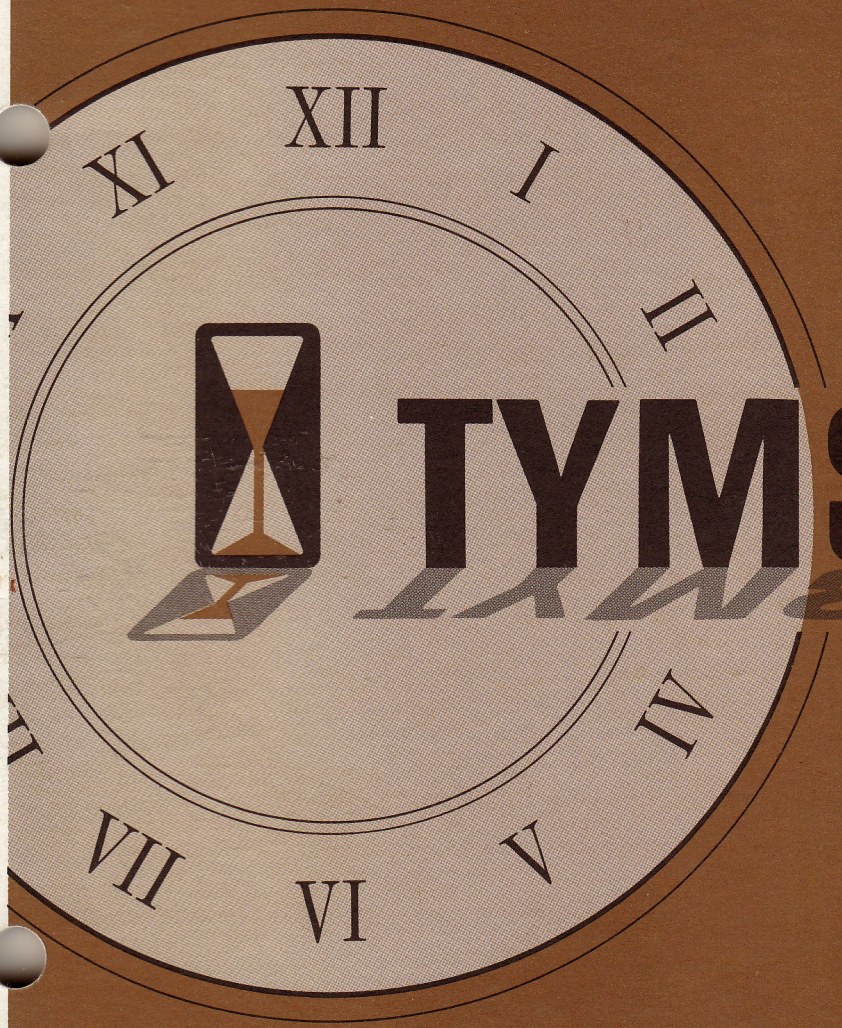
SUPER BASIC

REFERENCE SERIES



TYMSHARE®

TYMSHARE MANUALS

REFERENCE SERIES

SUPER BASIC

September 1968

Tymshare, Inc.
745 Distel Drive
Los Altos, California 94022

334 East Kelso Street
Inglewood, California
90301

464 Hudson Terrace
Englewood Cliffs, New Jersey
07632

Please send all comments about this manual to:
Library & Documentation Department, Tymshare, Inc.
925 East Meadow Drive, Palo Alto, California 94303

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

INTRODUCTION

Tymshare SUPER BASIC represents for the first time a truly conversational language incorporating features for both business and scientific applications.

It provides a powerful, yet simple set of commands and diagnostics that allow the new user to learn the language in a few hours and yet gives the experienced programmer the most extensive list of features ever included in a single language.

A few of the outstanding features of SUPER BASIC are:

- String manupulation
- Complex and logical variables
- Picture formats
- Conditional modifiers
- Direct commands
- Complete built-in editing

This manual contains a complete description of all the features of SUPER BASIC. Section 1 describes a subset of commands which, once learned by the beginning user, would enable him to write complete SUPER BASIC programs and run them on the Tymshare system.

Though written in a tutorial manner, Section 1 is well suited for reference. The rest of the commands are included in Section 2, which, while written more as reference material than as a tutorial guide, explains each feature in such a way that the user will be able to learn easily any new and unfamiliar material.

Section 3 contains a summary of the entire SUPER BASIC language.

Section 4 gives some sample programs written in SUPER BASIC and executed on the Tymshare system.

The appendices provide an alphabetic list of all SUPER BASIC commands and characteristics, and some information about storage allocation, the EXECUTIVE, and the terminal.

SECTION 1

A SUPER BASIC PRIMER

A. An Example

Suppose you want to write a SUPER BASIC program that will:

1. Request that you type in five numbers,
2. Add the numbers (if the result is zero, print the message SUM IS ZERO and stop),
3. Find the average (or mean) of the numbers,
4. Print out the sum and the mean,
5. Ask for five new numbers and repeat the cycle.

The simple program that solves this problem illustrates several elementary features and commands of SUPER BASIC which will be explained below. This is the program:

```
10 PRINT "TYPE FIVE NUMBERS"
20 INPUT A,B,C,D,E
27 S=A+B+C+D+E
32 IF S=0 THEN 70
45 M=S/5
50 PRINT S,M
60 GO TO 10
70 PRINT "SUM IS ZERO"
```

Before explaining this program step by step, we should first note some general SUPER BASIC conventions:

Line Numbers

All lines in the program begin with a number. These numbers identify the lines in the program, which are called statements, and specify the order in which the statements are to be executed. You can therefore type the program in any order provided that the statements are numbered in the order in which they are to be executed. Before the program is run SUPER BASIC sorts the statements into the order specified by their line numbers. NOTE: Line numbers must be integers between 0 and 999999.

Line Length

All statements in the sample program contain fewer than 72 characters (the maximum number of characters that may be typed across the page). Pressing the Line Feed key while a statement is being typed will continue the statement on the next line. A statement may be continued for several lines provided that the maximum limit of 256 characters is not exceeded. At the end of each entire statement, a Carriage Return must be typed. For example, the last statement in the program could have been typed as:

```
70 PRINT "SUM IS Lf
 ZERO"
```

NOTE: Spaces have no significance in SUPER BASIC except when they are included in messages to be printed (as in the above statement). Thus, spaces may be eliminated from all but these messages if you are not concerned with the readability of the printed copy.

Statements

Indirect Statements:

All the statements in the program above are called indirect statements. Any statement that begins with a line number is indirect; that is, the instruction or command in such a statement is not executed when it is typed, but is executed when the running program reaches the statement in normal sequence.

Direct Statments:

Direct statements do not begin with line numbers and are executed as soon as they are typed in. Direct statements cannot be saved as part of a program, as indirect statements can.

Some commands can be executed indirectly only, some directly only, and others may be used either way. For example, if GO TO 10 had been typed in our sample program without the line number 60, SUPER BASIC would have executed the command immediately by transferring to statement 10 and continuing execution from there. All this would have happened before you could have typed in any more statements. To find out if a command can be executed indirectly, directly, or both, see Appendix A.

PRINT "Text"

When SUPER BASIC encounters the first statement,

```
10 PRINT "TYPE FIVE NUMBERS"
```

the text included within the double quote marks is printed on the terminal. In this case the text is an instruction to the person who is running the program: he is instructed to type five numbers. The text also could have been enclosed in single quote marks.

INPUT Variable List

The INPUT command in the second statement,

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

will, when executed, cause SUPER BASIC to print a question mark followed by a space and wait for five numbers to be typed in.[1] The letters A through E in this statement are called variables.

---

[1] If you type in fewer numbers than required, SUPER BASIC will wait for the rest of the input.

Their purpose is to store values that will be used later in a computation. The first number typed will be stored in A, the second in B, and so on. Just as the comma is used to separate variables in the INPUT statement, it is used also to separate the values when they are typed in.[2] This will be shown later in an illustration of the actual run of the sample program.

Assignment Statement

Statement 27,

```
27 S=A+B+C+D+E
```

is called an assignment statement. This statement is similar to the others in the program which begin with a command word, except that in this case the word need not be typed. The optional word which may be included in an assignment statement is LET. For example, statement 27 could have been typed as:

```
27 LET S=A+B+C+D+E
```

The function of the assignment statement is to compute the value of the expression on the right of the = and assign that value to the variable on the left. NOTE: An expression may not be typed to the left of the =; for example, A+B=C is not a valid assignment statement.

Since the = means "is assigned the value of" rather than "is equivalent to", the following is a valid assignment statement:

```
15 X=X+1
```

If the value of X were 5 before this statement was executed, the statement would set X to 6.

The rules which govern naming variables and typing expressions correctly are included in Section 1(B).

IF Condition THEN Line Number

In statement 32,

```
32 IF S=0 THEN 70
```

we test to see if the value of S is zero. If it is, then, when this statement is executed, SUPER BASIC will go to line 70 where it prints the message SUM IS ZERO and stops since there are no more statements to execute. If S is not zero, SUPER BASIC will continue to the next statement in sequence,

```
45 M=S/5
```

This assignment statement calculates the mean and assigns the result to M. Note that since the sum of the five numbers has been calculated previously and assigned to the variable S, we do not need to repeat the computation of S in this statement.

---

[2]Spaces also may be used to separate the values when typed in.

PRINT Variable List

When SUPER BASIC encounters the next statement,
50 PRINT S,M
the values which were computed and assigned to S and M are printed. A comma is used to separate the variable names.

Since any number or expression also may follow the PRINT command, we could have omitted assignment statement 45 and typed the PRINT statement 50 as:
50 PRINT S, S/5

If SUPER BASIC were to encounter this statement, it would print S, compute S/5 and then print that result.

GO TO Line Number

The IF statement causes a conditional transfer; that is, SUPER BASIC will transfer to another part of the program provided that a certain condition is true. The GO TO command however, transfers to another statement unconditionally. Thus, when
60 GO TO 10
is executed, SUPER BASIC goes to line 10 and requests new values for A,B,C,D, and E.

Note the importance of certain statements in the program.

- What would happen if we were to omit statement 50? SUPER BASIC would solve for S and M but would never print the results; the solution would remain the secret of the computer.

- Suppose we omitted statement 32. Then, if S were zero, SUPER BASIC would not print SUM IS ZERO and stop as we had specified. Instead, the mean would be calculated (also as zero), the sum and mean would be printed, and five more numbers would be requested.

- If we were to omit statement 60 (the unconditional transfer), SUPER BASIC would, after printing the values of S and M, print the message SUM IS ZERO and stop.

Running The Example Program

The entire procedure for entering the Tymshare system, running the example program, and leaving the system is illustrated and explained below:

```
PLEASE LOG IN: Cr.....
```
The system types this request as soon as the connection to the Tymshare computer is made. Type a Carriage Return.

```
ACCOUNT: Q5 Cr.....
```
Type your account number followed by a Carriage Return.

```
PASSWORD: Cr.....
```
Type your password followed by a Carriage Return. The password does not print.

```
USER NAME: JONES Cr.....
```
The system next asks for the user name, which is typed and followed by a Carriage Return.

```
PROJ CODE: K123 Cr.....
```
A response to this last request is optional. If desired, type a project code followed by a Carriage Return; otherwise type only a Carriage Return.

```
READY 4/27  11:17.....
-SBASIC↵
```
The system is ready. The dash indicates that you in the EXECUTIVE and can call SUPER BASIC by typing SBASIC Cr. The > indicates that SUPER BASIC is ready and you may begin to type in the program statements.

```
>10 PRINT "TYPE FIVE NUMBERS"↵
>20 INPUT A,B,C,D,E↵
>27 S=A+B+C+D+E↵
>32 IF S=0 THEN 70↵
>45 M=S/5↵
>50 PRINT S,M↵
>60 GO TO 10↵
>70 PRINT "SUM IS ZERO"↵
```

```
>RUN Cr.....
```
The direct command RUN Cr causes SUPER BASIC to execute the program.

```
TYPE FIVE NUMBERS.....
? 10,20,30,40,50Cr
```
Five numbers are typed in, separated by commas. A Carriage Return is typed after the last number.

```
150           30.....
```
The sum is 150, the mean is 30.

```
TYPE FIVE NUMBERS.....
? 13,-7,-23,19,8Cr
```
SUPER BASIC again requests five numbers.

```
10            2.....
```
This time the results are 10 and 2.

```
TYPE FIVE NUMBERS.....
? 40,25,-50,15,-30Cr
SUM IS ZERO
```
The sum of the next five numbers is zero. SUPER BASIC prints the specified message and stops.

