLICATION

EXAMPLES:

```
93Ø MAT Z = B * C
94Ø MAT X = A * A
95Ø MAT C = Z * B
```

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

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

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

SCALAR MULTIPLICATION

EXAMPLES:

```
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.

COPYING A MATRIX

EXAMPLES:

```
4Ø5 MAT B = A
41Ø MAT X = Y
42Ø MAT Z = B
```

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

IDENTITY MATRIX

EXAMPLES:

```
2Ø5 MAT A = IDN
21Ø MAT B = IDN (3,3)
215 MAT Z = IDN (Q5, Q5)
22Ø 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 Ø'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  Ø  Ø
Ø  1  Ø
Ø  Ø  1
```

MATRIX TRANSPOSITION

EXAMPLES:

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

GENERAL FORM:

statement number MAT *matrix variable* = TRN (*matrix variable*)

PURPOSE

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

COMMENTS

Sample transposition:

Original	Transposed
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.

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.

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 READ

EXAMPLES:

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

GENERAL FORM:

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

or

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

or

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

or

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

PURPOSE

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

COMMENTS

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

Remember that a maximum of 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."

TERM: STRING

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

COMMENTS

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

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

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

TERM: SUBSTRING

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

COMMENTS

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

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

prints the substring:

```
BCDEF
```

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

Using single subscript, as:

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

prints the substring:

```
CDEFGH
```

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

Both strings and substrings may be used with relational operators.

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

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

THE STRING DIM STATEMENT

EXAMPLES:

```
35 DIM A$ (72), B$(6Ø)
4Ø DIM Z$ (1Ø)
45 DIM N$ (2), R(5,5), P$(8)
```

GENERAL FORM :

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

PURPOSE

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

COMMENTS

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

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

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

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

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

THE STRING ASSIGNMENT STATEMENT

NOTE: These strings have been mentioned in a DIM statement

EXAMPLES:

```
2ØØ LET A$ = "TEXT OF STRING"
21Ø B$ = "*** TEXT !!!"
22Ø LET C$ = A$(1,4)
23Ø D$ = B$(4)
24Ø F$(3,8)=N$
```

GENERAL FORM:

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

or

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

or

statement number string variable = " *string value* "

or

statement number string variable = *string or substring variable*

PURPOSE

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

COMMENTS

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

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

If the assigned value is longer than the string length, the assigned value is truncated at the appropriate point.

THE STRING INPUT STATEMENT

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

EXAMPLES:

```
5Ø INPUT R$
55 INPUT A$,B$, C9, D1Ø
6Ø INPUT A$(1,5)
65 INPUT B$(3)
```

GENERAL FORM:

statement number INPUT *string or substring variable*...

PURPOSE

Allows string values to be entered from the teleprinter.

COMMENTS

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

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

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

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

PRINTING STRINGS

EXAMPLES:

```
1Ø5 PRINT A$
11Ø PRINT A$, B$, Z$
115 PRINT C$(8) "END OF STRING" B3, B4, A9
12Ø PRINT C$(1,7)
13Ø PRINT "THE TOTAL IS:";X5
```

GENERAL FORM:

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

PURPOSE

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

COMMENTS

String and numeric values may be mixed in a PRINT statement (example statements 115 and 13Ø above).

Specifying only one substring parameter causes the entire substring to be printed. For instance, if C$ = "WHAT IS YOUR NAME?", example statement 12Ø prints:

```
WHAT IS
```

while statement 115 prints

```
YOUR NAME?END OF STRING 642
```

Numeric and string values may be "packed" in PRINT statements without using a ";", as in example statement 115.

0^C and N^C generate a *return* and *linefeed* respectively when printed as string characters.

READING STRINGS

EXAMPLES:

```
3ØØ READ C$
3Ø5 READ X$, Y$, Z$
31Ø READ Y$(5), A,B,C5,N$
315 READ Y$(1,4)
```

GENERAL FORM:

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

PURPOSE

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

COMMENTS

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

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

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

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

STRING IF

EXAMPLES:

```
34Ø IF C$<D$ THEN 8ØØ
35Ø IF C$>=D$ THEN 9ØØ
36Ø IF C$#D$ THEN 1ØØØ
37Ø IF N$(3,5)<R$(9) THEN 5ØØ
38Ø IF A$(1Ø)="END" THEN 4ØØ
```

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$) #5 THEN 6ØØ
5Ø9 IF LEN (P$) = 5 THEN 6Ø9
519 IF LEN (P$) = 5 OR LEN (P$) = 1Ø THEN 1Ø
529 LET X$(LEN(X$)+1) = "ADDITIONAL SUBSTRING"
 :
6ØØ STOP
6Ø9 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:

```
5ØØ DATA "NOW IS THE TIME."
51Ø DATA "HOW", "ARE", "YOU,"
52Ø 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 (example statement 520 above).

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

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

PRINTING STRINGS ON FILES

EXAMPLES:

```
35Ø PRINT #5; "THIS IS A STRING."
355 PRINT #8; C$, B$, X$, Y$, D$
36Ø 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 36Ø 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.

READING STRINGS FROM FILES

EXAMPLES:

```
71Ø READ #1, 5; A$, B$
715 READ #2; C$, A1, B2, X
72Ø READ #3,6; C$(5),X$(4,7),Y$
73Ø READ #N,P; C$, V$(2,7), R$(9)
```

GENERAL FORM:

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

or

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

PURPOSE

Reads string and substring values from a file.

COMMENTS

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

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

SECTION VII

LOGICAL OPERATIONS

LOGICAL VALUES AND NUMERIC VALUES

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

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

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

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

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

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

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

or

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

RELATIONAL OPERATORS

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

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

EXAMPLES:

```
15Ø IF B=7 THEN 6ØØ
2ØØ IF A9#27.65 THEN 7ØØ
3ØØ IF (Z/1Ø)>=Ø THEN 8ØØ
```

When a statement is evaluated, if the "IF" condition is currently true (for example, in statement 15Ø, if B = 7), then control is transferred to the specified statement.

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

Continued on the next page.

RELATIONAL OPERATORS CONTINUED

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

EXAMPLES:

```
61Ø LET X=27
615 PRINT X=27
62Ø PRINT X#27
63Ø 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, Ø if X does not equal 27." There are only two logical alternatives; 1 is used to represent "true", and Ø "false".