>QUIT Cr.....The QUIT command (which may be abbreviated as Q) returns you to the EXECUTIVE where you can leave the system by typing LOGOUT Cr.

-LOGOUT Cr

TIME USED 0:3:16

PLEASE LOG IN: After this message prints, hang up. The PLEASE LOG IN: request prints in case another user is waiting to use the system.

NOTE: We could have interrupted the execution of the program at any time by pressing the ALT MODE or ESC key twice in reply to the INPUT question mark.

B. Fundamental Concepts Of SUPER BASIC

Numbers

How To Type Numbers Into SUPER BASIC:

Numbers may be typed into SUPER BASIC in three ways:

- Integer format (whole numbers without a decimal point).
- Decimal format (numbers containing a decimal point).
- E format. The letter E means "times ten to the power of". For example, -53×10^9 can be typed as -53E9, and the number .00000000000063 (in which twelve zeroes follow the decimal point) can be typed as .63E-12. The E notation cannot stand alone; thus, 1000 may be typed as 10E2 or 1E3 but not as E3.

SUPER BASIC will accept up to eleven significant digits; any number containing more significant digits will be rounded to eleven.

The largest number that SUPER BASIC will accept is .57896044618E77. NOTE: This number will result if the user divides any number by zero.

Note that the following are not numbers in SUPER BASIC: 1/2, $\sqrt{4}$, $(5/6)^{17}$. They are expressions which SUPER BASIC must compute into a number of acceptable form. Such expressions may not be used as data input to a program.

How SUPER BASIC Prints Numbers:

SUPER BASIC ordinarily will print numbers as follows:[3]

[3]You can control the format in which SUPER BASIC will print numbers. For more information, see Section 2(L).

- Numbers are stored internally in SUPER BASIC with eleven significant digits but are rounded to eight digits when printed.

- If the absolute value of the number[4] is less than .1 and greater than or equal to 100,000,000, the number will be printed in E format. Otherwise, it will be printed as an integer or decimal number.

- Trailing zeroes after a decimal point are not printed.

To illustrate these rules, we will use the PRINT Command directly; that is, without a line number so that SUPER BASIC will execute the command immediately.

```
>PRINT .076, -568905117
 .76E-01        -.56890512E+09

>PRINT -.600174172, 63.810
-.60017417       63.81

>PRINT 6E7, 6E8
 60000000         .6E+09

>
```

Variables

The purpose of a variable is to be assigned or to store a single value that will be used in some computation or will be printed as a solution. A variable is so called because its value may be changed.

Variable Names

A variable can be named in one of three ways:[5]

- Any letter from A to Z.

- Any letter followed by any digit from 0 to 9.

- Any letter followed by the dollar sign, $.

Thus, some acceptable variable names are:

Z B0 M4 I$

and some unacceptable names are:

[4] Absolute value simply means: For positive numbers, the number itself; for negative numbers, the number without its minus sign.

[5] Subscripted variables are discussed in Section 2(F).

The VAR=ZERO Command

A variable ordinarily must be defined (previously assigned a value either by appearing on the left hand side of an assignment statement or in an INPUT[1] statement) before it can be referred to in a SUPER BASIC statement. Referring to an undefined variable will cause an error message to be printed unless the VAR=ZERO command has been executed previously. This command automatically assigns the initial value of zero to all variables yet to be typed which would otherwise be considered as undefined. For example:

```
>10 VAR=ZERO
>20 PRINT "TYPE A"
>30 INPUT A
>40 PRINT A,B
>RUN
TYPE A
? 6
 6              0

>
```

The user typed in the value of 6 for the variable A. B was never defined, but because of the VAR=ZERO command in line 10, B's initial value was set to zero. If line 10 had been omitted, the PRINT A,B statement would have caused SUPER BASIC to print A and then an error message indicating that B was never defined.

The VAR=ZERO command also can be executed directly. Note that the RUN command ordinarily ignores any direct commands that might have been given previously and executes only those statements preceded by line numbers. The direct VAR=ZERO command is an exception; it will not be ignored when the RUN command is given. For example:

```
>10 X=15
>20 PRINT X,Y
>VAR=ZERO

>RUN
 15             0

>
```

[1]Or READ statement (Section 1(D)).

Only the value of X was assigned in line 10. The direct VAR=ZERO command, since it was given before the RUN, caused the value of Y to be set to 0.

The VAR=UNDEF Command

This command nullifies the VAR=ZERO command. It affects only those variables which would be undefined if the VAR=ZERO command had never been given by once again declaring those variables to be undefined. For example:

```
>10 VAR=ZERO
>20 C=12
>30 PRINT C,D
>40 PRINT "NOW, 'VAR=UNDEF'"
>50 VAR=UNDEF
>60 PRINT C,D
>RUN
 12              0
NOW, 'VAR=UNDEF'
 12
ERROR IN STEP 60:
VARIABLE HAS NO VALUE
>
```

After the VAR=UNDEF command, C is still 12 but D is undefined, as though the VAR=ZERO command had never been given.

If we had assigned any value to D before giving the VAR=UNDEF command, D would not have been undefined by this command. Thus, if we were to insert 35 D=5 into the above program, VAR=UNDEF would have no effect and 5 would print as the value of D. Similarly, 35 D=0 would cause 0 to print as the value of D (since VAR=UNDEF undefines only those variables that are zero because of the VAR=ZERO command).

Arithmetic Expressions

Arithmetic expressions are formed by combining numbers and/or variables with arithmetic operators as in ordinary mathematical formulas.

There are seven arithmetic operators in SUPER BASIC:

SYMBOL	MEANING	EXAMPLE
↑	Exponentiation	X↑3 (=X^3)
-	Unary minus	-2↑2 (=-(2^2)=-4)
MOD	Modulo[6]	9 MOD 7 (=2)

[6]This standard mathematical operator is defined as follows: Y MOD Z =Y-Z*FIX(Y/Z). FIX is explained on page 37.

SYMBOL	MEANING	EXAMPLE
*	Multiplication	3*B (=3xB)
/	Division	1/4 (=1÷4)
+	Addition	8+F1
-	Subtraction	C$-5

Parentheses often are required in SUPER BASIC arithmetic expressions where they might not be needed in ordinary mathematical notation. For example, if you type $\frac{A+B}{C}$ as A+B/C in SUPER BASIC, the expression will be interpreted as $A+\frac{B}{C}$. This is because SUPER BASIC performs division before addition, unless parentheses are used to denote otherwise. Thus, $\frac{A+B}{C}$ must be typed as (A+B)/C.

The order in which SUPER BASIC will perform arithmetic operations is as follows:

1. Whatever is enclosed in parentheses will be computed first. When sets of parentheses appear within other sets of parentheses, the innermost set is evaluated first, then the next set, and so on.

2. Exponentiation.

3. Unary minus. Thus, if the expression is -2↑2, 2↑2 is computed first, and the value of the expression is -4.

4. Modulo operator. Thus, 15 MOD -6/2 is interpreted as (15 MOD -6)/2 and not 15 MOD -3.

5. Multiplication and division. If * and / appear in the same expression, SUPER BASIC calculates from left to right; that is, 3/B↑2*C is equivalent to $(\frac{3}{B2})$xC.

6. Addition and subtraction. Similarly as above, if these two operators appear in the same expression, SUPER BASIC calculates from left to right.

Mathematical Functions

A number of standard mathematical functions are available in SUPER BASIC. Each one has the same form: The name of the function followed by the argument (a number or an arithmetic expression) enclosed in parentheses.

Some of these functions are listed in the chart below.[7]

Function	Meaning
SIN(X)	Sine of X(X in radians)
COS (X)	Cosine of X(X in radians)
TAN (X)	Tangent of X(X in radians)
ATN (X) or ATAN (X)	Arctangent (in radians, over the range $-\pi/2$ to $+\pi/2$) of X
ATN (Y,X) or ATAN (Y,X)	Arctangent (in radians, over the range $-\pi$ to $+\pi$) of Y/X.
EXP (X)	Natural exponential of X, e^X.
ABS (X)	Absolute value of X
LGT (X) or LOG10(X)	Logarithm of X (base 10)
LOG (X)	Natural logarithm of X
SQR (X) or SQRT (X)	Square root of X

PI is a function with no argument. It is equal to the mathematical constant π, 3.1415926535.

These functions may be included in any expression; for example, all of the following are acceptable in SUPER BASIC:

```
Z$-EXP(X1+LOG(5/X1))
SQR(SIN(R)↑2+COS(Q)↑2)
LOG(N*X-SIN(PI/N))
```

Relational Expressions

A relational expression is one which compares one value to another (where the values may be represented by variables or arithmetic expressions) using the following relational operators:

[7] Additional mathematical functions of SUPER BASIC are described in Section 2(E).

SYMBOL	MEANING
<	less than
<=	less than or equal to
=	equal to
>=	greater than or equal to
>	greater than
<>or #	not equal to

Relational expressions commonly occur in an IF statement (where the THEN part of the statement will be executed only if the specified relation is true).
For example,

```
32 IF S=0 THEN 70
```

causes a transfer to statement 70 only if the value of S is zero; that is, if the expression S=0 is true. If S=0 is false, SUPER BASIC will continue to the next statement.

The following are acceptable relational expressions:

```
X>5
A<>B
Z$<=Y↑K+EXP(Z)
ABS(C3)=1
```

C. Using Loops In A Program: The FOR And NEXT Statements

Single Loops

Perhaps the single most important programming idea is the loop. While we can write useful programs in which each statement is performed only once, such programs place an unnecessary and substantial restriction on the power of the computer. Therefore, we prepare programs having parts which are performed not once but many times, perhaps with slight changes each time.

For example, suppose we want to write a program which will print out a table of the first 100 positive integers and their square roots. Without a loop, our program would be 100 lines long and would read as follows:

```
10 PRINT 1,SQR(1)
20 PRINT 2,SQR(2)
30 PRINT 3,SQR(3)
...
990 PRINT 99,SQR(99)
1000 PRINT 100,SQR(100)
```

Notice that the instruction is the same in every statement; only the number to which the instruction refers has changed from one line to the next.

Here is the same program written with a loop which uses the IF statement:

```
10 N=0
20 N=N+1
30 PRINT N, SQR(N)
40 IF N<100 THEN 20
```

Each statement in this program represents one of the four characteristics of every loop:

- Initialization (Statement 10 above) - The variable N is assigned the initial value of zero.[8] If this step were omitted, SUPER BASIC would not be able to compute the N+1 in the next statement, since N would be undefined.

- Modification each time through the loop (Statement 20) - The value of N is increased by 1. Without this statement, SUPER BASIC would execute the following instruction continually for zero and no other value.

- Body of the loop (Statement 30) - This is the actual instruction which we want to be executed repeatedly. The body of the loop may consist of any number of statements.

- Exit from the loop (Statement 40) - As long as N is less than 100, SUPER BASIC will go to statement 20 and once again pass through the modification and body of the loop. The last pass will be made when N is equal to 99; statement 20 will then set the value of N to 100, and statement 30 will print 100 followed by 10 (the square root of 100). Then the exit is made. N is not less than 100, so SUPER BASIC stops. If there were more statements in this program, the next statement in sequence then would be executed.