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

BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.

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

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

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

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

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

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

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


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

Note that the criteria for determining the logical values are:

true = any nonzero expression value
false = an expression value of Ø.

The numeric value 1 or Ø is assigned accordingly.

SOME EXAMPLES

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

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

EXAMPLES:

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

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

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

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

SECTION VIII
FOR THE PROFESSIONAL

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

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

SYNTAX REQUIREMENTS OF TSB

LEGEND

::= "is defined as..."

| "or"

< > enclose an element of Time Shared BASIC

LANGUAGE RULES

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

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

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>	::=	Ø|1|2|3|4|5|6|7|8|9
<exponent part>	::=	E<integer>|E+<integer>|E-integer *(see rule 1)*
<literal string>	::=	"<character string>"
<character string>	::=	<character>|<character string><character>
<character>	::=	any ASCII character except null, line feed, return, x-off, alt-mode, escape, ←, " , and rubout
<variable>	::=	<simple variable>|<subscripted variable>
<simple variable>	::=	<letter>|<letter><digit>
<letter>	::=	A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<subscripted variable>	::=	<letter>(<sublist>)
<sublist>	::=	<expression>|<expression>,<expression>
<string variable>	::=	<string simple variable>|<string simple variable>(<sublist>)
<string simple variable>	::=	<letter>$
<expression>	::=	<conjunction>|<expression>OR<conjunction>
<conjunction>	::=	<relation>|<conjunction>AND<relation>
<relation>	::=	<minmax>|<minmax><relational operator><minmax>
<minmax>	::=	<sum>|<minmax>MIN<sum>|<minmax>MAX<sum>
<sum>	::=	<term>|<sum>+<term>|<sum>-<term>
<term>	::=	<subterm>|<term>*<subterm>|<term>/<subterm>
<subterm>	::=	<denial>|<signed factor>

SYNTAX REQUIREMENTS OF TSB continued

SYNTAX REQUIREMENTS OF TSB, CONTINUED

<denial>	::=	<factor>|NOT<factor>
<signed factor>	::=	+<factor>|-<factor>
<factor>	::=	<primary>|<factor>↑<primary>
<primary>	::=	<variable>|<number>|<functional>|<parameter> *(rule 2)*| (<expression>)
<relational operator>	::=	<|<=|=|#|<>|>=|>
<parameter>	::=	<letter>|<letter><digit>
<functional>	::=	<function identifier>(<expression>)| <pre-defined function>(<expression>)| LEN (<string simple variable>)
<function identifier>	::=	FN <letter>
<pre-defined function>	::=	SIN|COS|TAN|ATN|EXP|LOG|ABS|SQR|INT|RND|SGN|TYP
<source string>	::=	<string variable>|<literal string>
<destination string>	::=	<string variable>
<file reference>	::=	#<file formula>|#<file formula>,<record formula>
<file formula>	::=	<expression>
<record formula>	::=	<expression>
<array identifier>	::=	<letter>
<sequence number>	::=	<integer> *(see rule 3)*
<program statement>	::=	<sequence number><BASIC statement>carriage return
<BASIC statement>	::=	<LET statement>|<IF statement>|<GOTO statement>| <GOSUB statement>|<RETURN statement>|<FOR statement>| <NEXT statement>|<STOP statement>|<END statement>| <DATA statement>|<READ statement>|<INPUT statement>| <PRINT statement>|<RESTORE statement>|<DIM statement>| <DEF statement>|<FILES statement>|<REM statement>| <MAT statement>
<LET statement>	::=	LET <leftpart><expression>| LET <destination string>=<source string>| <leftpart><expression>| <destination string>=<source string>
<leftpart>	::=	<variable>=|<leftpart><variable>=
<IF statement>	::=	IF<decision expression>THEN<sequence number>| IF END #<file formula>THEN<sequence number>
<decision expression>	::=	<expression>| <comparison string 1><relational operator> <comparison string 2>
<comparison string 1>	::=	<string variable>
<comparison string 2>	::=	<string variable>|<literal string>

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

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
	Ø	D	X
	1	E	Y
	2	F	Z
	3	G	[
	4	H	\
			]
			↑ *Highest*

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

MEMORY ALLOCATION BY A USER

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

SOME EXAMPLES OF USER-DETERMINED ALLOCATION*

a) Introduction of each simple, string, or matrix variable uses 4 words.

b) A 9 word stack is reserved for GOSUB's.

c) 6 X (maximum level of FOR...NEXT loop nesting)

d) Each file name mentioned in a FILES statement reserves 64 words for buffer space.

e) An approximate estimate of space required for a program is:

11 words per BASIC statement
+2X(number of matrix elements dimensioned)
+1/2X(number of string characters used)

* *This is variable "system overhead"; it is not included in word counts produced by the LEN command.*

HOW TO PREPARE A PAPER TAPE OFF-LINE

To prepare a paper tape for input:

1. Turn teleprinter control knob to "LOCAL".

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

3. Press the "HERE IS" key; or press @^{C} (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 @^{C} several times to put trailing holes on the tape.

6. Press the "OFF" button on the tape punch.

COMMENTS

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

Pressing the "B.SP." (backspace) botton on the tape punch, then the "RUBOUT" key will physically delete the previous character from a paper tape.

THE X-ON, X-OFF FEATURE

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

Data is punched on paper tape in this format:
line of data items separated by commas *x-off* *return* *linefeed*

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

COMMENTS

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

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

SAMPLE PROGRAM LISTING A FILE

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

```
9ØØ2 REM LISTS THE CONTENTS OF A FILE.
9ØØ3 LET N=1
9ØØ4 DIM A$[72]
9ØØ5 LET I=Ø
9ØØ6 LET I=I+1
9ØØ7 READ #N,I
9ØØ8 PRINT "**FILE#"N
9ØØ9 PRINT "RECORD"I
9Ø1Ø GOTO TYP(-N) OF 9Ø11,9Ø14,9Ø17,9ØØ6
9Ø11 READ #N;A
9Ø12 PRINT A;
9Ø13 GOTO 9Ø1Ø
9Ø14 READ #N;A$
9Ø15 PRINT A$
9Ø16 GOTO 9Ø1Ø
9Ø17 PRINT "**END OF FILE"N"**"
9Ø18 LET N=N+1
9Ø19 GOTO 9ØØ5
9Ø2Ø 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.

```
9ØØ3 PRINT "* TRUE ANNUAL INTEREST RATE *"
9ØØ4 PRINT
9Ø1Ø PRINT "THIS PROGRAM CALCULATES THE TRUE ANNUAL INTEREST RATE"
9Ø2Ø PRINT "ON AN INSTALLMENT LOAN"
9Ø3Ø PRINT
9Ø4Ø PRINT
9Ø5Ø PRINT "IF YOU NEED INSTRUCTIONS TYPE 1, OTHERWISE TYPE Ø: "
9Ø6Ø INPUT X
9Ø7Ø IF X=Ø THEN 912Ø
9Ø8Ø PRINT "TO USE THIS PROGRAM IT IS NECESSARY FOR YOU TO SUPPLY"
9Ø9Ø PRINT "FOUR VARIABLES: A = AMOUNT OF LOAN (IN $), P = AMOUNT OF"
91ØØ PRINT "PAYMENT ($), N = THE TOTAL NUMBER OF PAYMENTS DUE, AND K = NUMBER"
911Ø PRINT "OF PAYMENTS DUE IN ONE YEAR."
9115 PRINT
912Ø PRINT "WHAT ARE A,P,N,K ";
913Ø INPUT A,P,N,K
914Ø PRINT
915Ø IF N=1 THEN 955Ø
916Ø IF P*N >= A THEN 922Ø
917Ø PRINT
918Ø PRINT "THATS NOT REASONABLE; THE PAYMENTS ADD UP TO LESS THAN THE AMOUNT"
919Ø PRINT "OWED.  TRY AGAIN."
92ØØ PRINT
921Ø GOTO 912Ø
922Ø LET R=Ø
923Ø LET D=1ØØ
924Ø GOSUB 933Ø
925Ø IF P=P1 THEN 943Ø
```

SAMPLE PROGRAM- INTEREST RATES, CONTINUED