FOR And NEXT

Since loops are so important and are used so often in programming, SUPER BASIC provides the two indirect commands FOR and NEXT to simplify loop specification. The program above can be written as follows with these two commands:

```
10 FOR N=1 TO 100
20 PRINT N,SQR(N)
30 NEXT N
```

---

[8] N could have been replaced by any other acceptable variable name, but could not have been subscripted. Subscripted variables are discussed in Section 2(F).

Statement 10 specifies that N is initialized at the value 1 and that the final value of N is 100. N will not be set to a value greater than 100. The modification, an increase of 1 each time through the loop, is implied in this statement. The body of the loop is statement 20. The NEXT command in statement 30 instructs SUPER BASIC to return to the FOR statement for the next value of N. When the body of the loop has been executed for every specified value of N, SUPER BASIC will go to the statement following the NEXT. NOTE: The value of N after exit from the loop is the final value, 100.

The STEP Or BY Clause

N could have been increased to 100 in steps of any size other than the implied 1. To do this, we must specify the step size in a STEP or BY clause. For example, suppose we want to print the square roots of the first 50 even integers. The program would be written as the one above with statement 10 replaced by:

```
10 FOR N=2 TO 100 STEP 2
```

Note the following three equivalent forms of this statement:

```
10 FOR N=2 TO 100 BY 2
10 FOR N=2 STEP 2 TO 100
10 FOR N=2 BY 2 TO 100
```

The specified step size may be negative. For example, if we want to print the square roots of the first 100 integers in descending order, statement 10 would be:

```
10 FOR N=100 TO 1 STEP -1
```

FOR Value List

The FOR command also can be followed by a list of values for which the body of the loop is to be executed. For example, the following program prints the square roots of 2, 3, 8, 10, 12, 14 and 50:

```
10 FOR N=2,3,8 TO 14 STEP 2,50
20 PRINT N,SQR(N)
30 NEXT N
```

Errors In FOR Loops

If the FOR statement specifies an impossible range; that is, if the initial value is greater than the final value (less than the final value, for negative steps), the body of the loop will not be executed. SUPER BASIC will go to the statement following the corresponding NEXT.

Once a loop is entered, if the NEXT statement has been omitted, SUPER BASIC will execute the body of the loop once (for the initial value) and then execute the rest of the program which follows the loop.

More complicated FOR statements are allowed. The initial value, the final value, and the step size may be expressions of any complexity. For example, if N and Z have been assigned values earlier in the program, we could write:

```
55 FOR X=N+7*Z TO (Z-N)/3 STEP N
```

Note however, that a change in the values of N and Z within the loop will change neither the final value of X nor the step size. Variables and expressions in a FOR statement are evaluated only once; namely, the first time the statement is encountered. The final value and step size will not change once the loop has been entered.[9]

If the value of X in line 55 above were changed within the loop, this change would be accepted. For example, the following statements could be typed after line 55 to change the value of X to the value of N if X equals zero.

```
60 IF X=0 THEN 70
65 GO TO 75
70 X=N
75 (body of loop)
```

Nested Loops

It is often useful to have loops within loops. These nested loops are illustrated in the following skeleton examples:

Allowed

```
FOR X
  FOR Y
  NEXT Y
NEXT X
```

Not Allowed

```
FOR X
  FOR Y
NEXT X
  NEXT Y
```

Allowed

```
FOR X
  FOR Y
    FOR Z
    NEXT Z
    FOR W
    NEXT W
  NEXT Y
    FOR Z
    NEXT Z
NEXT X
```

Nested FOR loops of any complexity are allowed, but crossed FOR loops are not allowed.

The Multiple NEXT Statement

If the inclusion of nested FOR loops in a program results in two or more sequential NEXT statements, the NEXT statements may be combined and typed on one line as follows:

---

[9]Except when the WHILE or UNTIL modifier is used in the FOR statement (see page 59).

```
NEXT X
      } NEXT X,Y
NEXT Y
```

D. Supplying Data Within The Program: The READ And DATA Statements

We have already seen that assignment statements or INPUT statements may be used to assign values to variables. Another method involves the combined use of the READ and DATA statements.

Consider the following program:

```
10 READ N
20 S=0
30 FOR I=1 TO N
40 READ X
50 S=S+X
60 NEXT I
70 M=S/N
80 PRINT M
90 DATA 5,60,-10
100 DATA 40,-2,11
```

READ

The READ command always is followed by a variable name or a list of variable names separated by commas. When SUPER BASIC executes a READ statement, the first variable listed in the statement is assigned the first value in the collection of DATA statements (the "data block"), the next variable is assigned the next value, and so on.

Thus, when SUPER BASIC executes statement 10 of our sample program, N is assigned the value of 5. The next READ statement in the program is inside a FOR loop and is executed N (that is, 5) times. This statement causes X to be assigned the next available value in the data block at each pass through the loop. Therefore, when I=1 (the first pass through the loop), X is set to 60 and, in statement 50, added to S (which is initially zero). During each of the five times through the loop, a new value is assigned to X and added to S. The result is that when the exit from the FOR loop is made, S will be equal to the sum of the X's.

The program finally calculates M, the mean of the numbers, in statement 70.

DATA

The numeric values which are listed in DATA statements must be numbers, not expressions, and must be separated by commas.

The location of DATA statements in a program is arbitrary, although the usual procedure is to place them in a group at the end of the program. The only requirement is that the statements be numbered in the order in which the data is to be read.

The distribution of the elements of data among DATA statements also is arbitrary. For example, we could have typed, in place of statements 90 and 100 in our sample program, either

```
90 DATA 5,60,-10,40,-2,11
```

or

```
90 DATA 5
100 DATA 60,-10
110 DATA 40,-2,11
```

RESTORE

Once all the data has been read from a data block, another READ request will cause an error message telling you that you are out of data. However, if you wish at any time during the program to reread all or part of the data block, you can do this with a RESTORE command. When this command is executed, the next READ command will start reading data from the beginning of the data block; that is, from the first value in the first DATA statement. RESTORE can be executed either directly or indirectly.

For example, if now we wanted to use the formula

$$D=\sqrt{\frac{\sum_{i}^{N}(X-M)^2}{N}}$$ [10]

to calculate the standard deviation of the X's, we could add the following statements to our sample program:

```
110 RESTORE
120 READ N
130 A=0
140 FOR I=1 TO N
150 READ X
160 A=A+(X-M)↑2
170 NEXT I
180 D=SQR(A/N)
190 PRINT D
```

[10] The numerator of this fraction uses the mathematical symbol Σ meaning "the sum of". We want to find $(X-M)^2$ for every X and sum the results.

Statement 120 is necessary even though N already has the value of 5 at this point in the program. If this statement were omitted, the first X read by statement 150 would be 5, which is incorrect.

NOTE: If a program containing DATA statements is run more than once, the data block will be restored automatically.

E. Entering And Using A SUPER BASIC Program

Before you can call SUPER BASIC and run any of the sample programs or your own programs, you must enter the Tymshare system. The proper procedure is illustrated on page 5 and in Appendix C.

To call SUPER BASIC, type SBASIC followed by a Carriage Return. SUPER BASIC will reply with a > when ready to receive a command.

Typing A Program Into SUPER BASIC

Once SUPER BASIC is ready, start typing your program. Each statement must be terminated by a Carriage Return. Only after the Carriage Return is typed does SUPER BASIC analyze the statement and print an error message if the syntax is incorrect; that is, if the statement does not conform to the rules of SUPER BASIC's "grammar". After an error message prints, retype the line correctly.[11]

Remember that an indirect statement (one with a line number) is executed only when the running program reaches the statement in normal sequence; while a direct statement (without a line number) is executed immediately after you type the terminating Carriage Return.

Reading A Program From Paper Tape

Another way in which you may enter a program into SUPER BASIC is by reading the statements from paper tape which you previously punched "off line"; that is, while you were not connected to the computer.

How To Punch Paper Tape Off Line

To punch paper tape off line, turn the dial on the front of the terminal to LOCAL and depress the ON button on the paper tape punch controls. Then type the program exactly as you would

---

[11] SUPER BASIC's extensive editing features, which allow you to correct errors either before or after you type the Carriage Return at the end of an incorrect statement, will be described later in this manual.

if you were typing directly into SUPER BASIC, with the following exceptions:

- Always follow a Carriage Return with a Line Feed.
- Always follow a Line Feed with a Carriage Return.

If you make an error while punching a SUPER BASIC program onto paper tape, delete the incorrect character by typing a back arrow (←). Type the← as many times as necessary. In addition, an upward arrow (↑) immediately followed by a Carriage Return will delete an entire line.

After you have finished typing the program, punch a Control D and press the OFF button on the punch controls.

The TAPE Command

To read a program from paper tape into SUPER BASIC, type:

```
>TAPE Cr
```

and turn the paper tape reader control to START. NOTE: If you did not punch a Control D at the end of the tape, you may type the Control D from the keyboard after the tape is read.

Whatever you punched on the tape will print on the terminal when the tape is read. Any statements with syntax errors will be reprinted at the end of the entire program along with the appropriate error messages. Then the incorrect statements must be retyped.

Any part of a program may be punched on paper tape and entered with the TAPE command. For example, DATA statements alone can be saved on paper tape. If the rest of the program were saved on a file, you could enter the program first (with LOAD), and then append the DATA statements (with TAPE).

Running A Program

A SUPER BASIC program is executed with either of the direct commands RUN or GO TO.

- RUN begins execution at the lowest numbered statement of the program. Any direct or indirect statements executed previously are ignored.[12]

[12] With the exception of VAR=ZERO, VAR=UNDEF, and the BASE command, which is explained in Section 2(F).

- GO TO followed by a line number begins execution at the statement specified. Any direct or indirect statements executed previously are not ignored; all information is retained.[12.5]

If the program can be executed, the results will be given quickly. This does not necessarily mean that the program is free from error and the answers are correct. There might be a logical error that SUPER BASIC cannot find. Or, there might be an error (other than a syntax error) which prevents execution. If this is so, SUPER BASIC will print a message indicating why it cannot execute the program. Correct your error and try again.

Saving A Program

Once you have a program that is running correctly, you may want to save it on a file (a storage area set aside for you in the Tymshare computer). To do this, type the direct command SAVE followed by the name of the file and a Carriage Return. NOTE: The file name must be surrounded by slashes and may contain any characters except the slash.

SUPER BASIC replies with NEW FILE if you do not already have a file with that name, and OLD FILE if you do have a file with that name.

In reply to NEW FILE or OLD FILE, you either:

- Confirm the command by typing a Carriage Return.

NOTE: A carriage return after OLD FILE causes the contents of the old file to be replaced. Or,

- Abort the command by pressing the ALT MODE/ESC key.

Example:

```
>SAVE /KL22/ Cr
NEW FILE Cr
>
```

NOTE: Only indirect statements (those with line numbers) will be saved on the file.

To save part of your program, type SAVE followed by the file name and a comma. Then type the line numbers of the statements you wish to save. Separate the numbers with commas and use the dash (-) to indicate a range. Thus,

```
>SAVE /INT/,1-15,30,70-100 Cr
OLD FILE Cr
>
```

[12.5] When a program containing the READ command is executed more than once, the data is reread from the beginning of the data block even if a direct GO TO was given to execute the program.

replaces the former contents of the file /INT/ with statements 1 to 15,30, and 70 through 100.