```
926Ø IF P>P1 THEN 929Ø
927Ø LET R=R-D
928Ø GOTO 93ØØ
929Ø LET R=R+D
93ØØ LET D=D/2
931Ø IF D<.ØØØ1 THEN 943Ø
932Ø GOTO 924Ø
933Ø LET R1=R/(1ØØ*K)
934Ø LET Q=1+R1
935Ø IF N*LOG(Q)/LOG(1Ø) <= 75 THEN 938Ø
936Ø LET P1=A*R1
937Ø RETURN
938Ø IF Q>1 THEN 941Ø
939Ø LET P1=A/N
94ØØ RETURN
941Ø LET P1=A*Q↑N*R1/(Q↑N-1)
942Ø RETURN
943Ø LET R=.Ø1*INT(.5+1ØØ*R)
944Ø IF R<199.5 THEN 95ØØ
945Ø PRINT
946Ø PRINT "ARE YOU SURE THE DATA IS CORRECT?  THE INTEREST RATE IS OVER"
947Ø PRINT "2ØØ PERCENT.  TRY AGAIN."
948Ø PRINT
949Ø GOTO 912Ø
95ØØ PRINT "THE TRUE ANNUAL INTEREST RATE = ";R
951Ø PRINT
952Ø PRINT
953Ø PRINT "ANOTHER CASE?? TYPE 'N' TO QUIT, 'Y' TO TRY AGAIN";
9532 INPUT Q$
9534 IF Q$="N" THEN 9999
954Ø GOTO 912Ø
955Ø LET R=(P/A-1)*K
956Ø LET R=1ØØ*R
957Ø GOTO 943Ø
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 9Ø8Ø - 9Ø82 in which only a "NO" reply is significant.

```
9ØØ2 PRINT "THIS PROGRAM DETERMINES THE DAY OF THE WEEK"
9ØØ3 PRINT "ON WHICH A GIVEN DATE FALLS."
9ØØ4 DIM A$[5]
9ØØ5 LET W=W1=Ø
9ØØ6 DIM F[12],L[12]
9ØØ7 MAT READ F
9ØØ8 MAT READ L
9ØØ9 GOTO 9Ø76
9Ø1Ø IF Y<1 THEN 9Ø27
9Ø11 IF M>12 THEN 9Ø27
9Ø12 IF M<1 THEN 9Ø27
9Ø13 IF D<1 THEN 9Ø27
9Ø14 IF Y>1752 THEN 9Ø34
9Ø15 IF Y<1582 THEN 9Ø29
9Ø16 IF Y=1752 THEN 9Ø23
9Ø17 IF Y=1582 THEN 9Ø19
9Ø18 GOTO 9Ø31
9Ø19 IF M<1Ø THEN 9Ø29
9Ø2Ø IF M>1Ø THEN 9Ø31
9Ø21 IF D<15 THEN 9Ø29
9Ø22 GOTO 9Ø31
9Ø23 IF M<9 THEN 9Ø31
9Ø24 IF M>9 THEN 9Ø34
9Ø25 IF D<14 THEN 9Ø31
9Ø26 GOTO 9Ø34
9Ø27 PRINT "UNACCEPTABLE DATA -- TRY AGAIN."
9Ø28 GOTO 9Ø78
9Ø29 LET G1=Ø
9Ø3Ø GOTO 9Ø32
```

```
9Ø31 LET G1=1
9Ø32 LET J1=1
9Ø33 GOTO 9Ø36
9Ø34 LET G1=1
9Ø35 LET J1=Ø
9Ø36 IF J1 <> 1 THEN 9Ø54
9Ø37 LET L1=Ø
9Ø38 LET A=Y+INT((Y+3)/4)
9Ø39 IF Y <> INT(Y/4)*4 THEN 9Ø44
9Ø4Ø LET L1=1
9Ø41 IF M<3 THEN 9Ø44
9Ø42 LET L=1
9Ø43 GOTO 9Ø45
9Ø44 LET L=Ø
9Ø45 LET Z=A+D+L+F[M]+5
9Ø46 LET Z=Z-INT(Z/7)*7
9Ø47 LET Q=L[M]
9Ø48 IF M <> 2 THEN 9Ø5Ø
9Ø49 LET Q=Q+L1
9Ø5Ø IF D>Q THEN 9Ø27
9Ø51 PRINT "OLD STYLE CALENDAR: ";
9Ø52 LET W=1
9Ø53 GOSUB 9Ø89
9Ø54 IF G1 <> 1 THEN 9Ø8Ø
9Ø55 LET L1=Ø
9Ø56 LET Y=Y-4ØØ*INT(Y/4ØØ)
9Ø57 LET A=Y+INT((Y+3)/4)-INT((Y-1)/1ØØ)
9Ø58 IF Y <> INT(Y/4)*4 THEN 9Ø65
9Ø59 IF Y=Ø THEN 9Ø61
9Ø6Ø IF Y=1ØØ*INT(Y/1ØØ) THEN 9Ø65
9Ø61 LET L1=1
9Ø62 IF M<3 THEN 9Ø65
9Ø63 LET L=1
```

```
9Ø64 GOTO 9Ø66
9Ø65 LET L=Ø
9Ø66 LET Z=A+D+L+F[M]
9Ø67 LET Z=Z-INT(Z/7)*7
9Ø68 LET Q=L[M]
9Ø69 IF M <> 2 THEN 9Ø71
9Ø7Ø LET Q=Q+1
9Ø71 IF D>Q THEN 9Ø27
9Ø72 IF W=Ø THEN 9Ø74
9Ø73 PRINT "NEW STYLE CALENDAR: ";
9Ø74 GOSUB 9Ø89
9Ø75 GOTO 9Ø8Ø
9Ø76 PRINT "ENTER MONTH NUMBER, DATE, AND YEAR."
9Ø77 PRINT
9Ø78 INPUT M,D,Y
9Ø79 IF W1=Ø THEN 9Ø1Ø
9Ø8Ø PRINT "IS THERE ANOTHER DATE YOU WANT TO KNOW";
9Ø81 INPUT A$
9Ø82 IF A$="NO" THEN 9999
9Ø83 PRINT
9Ø84 PRINT "ENTER DATE: ";
9Ø85 LET W=Ø
9Ø86 LET W1=1
9Ø87 INPUT M,D,Y
9Ø88 GOTO 9Ø1Ø
9Ø89 GOTO Z+1 OF 9Ø9Ø,9Ø92,9Ø94,9Ø96,9Ø98,91ØØ,91Ø2
9Ø9Ø PRINT "FRIDAY"
9Ø91 RETURN
9Ø92 PRINT "SATURDAY"
9Ø93 RETURN
9Ø94 PRINT "SUNDAY"
9Ø95 RETURN
```

```
9Ø96 PRINT "MONDAY"
9Ø97 RETURN
9Ø98 PRINT "TUESDAY"
9Ø99 RETURN
91ØØ PRINT "WEDNESDAY"
91Ø1 RETURN
91Ø2 PRINT "THURSDAY"
91Ø3 RETURN
99ØØ DATA Ø,3,3,6,1,4,6,2,5,Ø,3,5
99Ø1 DATA 31,28,31,3Ø,31,3Ø,31,31,3Ø,31,3Ø,31
9999 END
```

SAMPLE PROGRAM

H-P FOOTBALL

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

```
1Ø PRINT "WELCOME TO THE CUPERTINO DIVISION FOOTBALL CHAMPIONSHIP GAME".
2Ø PRINT "THE DIVISION PLAYOFF IS BETWEEN THE HEWLETT HORNETS"
3Ø PRINT "AND THE PACKARD PANTHERS."
4Ø PRINT
5Ø PRINT "WE'LL NEED SOME HELP.  WILL YOU CALL THE PLAYS FOR HEWLETT";
6Ø DIM W$[12]
7Ø INPUT W$
8Ø PRINT "FINE.  THE COMPUTER WILL CALL THE PLAYS FOR PACKARD."
9Ø PRINT
1ØØ PRINT "OK, COACH- FIRST, LET'S GET ACQUAINTED. WHAT'S YOUR NAME";
11Ø INPUT W$
12Ø PRINT "OK "W$", TYPE ONE OF THE PLAY NUMBERS FOLLOWED BY A RETURN"
13Ø PRINT "THE PLAY NUMBERS ARE:"
14Ø PRINT "1 = SIMPLE RUN; 2 = TRICKY RUN; 3 = SHORT PASS;"
15Ø PRINT "4 = LONG PASS; 5 = PUNT; 6 = QUICK KICK; 7 = PLACE KICK."
16Ø PRINT
17Ø LET P1=51
18Ø LET Q1=Ø
19Ø LET T=Ø
2ØØ LET S[1]=Ø
21Ø LET S[3]=Ø
22Ø PRINT "TOSS OF THE COIN-TYPE A NUMBER FROM 1 TO 3ØØ (THEN RETURN)."
23Ø INPUT Z1
24Ø FOR I=1 TO Z1
25Ø LET X=RND(Q1)
26Ø NEXT I
27Ø IF RND(Q1)>1/2 THEN 3ØØ
28Ø PRINT "PACKARD WON THE TOSS."
```