An Example Of TAPE, RUN And SAVE

In the following example, a short program is read from paper tape, corrected, executed, and saved on a file. The example also illustrates logging in and logging out.

```
PLEASE LOG IN:↵
ACCOUNT:  A3↵
PASSWORD:↵
USER NAME:  SMITH↵
PROJ CODE:↵
READY 7/1 16:17
-SBASIC↵

>TAPE↵
10 PRINT "TYPE THE BASE AND THE
 HEIGHT"
20 IMPUT B,H
30 A=1/2*B*H
40 PRINT "THIS IS THE AREA:"
50 PRINT A
D^C                                  D^C ends the TAPE command.
20 IMPUT B,H                         This line contains a syntax error.
SYNTAX ERROR
>20INPUT B,H↵                        The error is corrected.
>RUN↵
TYPE THE BASE AND THE HEIGHT
? 10,6↵
THIS IS THE AREA:
 30
>SAVE /AREA/↵                        The program is saved on a new
                                     file named /AREA/.
NEW FILE↵
>QUIT↵

-LOGOUT↵
TIME USED 0:2:58
PLEASE LOG IN:
```

Reusing A Saved Program

To reenter a program saved on a file, type LOAD followed by the file name and a Carriage Return.

Example:

```
>LOAD /KL22/ Cr
>
```

Looking At A Program

At any time you may have part or all of your program printed by typing the direct command LIST.

Typed alone, LIST causes the entire program to be listed. When LIST is followed by a line number or numbers, only the statements specified are listed. For example,

```
>LIST 4,10,20-30,65 Cr
```

will print lines 4,10,20 through 30, and 65.

You can stop the printing at any time by pressing the ALT MODE/ESC key.

Comments In A Program

Either an exclamation point (!) or the word REM is used to insert remarks or comments as direct or indirect statements. For example:

```
>REM NOW WE WILL TYPE "RUN"↵
>
```

Since this remark is a direct statement, it will not be saved with the program. The following remarks

```
>10 ! THIS PROGRAM CALCULATES THE↵
>20 ! AREA OF A TRIANGLE↵
```

will be saved because they are indirect statements. They will be listed along with the rest of the program, but will not be printed out when the program is run. Any characters can be typed after ! or REM.

In addition, ! can be used to insert comments at the end of direct or indirect statements. For example,

```
>45 M=S/5          !CALCULATES THE MEAN↵
>GO TO 20          !OBSERVE THE RESULTS↵
>
```

Self-Starting Programs

A program which has been saved on a file may begin to execute automatically as soon as it is loaded.

To accomplish this, you must store a RUN or direct GO TO command on the file immediately following the program. You cannot do this in SUPER BASIC because direct commands execute as soon as they are typed and cannot be saved with the program when the SAVE command is given. However, the Tymshare editing language, EDITOR, allows you to read in the SUPER BASIC program from a file, append a RUN or direct GO TO command and then write the program back on the file.[13] When the program is loaded into SUPER BASIC, it will begin to execute immediately.

---

[13]For more information, see the Tymshare EDITOR Manual, Instant Series.

F. Simple Editing In SUPER BASIC

This section describes only the simplest editing features of SUPER BASIC. The advanced editing features - those which SUPER BASIC shares with EDITOR - are explained in Section 2(M).

Inserting Statements

To insert one of more lines between two existing statements in your program, simply type the new statements with line numbers that lie between the numbers of the existing statements. For example, if you have left out a statement between statements 40 and 50, type the additional statement with any number from 41 to 49. SUPER BASIC will list and execute your program in numerical sequence.

Deleting Statements

The DELETE Command

To delete a statement from your program, either type the line number of the statement followed by a Carriage Return or use the direct command DELETE (may be shortened to DEL). DELETE followed by a line number or numbers will delete the specified statements. For example, either DELETE 10 Cr or 10 Cr will delete statement 10. The command

```
>DEL 5,10-35,70 Cr
```

will delete lines 5, 10 through 35, and 70.

To delete the entire program, type DELETE ALL Cr. This command also deletes the values of all variables. Remember to give this command whenever you are finished with one program and wish to load another; SAVE will not remove a program from SUPER BASIC.

Control Q

In addition to deleting existing lines in your program, you may delete an incorrect statement (direct or indirect) at any time before typing the terminating Carriage Return. To do this, type a Control Q (Q^C). An ↑ will print on the terminal and the line will be deleted. Then retype the entire statement.

In the example below, the user deletes 40 FOR I=1 TO with a Q^C and retypes the statement correctly:

```
>40 FOR I=1 TOQ^C↑
40 FOR J=1 TO 3 Cr
>
```

Changing Statements

To change any statement in your program, simply retype it with the same line number. Whenever you enter a new statement with the same number as a line already in the program, the old

statement is replaced by the new one.

Control A

If you make an error while typing a statement, you may delete the incorrect character before you type the terminating Carriage Return. To do this, type a A^C after the incorrect character (a ← will print on the terminal). Use A^C repeatedly to delete as many characters as necessary.

Example:

```
>10 PRIMA^C←NT "TYPE X" Cr
>20 X=A^C←A^C←INPUT X Cr
>LIST Cr
10 PRINT "TYPE X"
20 INPUT X
>
```

NOTE: The back arrow (shift O) has the same effect as A^C.

G. Review Of Commands In Section 1

The following commands have been discussed thus far in this manual:

Command	Example	Purpose
Assignment Statement	45 M=S/5	Assigns values to variables
DATA	90 DATA 5,60,-10	Stores data in a program
DELETE or DEL	DEL 5,10-35,70	Deletes all or part of a program
FOR and NEXT	10 FOR N=1 TO 100 20 PRINT N, SQR(N) 30 NEXT N	Repeats execution of a line or lines for specified values
GO TO...	60 GO TO 10 GO TO 10	Unconditional transfer
IF...THEN...	32 IF S=0 THEN 70	Conditional transfer
INPUT	20 INPUT A,B,C,D,E	Accepts data input from the keyboard

(Continued)

LIST	LIST 4,10,20-30	Lists all or part of a program
LOAD	LOAD /KL22/	Enters program statements from a file
PRINT	70 PRINT "SUM IS ZERO" PRINT X,Y	Prints text and values of variables
QUIT or Q	QUIT	Returns to the EXECUTIVE
READ	10 READ N	Accepts input from DATA statements
REM and !	REM PRINT A 55 A=A+1 !ADD 1	For comments or remarks
RESTORE	110 RESTORE	Rereads DATA statements from the beginning
RUN	RUN	Starts execution at the lowest numbered statement
SAVE	SAVE /KL22/	Saves all or part of a program
TAPE	TAPE	Enters program statements from paper tape
VAR=UNDEF	70 VAR=UNDEF	Nullifies the effect of VAR=ZERO
VAR=ZERO	10 VAR=ZERO	Initializes variables to zero

Many useful SUPER BASIC programs can be written and used with these few commands. We conclude Section 1 with two more examples. Try these on the terminal, together with some programs of your own. The fastest and easiest way to learn the Tymshare system is to use it!

H. Sample Programs

Product Of A Set Of Numbers

This program will read up to 1000 numbers from DATA statements and print the product of the numbers. The last number typed in the data block is to be 5E55. This makes it unnecessary for the user to count how many data items he types, as will be explained below.

```
10 P=1
20 FOR I=1 TO 1000
30 READ X
40 IF X=5E55 THEN 80
50 P=P*X
60 IF P=0 THEN 80
70 NEXT I
80 PRINT P
90 DATA 15,-9,1.5,33,6,-4,22,9,5E55
```

Each number that is read is compared to what we know is the last data item, 5E55. If the number read is not equal to 5E55 (that is, we have not yet reached the end of the data block), the number will be accepted as one which should be multiplied. The product is stored in the variable P. P is initialized to 1 in line 10 so that the first time through the FOR loop, the first data item (1*X) will be stored in P. Each subsequent time through the loop, the product calculated thus far will be multiplied by the number just read. When 5E55 is read, SUPER BASIC will go immediately to line 80 and print the product, P.

Line 60 states another condition under which SUPER BASIC should print the product calculated thus far; that is, if this product is 0. In this case there is no reason to continue multiplying, since the result will be 0 regardless of what numbers follow. NOTE: This statement is optional; it merely saves calculation time if one of the data items is 0.

Try this sample program with any set of numbers. If you use the data provided in the above example, the answer should be 31755240. You can substitute any number in place of 5E55 in this program, as long as the number you choose appears only at the end of the data block.

Double Declining Balance Depreciation

This program calculates and lists the depreciation and book value of an asset at the end of every year of its useful life.

The original cost (C) and the estimated useful life (L) of

the asset are used to calculate the depreciation (D). At the end of the first year,

$$D = \frac{2xC}{L}$$

The book value at the end of the first year is C-D (original cost less depreciation). For each subsequent year, the depreciation and book value are calculated by the same formulas as above, substituting for C the book value at the end of the previous year.

The user is asked to type in the original cost and the estimated useful life. Following the listing of the program is a sample run for an asset which costs $7,000 and is depreciated over 15 years.

```
>LIST
0! DOUBLE DECLING BALANCE DEP.
10 PRINT "TYPE COST OF ASSET AND"
20 PRINT "ESTIMATED USEFUL LIFE"
30 INPUT C,L
40 PRINT "YEAR","DEP.","BOOK VALUE"
50 FOR X=1 TO L
60 D=2*C/L
70 C=C-D
80 PRINT X,D,C
90 NEXT X
>RUN
TYPE COST OF ASSET AND
ESTIMATED USEFUL LIFE
? 7000,15
YEAR            DEP.            BOOK VALUE
 1               933.33333       6066.6667
 2               808.88889       5257.7778
 3               701.03704       4556.7407
 4               607.56543       3949.1753
 5               526.55671       3422.6186
 6               456.34915       2966.2695
 7               395.50259       2570.7669
 8               342.76891       2227.9979
 9               297.06639       1930.9316
 10              257.45754       1673.474
 11              223.12987       1450.3441
 12              193.37922       1256.9649
 13              167.59532       1089.3696
 14              145.24928       944.12032
 15              125.88271       818.23761

>
```

The commas in statement 40 caused spaces to be printed between the column headings. All of the PRINT statement forms and rules are discussed in detail in Section 2(B).

SECTION 2

SUPER BASIC ADVANCED FEATURES

A. The Multiple Assignment Statement

More than one variable can be assigned the same value in one statement. The variables to be assigned must be separated by commas. For example,

```
10 X,Y=5
70 LET A,B,C(2),D(1,1)=0
X(1),Y,Z=15*S/R
```

More than one assignment can be made in a single statement, as follows:

```
15 LET Q=4,S=16
30 A=3,M,N=5,W=COS(15)
100 J=SQR(X),K=J+3,H,G(1)=0
```

The assignments are made from left to right; thus, in statement 100 above, the value of K is set to SQR(X)+3.

As shown above, use of the word LET is optional.

Be careful to note that each of the examples above is a single statement. Two separate statements cannot be typed on one line and separated by commas. For example, PRINT A,PRINT B is not acceptable, nor is B=C*EXP(C),PRINT A+B.

B. Additional Printing Features

Printing Blank Lines

The PRINT command typed alone causes a Carriage Return to be printed. This form of the command is useful in making terminal output more readable by inserting blank lines.

For example,

```
30 PRINT "LINE 1"
40 FOR I=1 TO 4
50 PRINT
60 NEXT I
70 PRINT "LINE 2"
```

will cause four blank lines to be printed between LINE 1 and LINE 2.

The PRINT Zones

Separate PRINT commands cause the specified printout to be on separate lines. Thus,

```
100 PRINT "BOOK VAL"
110 PRINT X
```

prints BOOK VAL at the beginning of one line and the value of X at the beginning of the next line. The following program:

```
15 FOR I=1 TO 12
20 PRINT I
25 NEXT I
```

will print the first twelve integers, each at the beginning of a line.

SUPER BASIC does, however, provide ways to print more than one number and/or string of text on one line. The characters to be printed fall into "zones", the length of which depend on whether the comma, semicolon, or colon is used in the PRINT statement.

Normal PRINT Zones

The width of the terminal paper normally is divided into five zones of fifteen spaces each. A comma is used in the PRINT statement to instruct SUPER BASIC to go to the beginning of the next zone. Thus, PRINT A,B,C,D,E will print the values of those five variables across the page. Each number will be left justified in a field of fifteen spaces. Any positive number will be preceded by a space due to the omission of the plus sign.

If there are more commas in a PRINT statement after the fifth zone is printed, printing will continue from the first zone on the next line. Thus,

```
10 FOR I=1 TO 12
20 PRINT I,
30 NEXT I
```

will print the first five integers on one line, the second five on the next line, and 11 and 12 on a third line.

If another PRINT statement were added to this example, the first value or text listed in the additional statement would be printed in the zone immediately following the 12 (the third zone on the line).
Thus,

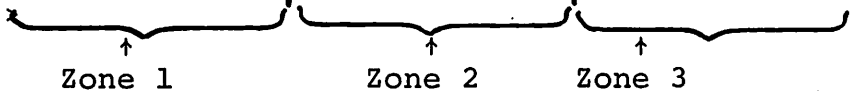
```
>10 FOR I=1 TO 12
>20 PRINT I,
>30 NEXT I
>40 PRINT "XXX"
>RUN
 1              2              3              4              5
 6              7              8              9              10
 11             12             XXX

>
```

Inserting the statement 35 PRINT in the above example would have caused the XXX to print at the beginning of the next (fourth) line.

If text to be printed contains more than fifteen characters, it will extend into the next zone, and the next value or text to be printed will occupy the following zone. For example,

```
>PRINT "CURB WEIGHT (LBS)=",A
CURB WEIGHT (LBS)=             111
```



The first string of text contains 18 characters. The value of A is printed in the third zone.

Packed PRINT Zones

A packed form of terminal output is available by using the semicolon in the PRINT statement. The semicolon instructs SUPER BASIC to skip from two to five spaces before printing the next number or text. The exact number of spaces depends on the last position in which SUPER BASIC printed before it encountered the semicolon.[14]

For example,

```
>PRINT "CURB WEIGHT (LBS)=";A
CURB WEIGHT (LBS)=    111
>PRINT "THIS IS";1;"EXAMPLE"
THIS IS   1    EXAMPLE
>
```

Concatenated PRINT Zones

To print numbers and/or text with no separating spaces, use the colon in the PRINT statement. Remember that positive numbers will be preceded by one space because of the missing plus sign. Thus,

```
>PRINT "CURB WEIGHT (LBS) =":A
CURB WEIGHT (LBS) = 111

>PRINT "B IS NEGATIVE":B:A
B IS NEGATIVE-76.3 111

>PRINT "CONCAT":"ENAT":"ED"
CONCATENATED

>
```

[14]The paper is divided into zones of three spaces each. SUPER BASIC first skips two spaces and then, if not positioned at the beginning of a zone, will move to the beginning of the next zone.

The following is not permitted in SUPER BASIC:

```
>PRINT "CURB WEIGHT (LBS) = "A
```

A comma, semicolon or colon must be typed after the text.

Concatenation Of PRINT And INPUT

When text is printed immediately before an INPUT command, the INPUT question mark need not appear on a separate line. A comma, semicolon, or colon at the end of the preceding PRINT statement will move the question mark to the end of that line. SUPER BASIC will wait there for the input. For example,

```
>10 PRINT "WHAT IS X":
>20 INPUT X
>30 PRINT "X SQUARED =":X↑2
>RUN
WHAT IS X? 15↵
X SQUARED = 225
>
```

SUPER BASIC provides another control to concatenate input with printed text. Instead of a Carriage Return, a D^C may be typed after the last item of data typed in reply to an INPUT command. The input will be accepted as usual, but the carriage will not be returned. Thus, any more text to be printed will appear on that same line rather than on the next line.

Example 1

```
>10 PRINT "B = ":
>20 INPUT B
>30 PRINT " (THIS IS THE BASE)"
>RUN
B = ? 13D^C (THIS IS THE BASE)
>
```

In this example, the user typed a D^C instead of a Carriage Return after the requested input. SUPER BASIC then printed the text (THIS IS THE BASE) on the same line.

Example 2

```
>10 PRINT "WHAT IS R":
>20 INPUT R
>30 PRINT " S":
>40 INPUT S
>50 PRINT " T":
>60 INPUT T
>RUN
WHAT IS R? -6D^C S? 4D^C T? 3↵
>
```

The TAB Function

The function TAB(X) is used in the PRINT statement to move the print head to the Xth print position on the line. The function is used with a colon if the number or text which follows it is to be printed at the specified position. For example,

```
>PRINT TAB(20):B
                    -456
                    ↑
                20th position
>PRINT A:TAB(12):B
 12         -456
            ↑
          12th position
>
```

If a semicolon is used after the TAB function, the print head will move beyond the specified print position; a comma causes it to move to the next PRINT zone of 15 spaces.

If the semicolon or comma which precedes the TAB function causes the print head to move beyond the position specified by the TAB, the TAB will be ignored. For example,

```
>PRINT A,TAB(12):B
 12              -456
                 ↑
             16th position
>
```

The comma caused the print head to move past the first field of 15, so TAB (12) was ignored.

C. Additional IF Statement Features

IF Condition THEN Statement

In addition to line numbers, SUPER BASIC statements may be typed after the word THEN in an IF-THEN statement. If the IF condition is false, the THEN statement will not be executed.

Examples

```
IF X>4 THEN A=B   If X is greater than 4, A will be set to the
                  value of B.
IF A=B THEN PRINT "A EQUALS B"
                  The message A EQUALS B will be printed only
                  if A and B are equal.
```

The IF-THEN-ELSE Sequence

The word ELSE followed by a statement can be added to the IF-THEN sequence. This form allows the THEN statement to be executed if the condition is true, but executes the ELSE state-

ment if the condition is false. The program continues to the next statement in order unless the THEN or ELSE clause it executes is one which transfers to another line.

Examples

```
IF X=.5 THEN 200 ELSE 300
                        If X is .5, the program will go to
                        line 200; otherwise, it will go to
                        line 300.
IF N=0 THEN 50 ELSE C=T,D=T/N
                        If N is 0, the program will go to
                        line 50; otherwise, the assignment
                        statement in the ELSE clause will
                        be executed, setting C to T and D
                        to T/N.
IF A=B THEN PRINT "A EQUALS B" ELSE PRINT "A AND B NOT EQUAL"
                        If A and B are equal A EQUALS B
                        will print; if not, A AND B NOT EQUAL
                        will print.
```

Any indirect statement (except DATA, REM or !) can be included in a THEN or an ELSE clause.

Combining IF Statements

Any number of IF-THEN and/or IF-THEN-ELSE sequences may be used together, such as:

IF X=4 THEN IF P=L THEN R=80 ELSE 300 ELSE X=X*Y

In this example, if X is not 4 (a false condition), the ELSE clause will set X to X*Y and the program will continue with the next statement in order. If X is 4 (a true condition), the THEN clause will be executed to check to see if P is equal to L. If so, the value of R will be set to 80 and the program will continue; otherwise, the program will transfer to line 300.

If the statement does not contain the same number of ELSE and THEN clauses, the last ELSE is matched with the closest THEN. FOR example, in the statement

```
IF A>0 THEN IF B>10 THEN C=1 ELSE C=2
```

if A is not positive, the program will go to the next statement since there is no ELSE clause accompanying the first THEN. If A is positive and B is greater than 10, C will be given the value of 1. If A is positive but B is less than or equal to 10, C will be given the value of 2.

D. Data File Input And Output

Files are a convenient method of supplying a program with large amounts of data or saving the results of the execution of a program. Up to three 90,000 character files can be used concurrently for input to or output from a program. The commands which will accomplish this are explained below.

Opening A File

Before a data file can be read or written, it must be opened (and at the same time given a number) with the command:

OPEN /file name/ FOR { SYMBOLIC or BINARY } { INPUT or OUTPUT } AS FILE n

or the short form:

OPEN /file name/, { SYMBOLIC or BINARY } { INPUT or OUTPUT } ,n

The file number n, which can be any positive numeric expression, is necessary in every OPEN statement to specify which file the user is working with, since he may have up to three files open at one time. A file number that is not an integer will be truncated.

Input or output files may be symbolic or binary. Since data written on a binary file is not in the usual decimal form but in binary form, the file cannot be printed on the terminal (and be meaningful). Binary form however, requires less storage space and is especially useful if a program creates a large number of results that are to be used as input to another SUPER BASIC program.

When a file is opened for input, it need not be specified in the OPEN statement as symbolic or binary. If the word SYMBOLIC or BINARY is omitted, SUPER BASIC will check to see what type of file it is and will read it as such. If the file type is specified but does not match the file, an error message will be printed.

When a file is opened for output, the user must specify if the file is to be binary; otherwise, a symbolic output file will be written. Thus,

```
OPEN  /BDATA/, BINARY OUTPUT, M*N
```

will open for binary output the file /BDATA/, the file number of which equals the value of M*N. The following

```
OPEN  /SDATA/, OUTPUT, 4
```

will open for symbolic output the file /SDATA/.

A file need not exist in the user's directory to be opened for output; the OPEN command will create a file of the specified name and type automatically.

NOTE: Opening a file initializes input or output at the beginning of the file.

Input From A File

The command used to read data from a file takes the form:

INPUT FROM n:variable list
where n is the input file number. For example,
10 OPEN /AFILE/,INPUT,2
20 INPUT FROM 2:X,Y,Z
reads three values from /AFILE/ and assigns them to the variables X,Y and Z respectively.

The entries in a data file may be separated by commas or spaces, with a Carriage Return at the end of each line of data. The entries can be numbers but not expressions.

Output To A File

To write on a file, use either of the equivalent forms:
WRITE ON n: or PRINT ON n:
followed by a list of numbers, variables, or expressions whose values are to be written on the file, where n is the output file number. For example,

```
80 OPEN /DATA1/,OUTPUT,3
85 OPEN /DATA2/,BINARY OUTPUT,N
90 WRITE ON 3:P,Q,R,W
95 WRITE ON N:P-Q,N↑W,A
```

Line 90 writes the values of the variables P,Q,R and W on the symbolic file /DATA1/ (file 3). Line 95 writes the values of the expressions P-Q and N↑W and the variable A on the binary file /DATA2/ (file number equal to the value of N).

Closing A File

After the last input or output operation is performed on a data file, the CLOSE command should be used to close the file. NOTE: An input or output file is closed automatically after a RUN, a DELETE ALL, or a return to the EXECUTIVE.

Files to be closed are specified by their file numbers in the CLOSE command. For example,

```
120 CLOSE 1,B-2 closes files 1 and B-2
200 CLOSE 3     closes file 3.
```

Once a file has been read or written, it can be reread or rewritten only by closing the file and opening it again.

If three files are open concurrently, any of them may be closed with a CLOSE command so that other files can be opened.[14.5]

14.5 "TELETYPE" (or "TEL") may be used in the OPEN statement to refer to the terminal without being considered an open file. Three files can be open concurrently with the terminal.

Example

Twelve numbers are read from a file named /XDATA/. The positive numbers are written on /POSX/, the negative are written on /NEGX/.

```
-COPY /XDATA/ TO TEL

1,16,-4,6,-11,-2,30,-4,6,8,13,-7

-SBASIC

>10 OPEN /XDATA/, INPUT, 1
>20 OPEN /POSX/, OUTPUT, 2
>30 OPEN /NEGX/, OUTPUT, 3
>40 FOR I=1 TO 12
>50 INPUT FROM 1: X
>60 IF X>0 THEN WRITE ON 2: X;
    ELSE WRITE ON 3: X;
>70 NEXT I
>80 CLOSE 1,2,3
>RUN

>QUIT

-COPY /POSX/ TO TEL

 1    16    6    30    6    8    13
-COPY /NEGX/ TO TEL

-4   -11   -2   -4   -7
-
```

E. Additional Functions

Additional SUPER BASIC Functions

INT(X) or IP(X)

The integer function is INT(X) or IP(X) where, as with other functions, X can be any expression. This function yields the greatest integer not exceeded by X. Thus,
INT(7.8)=7
INT(-2.4)=-3

INT(X+.5) may be used for rounding any expression X to the nearest integer, such as

INT(7.8+.5)=8
INT(-2.4+.5)=-2

The following use of the INT function will round an expression X to N decimal places:
INT(10↑N*X+.5)/10↑N
For example,
INT(10*X+.5)/10 will round to 1 decimal place.
INT(100*X+.5)/100 will round to 2 decimal places.