H-P FOOTBALL

```
29Ø GOTO 158Ø
3ØØ PRINT "HEWLETT WON THE TOSS."
31Ø PRINT "HEWLETT'S BALL ON ITS OWN 2Ø."
32Ø LET P=1
33Ø LET X=2Ø
34Ø LET X1=2Ø
35Ø LET D=1
36Ø GOTO 172Ø
37Ø PRINT "CALL IT, "W$".";
38Ø INPUT Z
39Ø LET R=RND(Q1)
4ØØ LET R=R*(.97+P*.Ø3)
41Ø LET T=T+1
42Ø IF T<P1 THEN 54Ø
43Ø PRINT P1-1; "PLAYS HAVE BEEN MADE.  DO YOU WISH TO STOP NOW?"
44Ø PRINT "TYPE 1 FOR YES, Ø FOR NO.""YOUR REPLY";
45Ø INPUT D1
46Ø GOTO D1+1 OF 52Ø, 49Ø
47Ø GOTO 43Ø
48Ø PRINT
49Ø PRINT "END OF GAME ***"
5ØØ PRINT "FINAL SCORE: HEWLETT";S[3];" PACKARD";S[1]
51Ø STOP
52Ø PRINT "2Ø MORE PLAYS WILL BE ALLOWED."
53Ø LET P1=P1+2Ø
54Ø LET R1=RND(Q1)
55Ø LET F=Ø
56Ø IF Z>4 THEN 62Ø
57Ø IF Z=1 THEN 74Ø
58Ø IF Z=2 THEN 79Ø
59Ø PRINT "PASS PLAY-----";
6ØØ IF Z=3 THEN 86Ø
61Ø GOTO 1Ø1Ø
62Ø REM PUNT
63Ø LET Y=INT(1ØØ*(R-.5)↑3+35)
64Ø IF Z=7 THEN 233Ø
65Ø IF D=4 THEN 67Ø
66Ø LET Y=INT(Y*1.3)
67Ø PRINT "PUNT GOOD FOR"Y "YARDS"
68Ø IF D<4 THEN 72Ø
69Ø LET Y1=INT(R1↑2*2Ø)+(1-P)*INT(R↑2*3Ø)
7ØØ PRINT "RUN BACK FOR "Y1 "YARDS"
71Ø LET Y=Y-Y1
72Ø LET F=-1
73Ø GOTO 118Ø
74Ø REM SIMPLE RUN
75Ø PRINT "RUNNING PLAY--";
76Ø LET Y=INT(24*(R-.5)↑3+3)
77Ø IF R1<.Ø5 THEN 83Ø
```

H-P FOOTBALL

```
78Ø GOTO 1Ø7Ø
79Ø REM TRICKY RUN
8ØØ PRINT "RUNNING PLAY--";
81Ø LET Y=INT(2Ø*R-5)
82Ø IF R1>.1 THEN 1Ø7Ø
83Ø LET F=-1
84Ø PRINT "*** FUMBLE AFTER ";
85Ø GOTO 1Ø7Ø
86Ø REM SHORT PASS
87Ø IF R<.Ø5 THEN 92Ø
88Ø IF R<.15 THEN 98Ø
89Ø IF R<.55 THEN 95Ø
9ØØ PRINT "COMPLETE. ";
91Ø GOTO 1Ø7Ø
92Ø PRINT "INTERCEPTED. "
93Ø LET F=-1
94Ø GOTO 118Ø
95Ø PRINT "INCOMPLETE. ";
96Ø LET Y=Ø
97Ø GOTO 1Ø7Ø
98Ø PRINT "PASSER TACKLED. ";
99Ø LET Y=-INT(1Ø*R1)
1ØØØ GOTO 1Ø7Ø
1Ø1Ø REM LONG PASS
1Ø2Ø LET Y=INT(16Ø*(R1-.5)↑3+3Ø)
1Ø3Ø IF R<.1 THEN 92Ø
1Ø4Ø IF R<.25 THEN 98Ø
1Ø5Ø IF R<.7 THEN 95Ø
1Ø6Ø GOTO 9ØØ
1Ø7Ø REM RESULT OF PLAY
1Ø8Ø LET X2=X+P*Y
1Ø9Ø IF X2 >= 1ØØ THEN 126Ø
11ØØ IF X2 <= Ø THEN 176Ø
1111Ø IF Y<Ø THEN 115Ø
1112Ø IF Y=Ø THEN 117Ø
1113Ø PRINT "GAIN OF"Y"YARDS"
1114Ø GOTO 118Ø
1115Ø PRINT "LOSS OF"-Y"YARDS"
1116Ø GOTO 118Ø
1117Ø PRINT "NO GAIN"
1118Ø LET X=X+P*Y
1119Ø IF X <= Ø THEN 176Ø
12ØØ IF X>5Ø THEN 123Ø
121Ø PRINT "BALL ON HEWLETT'S"X"YARD LINE. ";
122Ø GOTO 144Ø
123Ø IF X >= 1ØØ THEN 126Ø
124Ø PRINT "BALL ON PACKARD'S"1ØØ-X;"YARD LINE, ";
125Ø GOTO 144Ø
126Ø IF P<Ø THEN 134Ø
```