FP(X)

The fractional part of X is defined as follows:
FP(X)=X-INT(X)
Thus,
FP(8)=0
FP(123.456)=.456
FP(-1.8)=.2 [-1.8-(-2)]

FIX(X)

The function FIX(X) is defined as SGN(X)*INT(ABS(X)). It truncates the value of the expression X as follows:
FIX(7.8)=7
FIX(-2.4)=-2

Whatever follows the decimal point is dropped. Note that FIX(X) is equivalent to INT(X) for positive numbers, but not for negative numbers; for example, FIX(-2.4)=-2, but INT (-2.4)=-3.

RND(X)

The RND function is a pseudo random number generator and requires a single argument that may be zero, positive, or negative. The random number will be between 0 and 1 exclusive.

If the argument is zero, the first use of the function in a program always will produce the same number. When RND(0) is used again in the same program, the next random number in sequence is given. NOTE: Another form of RND(0) is simply RND.

If the argument is positive, a random number is generated from this number. Thus, RND(16) always will produce the same number, which will be different from the number RND(30). A sequence of random numbers can be initiated by RND with a positive argument and then RND(0) (or RND) can be used repeatedly to generate the next random numbers in the sequence.

If the argument is negative, a random number is generated from a random number (set by reading the internal clock of the computer in milliseconds). The value of the negative argument has no bearing on the random number it generates; for example, RND(-1) used twice in a program will yield different random numbers which depend only on the reading of the internal clock. Thus, using RND with a negative argument to initiate a sequence of random numbers will produce a different sequence of numbers each time the program is run.

Example

```
10 PRINT RND(-1);
20 FOR I=1 TO 9
30 PRINT RND;
40 NEXT I
```

If this program is run twice, two different sequences of random numbers will be printed. However, if the argument of the RND function in line 10 were changed to 0 (or no argument) or to a positive number, running the program twice would yield the same sequence of random numbers.

SGN(X)

The sign function SGN(X) yields 1 if the value of the argument X is positive, 0 if X is equal to 0, and -1 if X is negative. Thus,

SGN(31)=1
SGN(0)=0
SGN(-.2387)=-1

POS and POS(X)

The function POS can have either no argument or one argument.

When no argument is given, the function specifies the position on the terminal at which the print head is located.

Example

```
>10 FOR I=1 TO 10
>20 READ X
>30 PRINT X;     !CONCATENATED ZONES
>40 IF POS>15 THEN PRINT
>50 NEXT I
>60 DATA 10,20,30,40,50,60,70,80,90,100
>RUN
 10 20 30 40 50
 60 70 80 90 100

>
```

As specified in line 40, a Carriage Return is printed after the print head passes position 15.

The POS function is used with an argument only when writing on files.[15] The argument is the file number of the output file.

[15] When POS(X) is used in writing on binary files, it specifies the word position (where a word is considered to be three characters).

Example

```
>10 OPEN /XX/, OUTPUT, 2
>20 FOR I=1 TO 10
>30 READ X
>40 PRINT ON 2: X:
>50 IF POS(2)>15 THEN PRINT ON 2:
>60 NEXT I
>70 DATA 10,20,30,40,50,60,70,80,90,100
>RUN

>QUIT

-COPY /XX/ TO TEL

 10 20 30 40 50
 60 70 80 90 100

-
```

Programmer Defined Functions

In addition to the standard SUPER BASIC functions, the user may define any other function which he expects to use a number of times in a program. The indirect command DEF is used for this purpose. The names of programmer defined functions must contain three letters, the first two of which must be FN. The form of the DEF statement is shown below; the programmer defines a function which will calculate the sine of an angle in degrees.

```
10 DEF FNS(X)=SIN(X*PI/180)
```

NOTE: The same name cannot be given to more than one programmer defined function in a single program. If two DEF statements are used with the same function name, the second statement, when executed, will redefine the function.

An argument used in defining a function (X in the above example) is called a parameter. A programmer defined function can have either no parameters or any number of parameters (separated by commas and enclosed in parentheses). Parameters are "dummy" arguments; that is, when a defined function is used, certain specified values will replace temporarily the parameters where they appear in the function definition. For example,

```
>10 DEF FND(A,B)=4*A*B+A↑2
>20 Y=FND(2,1)
>30 PRINT Y
>RUN
 12
>
```

When the defined function was used in line 20, 2 and 1 replaced A and B respectively in the function definition in line 10. Thus, Y was set to $(4x2x1)+2^2$, or 12.

Parameters can have any variable name, including the names of variables used in the same program; in other words, the parameters are local to the function definition. Continuing from the above example, if the lines

```
>5 A=6,B=4
>35 PRINT A,B
```

are written into the program, the A and B parameters still will be replaced by 2 and 1 (as specified in line 20). Once the function has been evaluated however, A and B are restored to their former values as assigned in line 5. Therefore, line 35 will print 6 and 4 as the values of A and B.

Any variables in a function definition which are not parameters of that function simply take the values assigned to them in some previous part of the program; that is, these variables are global. For example, consider the following defined function:

```
35  DEF FNK=6.21083*R↑2+W
```

When the function FNK is used, the variables R and W must have been assigned values previously; these values will be used in evaluating FNK.

When a defined function with parameters is used in a program, any argument (number, variable, or expression) can replace the parameters in the definition. For example, the following is permitted:

```
60 DEF FNP(X,Y,Z)=X/2-4*Y*Z+Z↑2
65 B=FNP(3,Q,R↑3)
```

When line 65 is executed, the parameters X,Y, and Z are set temporarily to the values of 3,Q, and R^3.

The defining expression in a DEF statement may include other programmer defined functions as well as parameters, program variables, and standard functions. For example,

```
40 DEF FNR(A)=TAN(B)+A↑2/W
50 DEF FNF(X,Y,Z,K)=2*Y*Z+LOG(X)-FNR(K)
60 G=FNF(M,N,P,Q)
```

In this example, the DEF statement on line 50 calls for another function previously defined by the programmer; namely, FNR on line 40. When line 60 is executed, the current values of M,N,P, and Q will be transferred directly to the defining expression of line 50. The value of G will be set to

```
2*N*P+LOG(M)-TAN(B)-Q↑2/W
```

Note that when a DEF statement uses one or more previously defined functions, it is possible that parameters will be listed which do not appear directly in the defining expression. For example,

```
100 DEF FNY(Q)=A+6*EXP(Q)
105 DEF FNZ(A)=FNY(2)↑B
110 M=FNZ(5)
```

The parameter A of the function FNZ does not appear in the defining expression, but it specifies that when FNY(2) is

evaluated as a part of that function, A will be replaced temporarily by the argument of FNZ (5 in line 110). Thus A is local to FNZ even though it is global (assigned the value of the program variable A) in the function FNY.

Using DEF is limited to those cases in which the value of the function can be computed within a single statement. Often more complicated functions, or even pieces of a program, must be calculated at several points within the program. In this case, the user would more likely use a subroutine (see Section 2(G)).

F. Subscripting And Array Manipulation

So far, variables have been described as being able to store one value. There are times however, when the user will want to store a set of values in a list or table which he can refer to by a single name. This is done by using subscripted variables to designate elements of such lists or tables, which are called arrays. A variable may have any number of subscripts; in other words, SUPER BASIC allows arrays of any dimension (each subscript representing a dimension).

Subscripts are typed in parentheses after the variable name. For example, A(7) refers to the seventh item in a list (a one-dimensional array, or vector) named A, and B(3,7) denotes the element in the third row and seventh column of a table (a two-dimensional array, or matrix) named B.

Subscripted Variable Names

The name of a subscripted variable must be a single letter or a single letter followed by a $ (dollar sign). The variable name used for a subscripted variable also may be used to denote a simple variable in the same program. FOR example, A and A(1) are considered to be separate variables. However, the same name cannot be given to arrays of different dimensions in the same program; for example, A(1) and A(3,7) are not allowed in the same program.

Subscripts

Subscripts may be variables (including other subscripted variables) or expressions of any complexity. The following subscripts are acceptable:

```
A(5) C(1+K) F(5,30) R(B(3,J),C-D) X(A*B,20)
```

Subscripts may have any value, including negative and non-integer. If the value of a subscript is non-integer, SUPER BASIC will truncate the value.

Size Of Arrays

SUPER BASIC automatically supplies space for subscripts 1 to 10 for arrays of one or two dimensions. Therefore, a vector named A containing 10 elements could be entered simply with the statements

```
10 FOR I=1 TO 10
20 READ A(I)
30 NEXT I
40 DATA 2,3,-5,7,2.2,4,-9,123,4,-4
```

The DIM Command

If an array is to have a subscript greater than 10 or have more than two dimensions, the size of the array must be specified by the DIM command which can be executed directly or indirectly. This command instructs SUPER BASIC to reserve a specified amount of space for array elements. For example,

```
10 DIM A(15)
```

will reserve 15 spaces for elements A(1) to A(15). The DIM statement does not define any array elements; it simply allows a certain number of values to be accepted as input to the array.

Any number of arrays can be dimensioned in a single DIM statement as follows:

```
60 DIM K(20),L(3,3,1),M(A*B),N(X,3,3,2)
```

The user may save storage space by dimensioning arrays with subscripts less than 10, even though such dimensioning is not required. For example, DIM E(3,5) will reserve space for exactly 15 elements, whereas without the DIM statement, 100 (10x10) spaces would be reserved for the array E. NOTE: Whatever the maximum subscript value, arrays of three or more dimensions require a DIM statement.

Subscripts start from 1 unless otherwise specified. One way to specify a different subscript base is with the following form of the DIM command:[16]

```
10 DIM A(0:15)
```

This statement will reserve space for elements A(0) to A(15). The user may specify that a subscript start from any number. For example,

```
DIM B(5:10)          reserves space for A(5) to A(10).
```

---

[16] A second method of specifying a base other than 1 uses the BASE command, described below.

```
DIM B(-6:10,-2:4)    starts subscripts at negative values; the
                     0th elements are included.
```

The user may redimension an array at any time by using another DIM statement. Note however, that redimensioning (or executing the same DIM statement a second time) causes any existing elements in an array to be cleared (that is, be undefined). For example,

```
>10 DIM X(20)          The array X is dimensioned.
>20 X(1)=3,X(2)=7      Two elements are defined.
>30 DIM X(0:20)        X is redimensioned to include the
                       0th element.
>RUN                   The above statements are executed.
>PRINT X(1),X(2)
SUBSCRIPTED VARIABLE HAS NO VALUE    X(1) and X(2) have
>                                    been undefined.
```

The BASE Command

Another method of specifying that subscripts start with some number other than 1 is by using the BASE command which can be executed directly or indirectly. The form of this command is

```
BASE n
```

where n can be any numeric expression. BASE applies only to arrays which have not yet been dimensioned, and will cause the subscripts of those arrays to begin from n unless:

- The lower limit of a subscript is specified in a DIM statement, such as DIM A(-2:5), or
- Another BASE command is given which specifies a different base.

For example,

```
5 BASE 0
10 DIM A(15),B(-2:2,10)
```

will cause the A subscript and the second B subscript to start at 0. Suppose that the following statements were added to the above:

```
15 BASE 1
20 DIM C(3)
```

The dimensions of arrays A and B would not be affected; the subscript of array C would begin at 1 and not 0.

NOTE: A BASE command executed previous to a RUN will not be ignored when the RUN command is given.

Matrix Operations

Although the user may write his own routines for matrix operations, SUPER BASIC contains a set of commands which make calculations involving matrices or vectors considerably easier. All of these commands begin with the word MAT, and many of them are similar in form to the ordinary SUPER BASIC instructions. NOTE: The MAT commands apply only to arrays of one or two dimensions. Any attempt to use them with multi-dimensional arrays will cause an error message to be printed.

1. Input Of Matrix Data

The following input commands do not require that the specified matrices or vectors be dimensioned before the commands are given. A matrix or vector that has not been dimensioned previously however, must be dimensioned in the MAT command itself (see details below). NOTE: This rule applies in all cases, even if the subscript value will not exceed 10. SUPER BASIC must know when to stop accepting data for input.

MAT READ

```
MAT READ A,B,C
```

will read values into the previously dimensioned matrices (or vectors) A,B and C from the data block defined in the DATA statements of a program. Any number of matrices can be read with a single MAT READ instruction.

It is possible to use the MAT READ statement itself to dimension a matrix or vector which has not been dimensioned previously (or to redimension one which already has). In this case, simply type the dimensions of the arrays just as they would be typed in a DIM statement. For example,

```
65 MAT READ K(15),L(-1:1,3),M
```

will read values into a 15 element vector K, a 3 by 3 matrix L (with the first subscript ranging from -1 to 1), and a previously dimensioned matrix M.[17] This statement is exactly equivalent to

```
65 DIM K(15),L(-1:1,3)
70 MAT READ K,L,M
```

Matrices are read in row order; that is, the second subscript varies more rapidly. For example,

```
10 MAT READ A(4,3)
```

---

[17] It is assumed here, and in the remaining examples in this section, that no BASE command has been given previously, so that subscripts start from 1 unless otherwise specified.

is equivalent to

```
10 FOR I=1 TO 4
20 FOR J=1 TO 3
30 READ A(I,J)
40 NEXT J,I
```

In both cases, values will be read from the DATA statements in the following order: A(1,1),A(1,2),A(1,3),A(2,1),...,A(4,2), A(4,3).

MAT INPUT

The MAT INPUT command performs the same function for matrices and vectors as the INPUT command does for variables; SUPER BASIC prints a question mark and waits for the data to be typed from the keyboard. Matrix values should be typed in the same order that they would be read by a MAT READ statement; that is, in row order (with the second subscript varying more rapidly).

The form of the MAT INPUT command is similar to MAT READ in that the matrices or vectors may be dimensioned either previously or in the MAT statement itself.

Also included in SUPER BASIC is a MAT INPUT FROM command corresponding to the INPUT FROM command for reading data from a file.

Example 1

```
MAT INPUT A(2,3)
```

will cause SUPER BASIC to wait for six values to be typed, in the order: A(1,1),A(1,2),A(1,3),A(2,1),A(2,2),A(2,3).

Example 2

```
10 OPEN /MATDATA/,INPUT,1
20 MAT INPUT FROM 1: A(2,3)
```

accepts six values from /MATDATA/ as input to the matrix A.

Example 3

```
10 DIM F(5),G(4,4)
●
●
●
95 MAT INPUT F,G(4,X),H(7,7)
```

Vector F and matrix G are dimensioned in line 10. Statement 95 redimensions matrix G, dimensions a new matrix H, and requests data for F,G and H.

2. Output Of Matrix Data

MAT PRINT

A command of the form

MAT PRINT A,B,C

can be executed directly or indirectly to print the matrices (or vectors) A,B, and C. Every element of A,B, and C must have a value.

Matrices are printed row by row. The elements of each row are printed in normal (15 space) print zones unless the matrix name is followed by a semicolon or a colon in the PRINT statement. A semicolon after a matrix name will cause the elements of each row to be printed in packed zones; a colon will cause concatenated print zones. Each row is separated from the next by a blank line.

Example 1

```
>10 MAT INPUT F(2,3)
>20 PRINT
>30 MAT PRINT F;
>RUN
? 1,2,3,4,5,6

 1      2      3

 4      5      6
```

Example 2

```
MAT PRINT R;S,T;
```

will print R and T in packed zones and S in normal zones. NOTE: If the semicolon after T were omitted, a comma would be understood and T would be printed in normal zones also.

SUPER BASIC one-dimensional arrays are column vectors and therefore will be printed vertically. A row vector (consisting of one row instead of one column) can be dimensioned as, for example, V(1,N), which would set up a 1 row, N column array and therefore print the N elements of the array horizontally.

The MAT PRINT ON (or MAT WRITE ON) command corresponds to the PRINT ON (or WRITE ON) command for writing data on a file.

3. Mathematical Operations With Matrices

All of the following operations require that the solution matrix or vector be dimensioned properly before the operation is performed. For example, the statement MAT C=A+B will add the

matrices A and B and store the result in matrix C; C must be dimensioned properly before this statement is executed (even if neither subscript exceeds 10).

Only one mathematical operation with matrices may be performed per statement. Thus, MAT X=R+S+T is not allowed, but can be achieved by two MAT instructions.

Each of the following statements can be executed directly or indirectly.

Matrix Addition

```
MAT C=A+B
```

A statement of this form adds the matrices (or vectors) A and B and stores the result in C. A,B and C all must be of the same dimensions for this statement to be executed.

Matrix Subtraction

```
MAT C=A-B
```

This statement subtracts the matrix (or vector) B from the matrix (or vector) A and stores the result in C. A,B and C must have the same dimensions.

Matrix Multiplication

```
MAT C=A*B
```

In order for this statement to be executable, A and B must be "conformable"; that is, they must be of such dimensions that their product is defined. In addition, C must be dimensioned properly to contain the result. NOTE: This instruction applies to matrices only. Multiplying vectors is not permitted in SUPER BASIC at this time.

Scalar Multiplication

```
MAT C=(n)*A
```

This statement performs scalar multiplication; that is, each element of the matrix (or vector) A is multiplied by the number (or numeric expression) n (which must be enclosed in parentheses) and stores the result in C. C must be the same dimension as A. NOTE: The instruction MAT C=(1)*A may be typed simply as MAT C=A.

Matrix Transposition

```
MAT C=TRN(A)
```

This statement transposes the rows and columns af A and places the result in C; it is equivalent to letting C(I,J)=A(J,I) for all values of I and J. C and A need not be square; an M by N matrix will be transposed into an N by M matrix. NOTE: This instruction applies to matrices only. Vector transposition presently is not permitted in SUPER BASIC.

Matrix Inversion

```
MAT C=INV(A)
```

This statement inverts the square matrix A (using the Gauss-Jordan method with complete matrix pivoting) and stores the result in C. SUPER BASIC will print an error message if the matrix to be inverted is singular or nearly so (that is, "ill-conditioned", so that it is difficult to invert accurately). The determinant of the matrix is inspected internally; an inverse will be given only if the value of the determinant is large enough to produce a meaningful inverse.

NOTE: The same matrix may appear on both sides of a MAT statement for addition, subtraction, scalar multiplication, or inversion, but not in any of the other instructions. Thus,

```
MAT A=A+B
MAT A=(2.5)*A
MAT A=A-B
MAT A=INV(A)
```

are all legal, while use of

```
MAT A=B*A
MAT A=TRN(A)
```

will result in nonsense.

4. Matrix Initialization

Setting All Elements To Zero

```
MAT C=ZER
```

This instruction sets all elements of the previously dimensioned matrix (or vector) C to zero. It can be used also to dimension (or redimension) a matrix or vector and initialize all elements to zero. Thus, the statement

```
MAT C=ZER(M,N)
```

sets up an M by N matrix C, where C need not be dimensioned pre-

viously, and fills the matrix with zeroes. An instruction of the form

MAT C=ZER(M)

performs a similar function for an M element vector.

Setting All Elements To One

MAT C=CON

This instruction is similar in form and function to MAT C=ZER, except that the matrix (or vector) is filled with ones instead of zeroes. It can be used also to dimension (or re-dimension) a matrix or vector, in the form

MAT C=CON(M,N) or

MAT C=CON(M)

Setting An Identity Matrix

MAT C=IDN

This statement sets the previously dimensioned square matrix C equal to an identity matrix; that is, a matrix with ones on the main diagonal and all other elements equal to zero. It can be used also to dimension (or redimension) a matrix, in the form

MAT C=IDN(M,M)

5. Example Of Matrix Operations

This program reads the dimensions and values of matrices A and B from DATA statements. A,B, and A*B are printed, then A*B with one element changed.

```
>LIST
10 READ M,N
20 MAT READ A(M,N),B(N,N)
30 MAT PRINT A;B;   !NOTE THE FORMATS
40 DIM C(M,N)
50 MAT C=A*B
60 MAT PRINT C;
70 C(1,3)=99     !ONE ELEMENT CHANGED
80 MAT PRINT C
90 DATA 2,3
100 DATA 1,2,3,4,5,6
110 DATA 1,0,1,-2,1,-1,0,2,3
>RUN
 1  2  3

 4  5  6
```

```
 1     0     1

-2     1    -1

 0     2     3


-3     8     8

-6    17    17


-3              8              99

-6             17              17


>
```

G. Subroutines

When a part of a program is repeated several times in different places, it can be programmed more efficiently as a subroutine. Subroutine statements are written only once but can be used many times from any place in the main program.

GOSUB And RETURN

The command used to transfer to a subroutine may be executed directly or indirectly. Its form is GOSUB followed by the line number of the first statement of the subroutine. The GOSUB command is similar to GO TO followed by a line number in that it transfers unconditionally to another part of the program. GOSUB differs in that it will not go beyond the end of the subroutine, which must be indicated by a RETURN command. If the GOSUB command was executed indirectly, the return will be to the statement following the one in which the GOSUB command was given. If GOSUB was executed directly, SUPER BASIC will simply stop when it reaches the end of the subroutine.

The following example of a small subroutine shows two sections of the main program in which the GOSUB command is used.

```
10 S=3
20 GOSUB 400
30 PRINT H,P,X
●
●
●
100 S=7
110 GOSUB 400
120 Z=3*H+P/X
```

```
•
•
•
400 H=S*SQR(2),P=2*S+H
410 IF P<=10 THEN X=1 ELSE X=2
420 RETURN
•
•
•
```

When this program is run, line 20 instructs SUPER BASIC to transfer to the subroutine beginning at line 400. When the RETURN command at the end of the subroutine is reached, a return is made to line 30 (the line following the GOSUB command). Similarly, when the subroutine is called later from line 110, the return will be to line 120.

As an example of the GOSUB command used directly, suppose that the above program has been loaded into SUPER BASIC. A direct GOSUB can be used to execute only the subroutine for a particular value of S as shown below.

```
>S=4
>GOSUB 400
>
```

A GO TO or an IF statement within a subroutine can cause transfer out of the subroutine before the RETURN command is reached. In addition, a subroutine can contain a GOSUB statement which calls either another subroutine or itself.

Example 1

```
•
•
•
40 X=SIN(Y+Z)
50 GOSUB 200
60 PRINT X
•
•
•
200 Q=X+R/S
210 IF Q<.5 THEN RETURN
220 PRINT "Q=";Q
230 GOSUB 500
240 RETURN
•
•
•
```

```
500 V=Q+R/S
510 PRINT "V =";V
520 RETURN
●
●
●
```

The subroutine beginning at line 200 contains both an IF... THEN... statement and a GOSUB command which calls another subroutine. As specified in line 210, if Q<.5, a return will be made (to line 60). If Q>=.5, the program will continue with the next statements in order until it reaches the GOSUB 500 command. A transfer is then made to the subroutine beginning at line 500. Note the effect of the RETURN commands in this program: Line 520 causes a return to line 240, which in turn causes a return to line 60 (the statement following the GOSUB 200 command).