H-P FOOTBALL

```
127Ø IF F<Ø THEN 132Ø
128Ø PRINT "TOUCHDOWN!!!"
129Ø LET P=-1
13ØØ GOSUB 225Ø
131Ø GOTO 158Ø
132Ø PRINT "TOUCHBACK FOR PACKARD."
133Ø GOTO 158Ø
134Ø IF F<Ø THEN 141Ø
135Ø PRINT "SAFETY!"
136Ø GOSUB 221Ø
137Ø PRINT "TOUCHDOWN FOR HEWLETT!!!"
138Ø LET X=4Ø
139Ø LET P=1
14ØØ GOTO 161Ø
141Ø PRINT "TOUCHDOWN HEWLETT!!!"
142Ø GOSUB 225Ø
143Ø GOTO 158Ø
144Ø LET D=D+1
145Ø IF F >= Ø THEN 154Ø
146Ø IF P>Ø THEN 151Ø
147Ø PRINT
148Ø PRINT "HEWLETT'S BALL."
149Ø LET P=1
15ØØ GOTO 161Ø
151Ø PRINT
152Ø PRINT "PACKARD'S BALL."
153Ø GOTO 16ØØ
154Ø IF P*(X-X1) >= 1Ø THEN 161Ø
155Ø IF D<5 THEN 172Ø
156Ø IF P<Ø THEN 147Ø
157Ø GOTO 151Ø
158Ø LET X=8Ø
159Ø PRINT "PACKARD'S BALL ON ITS OWN 2Ø."
16ØØ LET P=-1
161Ø LET D=1
162Ø PRINT "FIRST DOWN."
163Ø IF P<Ø THEN 167Ø
164Ø IF X<9Ø THEN 17ØØ
165Ø LET X1=9Ø
166Ø GOTO 173Ø
167Ø IF X>1Ø THEN 17ØØ
168Ø LET X1=1Ø
169Ø GOTO 173Ø
17ØØ LET X1=X
171Ø GOTO 173Ø
172Ø PRINT "DOWN"D;"AND"1Ø+P*(S1-S);"YARDS TO GO."
173Ø PRINT
174Ø IF P>Ø THEN 37Ø
175Ø GOTO 194Ø
```

H-P FOOTBALL

```
176Ø IF F<Ø THEN 188Ø
177Ø IF P>Ø THEN 182Ø
178Ø PRINT "TOUCHDOWN!!!"
179Ø LET P=1
18ØØ GOSUB 225Ø
181Ø GOTO 31Ø
182Ø PRINT "SAFETY!!"
183Ø GOSUB 221Ø
184Ø PRINT "PACKARD GETS THE BALL ON ITS OWN 4Ø."
185Ø LET X=6Ø
186Ø LET P=-1
187Ø GOTO 161Ø
188Ø IF P>Ø THEN 191Ø
189Ø PRINT "TOUCHBACK FOR HEWLETT."
19ØØ GOTO 31Ø
191Ø PRINT "TOUCHDOWN PACKARD!!!"
192Ø GOSUB 225Ø
193Ø GOTO 31Ø
195Ø LET P=-1
196Ø IF D>1 THEN 2Ø2Ø
197Ø IF RND(Q1)>1/3 THEN 2ØØØ
198Ø LET Z=3
199Ø GOTO 219Ø
2ØØØ LET Z=1
2Ø1Ø GOTO 219Ø
2Ø2Ø IF D<4 THEN 2Ø9Ø
2Ø3Ø IF X <= 3Ø THEN 2Ø6Ø
2Ø4Ø LET Z=5
2Ø5Ø GOTO 219Ø
2Ø6Ø IF 1Ø+X-X1<3 THEN 197Ø
2Ø7Ø LET Z=7
2Ø8Ø GOTO 219Ø
2Ø9Ø IF 1Ø+X-X1<5 THEN 197Ø
21ØØ IF X>X1 THEN 216Ø
211Ø IF RND(Q1)>1/2 THEN 214Ø
212Ø LET Z=2
213Ø GOTO 219Ø
214Ø LET Z=4
215Ø GOTO 219Ø
216Ø IF RND(Q1)>1/4 THEN 218Ø
217Ø GOTO 212Ø
218Ø GOTO 214Ø
219Ø GOTO 39Ø
22ØØ REM KEEP SCORE
221Ø LET S[2-P]=S[2-P]+7
222Ø PRINT "SCORE: HEWLETT "S[3];"PACKARD"S[1]
223Ø PRINT
224Ø RETURN
225Ø IF RND(Q1)>.8 THEN 229Ø
```

H-P FOOTBALL

```
226Ø PRINT "KICK IS GOOD"
227Ø LET S[2-P]=S[2-P]+7
228Ø GOTO 222Ø
229Ø PRINT "KICK IS OFF TO THE SIDE"
23ØØ LET S[2-P]=S[2-P]+6
231Ø GOTO 221Ø
232Ø PRINT
233Ø REM FIELD GOAL
234Ø PRINT "PLACE KICK"
235Ø LET F=-1
236Ø IF R>.15 THEN 239Ø
237Ø PRINT "KICK IS BLOCKED***"
238Ø GOTO 118Ø
239Ø IF P<Ø THEN 25ØØ
24ØØ IF X+Y >= 11Ø THEN 246Ø
241Ø IF X+Y<8Ø THEN 244Ø
242Ø PRINT "KICK IS OFF TO THE SIDE"
243Ø GOTO 132Ø
244Ø PRINT "KICK IS OFF TO THE SIDE"
245Ø GOTO 118Ø
246Ø PRINT "FIELD GOAL!!!"
247Ø LET S[3]=S[3]+3
248Ø GOSUB 222Ø
249Ø GOTO 158Ø
25ØØ IF X-Y <= -1Ø THEN 254Ø
251Ø IF X-Y>2Ø THEN 244Ø
252Ø PRINT "KICK IS OFF TO THE SIDE."
253Ø GOTO 189Ø
254Ø PRINT "FIELD GOAL!!!"
255Ø LET S[1]=S[1]+3
256Ø GOSUB 222Ø
257Ø GOTO 31Ø
258Ø 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

NEGATIVE STRING LENGTH

NEXT WITHOUT MATCHING FOR

NO '*' AFTER RIGHT PARENTHESIS

NO LEGAL BINARY OPERATOR FOUND

NO CLOSING QUOTE

NON-CONTIGUOUS STRING CREATED

NON-EXISTENT FILE REQUESTED

NO STATEMENT TYPE FOUND

OUT OF DATA

OUT OF STORAGE

OVERFLOW - WARNING ONLY

OVER/UNDERFLOWS - WARNING ONLY

PARAMETER NOT STRING VARIABLE

READ-ONLY FILES:

REDIMENSIONED ARRAY TOO LARGE

RETURN WITH NO PRIOR GOSUB

SAME FOR-VARIABLE NESTED

SECOND FILES STATEMENT

72 CHARACTERS MAX FOR STRING

SIGN WITHOUT NUMBER

STATEMENT HAS EXCESSIVE LENGTH

STRING OVERFLOW

STRING VARIABLE NOT LEGAL HERE

SQR OF NEGATIVE ARGUMENT

SUBSCRIPT OUT OF BOUNDS

UNDECIPHERABLE OPERAND

UNDEFINED FUNCTION

UNDEFINED STATEMENT REFERENCE

UNDEFINED VALUE ACCESSED

UNDERFLOW - WARNING ONLY

UNMATCHED FOR

VARIABLE DIMENSIONED TWICE

WRITE TRIED ON READ-ONLY FILE

ZERO TO NEGATIVE POWER-WARNING

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.)

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

OPERATORS

SYMBOL	SAMPLE STATEMENT	PURPOSE/MEANING/TYPE
=	1ØØ A=B=C=Ø	Assignment operator; assigns a value to a variable;
	11Ø LET A = Ø	May also be used with LET.
↑	12Ø PRINT X↑2	Exponentiate (as in X^2).
*	13Ø LET C5 = (A*B)*N2	Multiply
/	14Ø PRINT T5/4	Divide
+	15Ø LET P = R1 +1Ø	Add
-	16Ø X3 = R3 - P	Subtract

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

=	17Ø IF D=E THEN 6ØØ	*expression* "equals" *expression*
#	18Ø IF (D+E)#(2*D)THEN 71Ø	*expression* "does not equal" *expression*
<>	18Ø IF(D+E)<>(2*D)THEN 7ØØ	*expression* "does not equal" *expression*
>	19Ø IF X>1Ø THEN 62Ø	*expression* "is greater than" *expression*
<	2ØØ IF R8<P7 THEN 64Ø	*expression* "is less than" *expression*
>=	21Ø IF R8>=P7 THEN 71Ø	*expression* "is greater than or equal to" *expression*
<=	22Ø IF X2<=1Ø THEN 65Ø	*expression* "is less than or equal to" *expression*
AND	23Ø IF G2 AND H5 THEN 9ØØ	*expression1* AND *expression2* must both be "true" for statement to be "true"
OR	24Ø IF G2 OR H5 THEN 91Ø	If either *expression1* OR *expression2* is "true", statement is "true"
NOT	25Ø IF NOT G5 THEN 95Ø	Statement is "true" when *expression* (NOT G5) is "false".
MAX	26Ø LET B = A2 MAX C3	Evaluates for the larger of the two expressions
MIN	27Ø LET B1 = A7 MIN A9	Evaluates for the smaller of the two expressions

STATEMENTS

NAME	EXAMPLE	PURPOSE
DATA	36Ø DATA 99,1Ø6.7, "HI!",16.2	Specifies data; read from left to righ
DIM	31Ø DIM A(72)	Specifies maximum string or matrix size.
END	4ØØ END	Terminates the program; the last statement in a program must be an END statement.
FOR...NEXT	35Ø 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 *STEP SIZE* may be omitted.
GO TO	33Ø GO TO 9ØØ	Transfers control (jumps) to specified statement number.
GO TO...OF	412 GO TO *n* OF 1ØØ,1Ø.2Ø	Transfers control to the *n*th statement of the statements listed after "OF".
GOSUB	42Ø GOSUB 8ØØ	Begins executing the subroutine at specified statement (see RETURN).
GOSUB...OF	415 GOSUB *n* OF 1ØØ,1Ø,2Ø	Begins executing the subroutine *n* of the subroutines listed after "OF" (See RETURN).
IF...THEN	34Ø IF A#1Ø THEN 35Ø	Logical test of specified condition; transfers control if "true".
INPUT	39Ø INPUT X$,Y2,B4	Allows data to be entered from tele printer while a program is running.
LET	3ØØ LET A=B=C=Ø	Assigns variable a value; LET is optional.
NEXT	355 NEXT J	Marks the boundary of the FOR loop.
READ	36Ø READ A,B,C	Reads information from DATA statement.
REM	32Ø REM--ANY TEXT**!!	Inserts non-executable remarks in a program.
PRINT	356 PRINT A,B,C$	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

NAME	EXAMPLE	PURPOSE
RESTORE	38Ø RESTORE	Permits re-reading data without re-running the program.
	385 RESTORE *n*	Permits data to be re-read, beginning in statement *n*.
RETURN	85Ø RETURN	Transfers control to statement following its GOSUB.
STOP	41Ø STOP	Terminates the program; may be used anywhere in program.

COMMANDS

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

FULL NAME	EXAMPLE	PURPOSE
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-1ØØ	Deletes all statements after and including the specified one.
	DEL-1ØØ,2ØØ	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-DØØ7,P^{C}D^{C}	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

FULL NAME	EXAMPLE	PURPOSE
LIST	LIS	Produces a listing of current program.
	LIS-15Ø	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-5Ø	Punches program to paper tape, beginning at specified statement.
RENUMBER	REN	Renumbers program from 10 in multiples of 10.
	REN-5Ø	Renumbers program from specified statement number in multiples of 10.
	REN-6Ø,*y*	Renumbers program from specified statement number in multiples of *y*.
RUN	RUN	Starts program execution.
	RUN-5Ø	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.

FULL NAME	EXAMPLE	PURPOSE
DEF FN	3ØØ 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)	31Ø PRINT ABS (X)	Gives the absolute value of the expression (X).
EXP (X)	32Ø PRINT EXP (X)	Gives the constant *e* raised to the power of the expression value (X); in this example, *e*↑X.
INT (X)	33Ø PRINT INT (X)	Gives the largest integer ≤ the expression (X).
LOG (X)	34Ø PRINT LOG (X)	Gives the natural logarithm of an expression; expression must have a positive value.
RND (X)	35Ø PRINT RND (X)	Generates a random number between Ø and 1; the expression (X) is a dummy argument.
SQR (X)	36Ø PRINT SQR (X)	Gives the square root of the expression (X); expression must have a positive value.
SIN (X)	37Ø PRINT SIN (X)	Gives the sine of the expression (X); X is real and in radians.
COS (X)	38Ø PRINT COS (X)	Gives the cosine of the expression (X); X is real and in radians.
TAN (X)	39Ø PRINT TAN (X)	Gives the tangent of the expression (X); X is real and in radians.
ATN (X)	4ØØ PRINT ATN (X)	Gives the arctangent of the expression (X); is real and in radians.
LEN (X)	41Ø PRINT LEN (A$)	Gives the current length of a string (A$), i.e., number of characters.
TAB (X)	42Ø PRINT TAB (X);A	Tabs to the specified position (X), then prints the specified value (A).
TYP (X)	43Ø 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) = Ø, gives the type of data in a DATA statement as: 1=number; 2=string; 3="out of data".
SGN (X)	44Ø PRINT SGN (X)	Gives: 1 if X>Ø, Ø if X=Ø, -1 if X<Ø

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: 1ØØ PRINT LEN (A$).*

FULL NAME	EXAMPLE	PURPOSE
DIM	1Ø DIM A$ (27)	Declares string length in characters.
LET	2Ø LET A$ = "**TEXT 1"	Assigns the character string in quotes to a string variable.
LEN	3Ø PRINT LEN (B$)	Gives the current length of the specified string.
= # > < >= <=	1Ø5 IF A$=C$ THEN 6ØØ 11Ø IF B$#X$ THEN 65Ø 115 IF N$(2,2)>B$(3,3) THEN 1Ø 12Ø IF N$<B$ THEN 999 125 IF P$ (5,8)>=Y$(4,7)THEN 1Ø 13Ø IF X$<=Z$ THEN 999	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.
INPUT	2Ø5 INPUT N$	Accepts the appropriate number of characters (followed by a *return*). The characters need not be in quotation marks if only one string is input.
INPUT	21Ø 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#	22Ø READ#5; A$,B$	Reads strings from the specified file.
PRINT#	31Ø PRINT#2; A$,C$	Prints strings on a file.

MATRICES, CONTINUED

NOTES: 1. Absolute maximum matrix size is 2500 elements.

2. Matrix variables must be a single letter from A to Z.

NAME	SAMPLE STATEMENT	PURPOSE
DIM	1Ø DIM A (1Ø, 2Ø)	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	2Ø MAT B = ZER	Sets all elements of the specified matrix equal to Ø.
	25 MAT D = ZER (*m,n*)	A new working size (*m,n*) may be specified after ZER.
MAT CON	3Ø 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	4Ø 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	5Ø 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.
	6Ø 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	7Ø MAT READ A	Reads matrix from DATA statements.
	75 MAT READ A(5,5)	Reads matrix of specified size from DATA statements.
	8Ø READ A(X,Y)	Reads the specified matrix element from a DATA statement.
	85 MAT READ #3; A	Reads matrix from the specified file.
	9Ø MAT READ #3,5; A	Reads matrix from the specified record of a file.

Continued on the next page.

MATRICES CONTINUED

NAME	SAMPLE STATEMENT	PURPOSE
MAT +	1ØØ MAT C = A + B	Matrix addition; A and B must be the same size.
MAT -	11Ø MAT C = A - B	Matrix subtraction; A,B, and C must be the same size.
MAT*	12Ø MAT C = A * B	Matrix multiplication; No. columns in A must equal No. rows in B.
MAT =	13Ø MAT A = B	Establishes equality of two matrices; assigns values of B to A.
MAT TRN	14Ø MAT B = TRN (A)	Transposes an m by n matrix to an n by m matrix.
MAT INV	15Ø 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:*

$$1 + \frac{\text{number of characters in the string}}{2}$$ *if there are an even number of characters,*

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

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

FULL NAME	EXAMPLE (Abbreviation)	PURPOSE
OPEN-	OPE-MYFILE,85	Opens file; assigns specified name and number of 64-word records.
KILL-	KIL-MYFILE	Deletes specified file.
FILES	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.

FILES , CONTINUED

FULL NAME	EXAMPLE (Abbreviation)	PURPOSE
READ#	18Ø READ# 2,3; A,B	Reads specified values from a specified file (2) and a specified record in that file (3).
	185 READ# 2,5	Resets the file pointer to the specified file record without erasing the record.
IF END#	19Ø IF END# 1 THEN 8ØØ	Transfers control to a specified statement number if the end of the specified file is encountered.

INDEX

This index is produced with the help of the two programs listed below. The first accepts two strings (topic and page references), and writes them on a file. The second program reads the file, sorts the topics alphabetically, and prints them on the teleprinter.

```
 1Ø      DIM A$[4Ø],B$[4Ø]
 2Ø      FILES INDX2
 7Ø      PRINT "START NEW INPUT."
 8Ø      PRINT "T";
 9Ø      INPUT A$
1ØØ      PRINT "P";
11Ø      INPUT B$
13Ø      PRINT #1;A$,B$
14Ø      GOTO 8Ø
15Ø      END

LIS
 OK
```

```
  1      IF END #1 THEN 2ØØ
  5      DIM A$[4Ø],B$[4Ø],C$[4Ø],L$[4Ø],M$[4Ø]
 1Ø      FILES INDX2, INDX3
 2Ø      LET C$="
 25      LET L$="ZZZZZZ"
 3Ø      READ #1;A$,B$
 4Ø      IF A$ < = C$ THEN 3Ø
 5Ø      IF A$ > = L$ THEN 3Ø
 6Ø      LET L$=A$
 65      LET M$=B$
 67      LET R=Ø
 7Ø      GOTO 3Ø
2ØØ      IF R THEN 4ØØ
2Ø5      PRINT L$" ..."M$
21Ø      PRINT #2;L$,M$
22Ø      LET C$=L$
225      LET L$="ZZZZ"
23Ø      LET R=1
235      READ #1,1
24Ø      GOTO 3Ø
4ØØ      END
```

Continued on the next page.