Example 2

```
10 INPUT A
20 IF A<>0 THEN GOSUB 1000
30 B=1/COS(A)
●
●
●
1000 A=1/SIN(A/3)
1010 IF A>0 THEN RETURN
1020 GOSUB 1000
1030 RETURN
●
●
●
```

Line 20 instructs SUPER BASIC to execute the subroutine beginning at line 1000 if A is not zero. The specified subroutine assigns a new value to A (on line 1000), and a return is made to line 30 if A is positive. If A is not positive, the GOSUB 1000 command in line 1020 is executed. The subroutine will continue to call itself in this way until A is positive. Then a return will be made to line 1030, which in turn causes a return to line 30.

Note that a subroutine which calls itself must contain at least one condition on which a transfer out of the subroutine can be made (such as line 1010 above); otherwise, an infinite loop will result.

Isolating Subroutines

Subroutines must be isolated from the main program; this is not done automatically by SUPER BASIC. The sequence of steps in the program should be designed so that the statements of the subroutine are executed only after a GOSUB command.

STOP or END

Either of the indirect commands STOP or END may be used to isolate subroutines. These commands cause execution of the program to terminate. All subroutines can be placed at the end of the main program and separated from the main program by a STOP or END statement as illustrated below:

```
10! MAIN PROGRAM BEGINS
●
●
●
100 GOSUB 700
●
●
●
690 STOP !MAIN PROGRAM ENDS
700 !SUBROUTINE BEGINS
●
●
●
790 RETURN !SUBROUTINE ENDS
```

NOTE: A STOP or END statement may be used anywhere in a program to terminate execution. Remeber that no such command is required at the end of an entire program, since SUPER BASIC stops automatically as soon as there are no more statements to be executed.

Computed GO TO And GOSUB Statements

The computed GO TO and computed GOSUB statements, which may be executed directly or indirectly, cause transfer to one of several different parts of a program depending on the value of a specified expression.

ON...GO TO...

The form of the computed GO TO statement is
ON expression GO TO $line_1$, $line_2$,...
where $line_1$, $line_2$ ... is a sequence of line numbers to which the program will transfer depending on the value of the expression. If the value of the expression is 1, the program will transfer to $line_1$; if the value of the expression is 2, the program will transfer to $line_2$, and so on. For example,

```
ON I*J GO TO 60,70,85
```

will transfer to lines 60,70 or 85 depending on whether the value of the expression I*J is 1,2, or 3 respectively.

If the value of the expression is less than one or greater than the number of line numbers, an error message will be

printed. If the value of the expression is not an integer, the value will be truncated.

ON...GOSUB...

The form of the computed GOSUB statement is
ON expression GOSUB $line_1$, $line_2$,...
If the value of the expression is 1, the program will transfer to the subroutine starting on $line_1$; if the value is 2, the transfer will be to the subroutine starting on $line_2$, and so on. After the subroutine is executed, the program returns to the next statement in order after the computed GOSUB statement.

Example

```
>10 FOR A=1,2,3
>20 ON A GOSUB 100,200,300
>30 PRINT "NEXT"
>40 NEXT A
>50 STOP
>100 PRINT "SUBROUTINE AT 100, A =":A
>110 RETURN
>200 PRINT "SUBROUTINE AT 200, A =":A
>210 RETURN
>300 PRINT "SUBROUTINE AT 300, A =":A
>310 RETURN
>RUN
SUBROUTINE AT 100, A = 1
NEXT
SUBROUTINE AT 200, A = 2
NEXT
SUBROUTINE AT 300, A = 3
NEXT

>
```

H. Logical Variables, Expressions And Operators

Logical Variables And Expressions

Every variable in SUPER BASIC is considered to have, in addition to a numeric value, a logical value which is either TRUE or FALSE.[18] The logical value of a variable is defined as TRUE if the numeric value is not zero, and FALSE if the numeric value is zero.[18.5] For example,

---

[18] Since they do not have numeric values, string variables (which are discussed in section 2(J)) do not have logical values.

[18.5] If the variable is complex, its logical value is set to the logical value of its real part.

Numeric Value	Logical Value
A=0	A is FALSE
B=18	B is TRUE
C=-7	C is TRUE

Thus, a single variable can be used as the condition in an IF statement as follows:

```
10 IF X THEN 200
```

This statement specifies that if X is TRUE (not zero) the program will transfer to line 200. If X is FALSE (zero), the program will continue with the next statement in order.

More commonly used in the IF...THEN... statement to specify a condition is an expression containing one of the relational operators (<,<=,=,>=,>,<>or #). Note that a relational expression must have one of the logical values TRUE or FALSE and can, therefore, be considered as a logical expression. For example,

```
30 IF S=0 THEN 70
```

causes a transfer to line 70 if the expression S=0 is TRUE, and no transfer if S=0 is FALSE.

SUPER BASIC stores the logical value of an expression as either 1 or 0. A TRUE expression is set to 1 and a FALSE expression is set to 0. For example,

Expression	Logical Value
A=B	1 (for TRUE) if A=B, 0 (for FALSE) if A<>B
C<D↑2	1 if C<D↑2, 0 if C>=D↑2

Thus

PRINT A=B	prints 1 if A=B, prints 0 if A<>B
Z=C<D↑2	sets Z=1 if C<D↑2, sets Z=0 if C>=D↑2
X=Y=5	sets X=1 if Y=5, sets X=0 if Y<>5

Declaring Logical Variables

If a variable is declared to be a logical variable, it will be set to its logical value (1 for TRUE or 0 for FALSE) and not to its numeric value. To set a variable to its logical value, simply type the variable name (or names, separated by commas) in a LOGICAL statement which can be executed either directly or indirectly. For example,

```
>10 LOGICAL A,B
>20 A=18,B,C=6
>30 PRINT "A =";A;" AND B =";B
>40 PRINT "BUT C =";C
>RUN
A = 1 AND B = 1
BUT C = 6

>
```

Since A and B were declared logical, their logical values were printed. Because 18 and 6 are non-zero (that is, TRUE), the logical value of both A and B was printed as 1.

The LOGICAL statement also can be used to declare that an array will store logical values. As it is declared, the array is dimensioned exactly as it would be in a DIM statement. No previous dimensioning is necessary. For example,

```
10 LOGICAL X(10),Y(4,N)
```

reserves space for a 10 element logical array X, and a 4 by N logical array Y.

Logical Operators

In SUPER BASIC there are five logical operators which operate on logical variables and expressions. The result of a logical operation is a logical expression which is either TURE (1) or FALSE (0).

The results of using logical operators where A and B are logical variables or expressions are shown in the following table:

T = True F = False

Operator	A B	T T	T F	F T	F F
AND	A AND B	T	F	F	F
OR	A OR B	T	T	T	F
EQUIVALENCE	A EQV B	T	F	F	T
IMPLICATION	A IMP B	T	F	T	T
NOT	NOT A:	If A is True, then NOT A is False If A is False, then NOT A is True.			

Some examples of logical expressions containing logical operators are:

```
A AND NOT B
X=3 OR X=5
E*5>A-B OR E<=100
A<>2*EXP(5) AND I=J
```

Note that a logical operator works only with the logical value of what is on either side of it. Thus, X=3 OR X=5 may not be typed as X=3 OR 5. The 5 will be considered to be true, since it is a non-zero value. Therefore, whatever the value of X, the expression X=3 OR 5 always will be true. The correct form of the expression will operate as follows:

```
75 IF X=3 OR X=5 THEN NEXT X
```

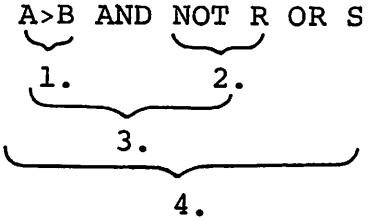
If the value of X is 3 or 5, the expression is true and the THEN statement will be executed. If the value of X is neither 3 nor 5, the expression is false and the program will go on to the next line.

The order of priority among the different types of operators in SUPER BASIC is as follows, in descending order:

1. Expressions in parentheses
2. Evaluation of functions
3. Exponentiation (↑)
4. Unary minus (-)
5. MOD
6. Multiplication and division (* and /)
7. Addition and subtraction (+ and -)
8. Relational operators (<,<=,=,>,>=,<>or #)
9. NOT
10. AND
11. OR
12. IMP
13. EQV

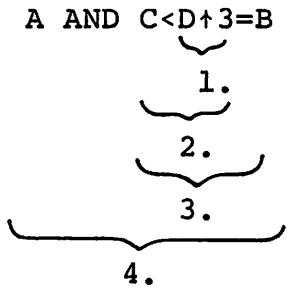
For example, the following logical expressions are evaluated in the indicated order.

Example 1



1. Relational operator>
2. Logical operator NOT
3. Logical operator AND
4. Logical operator OR

Example 2



1. Exponentiation↑
2. First relational operator<
3. Second relational operator =
4. Logical operator AND

I. Statement Modifiers

The number of statements in a program can be reduced greatly by using statment modifiers. One or more modifiers may be appended to most direct statements and to all indirect statements except DATA. The statement modifiers are IF, UNLESS, FOR, WHILE, and UNTIL.

IF And UNLESS

The IF modifier followed by a logical expression causes the command to which the IF clause is appended to be executed if the logical expression is true. The command is not executed if the logical expression is false. For example,

PRINT X IF X>0	The value of X will be printed only if X is positive.
GO TO 100 IF B	If B is not equal to zero; that is, true, the program will transfer to line 100. If B is zero; that is, false, no transfer will be made.

Other examples using the IF modifier are:

```
30 INPUT N IF M<=SQR(7)
55 BASE I IF I<>1
100 NEXT X IF G2=0
R=S IF Q>100
```

The UNLESS modifier followed by a logical expression causes the command to which the UNLESS clause is appended to be executed if the logical expression is false. The command is not executed if the logical expression is true. For example,

```
PRINT X UNLESS X>0
```

The value of X will be printed only if X is not positive.

Other examples using the UNLESS modifier are:

```
15 GOSUB 100 UNLESS X=0
130 A=B↑2 UNLESS A=C
200 PRINT ON 2:Z UNLESS I<J
GO TO 55 UNLESS V*W=1
```

FOR

FOR causes the command to which it is appended to execute repeatedly over a range of values. The FOR clause takes the same form as the FOR statement used in defining loops. For example,

```
>INPUT A(I) FOR I=1 TO 4
 ? 6,-4,3,2
>
```

The command INPUT A(I) is executed repeatedly from the initial value of I to the final value of I (in steps of 1, since there is no STEP or BY clause).

```
>PRINT X-2 FOR X=5,15,-9
 3
 13
-11
```

The command PRINT X-2 is executed for each value of X listed.

```
>PRINT X FOR X=1 TO 6 STEP 2
 1
 3
 5
>
```

The command PRINT X is executed repeatedly from the initial value of X to the final value of X, in steps of 2.

WHILE And UNTIL

WHILE followed by a logical expression causes the command to which the WHILE clause is appended to be executed repeatedly as long as the logical expression is true. WHILE often is used with the FOR modifier (or the FOR statement) in place of the TO clause as a means of specifying the final value. For example,

```
>PRINT A↑2 FOR A=1 WHILE A<4
 1
 4
 9
```

The command PRINT A↑2 is executed repeatedly from the initial value of A (in steps of 1) as long as the WHILE condition (A<4) is true.

```
>X=2*X WHILE X<Y
```

X is reset to the value of 2*X repeatedly as long as X is less than Y. For example, if X were 1 initially and Y were 17, X would be reset to 32, since the last value of X to be multiplied by 2 would be 16.

UNTIL followed by a logical expression causes the command to which the UNTIL clause is appended to be executed repeatedly as long as the logical expression is false. UNTIL may be used with FOR in a similar manner as WHILE. For example,

```
>PRINT A↑2 FOR A=1 STEP 2 UNTIL A>5
 1
 9
 25
```

The command PRINT A↑2 is executed repeatedly from the initial value of A (in steps of 2) as long as the UNTIL condition (A>5) is false. NOTE: the STEP clause could not have been typed at the end of this statement.

```
>X=2*X UNTIL X>=Y
```

X is reset to the value of 2*X repeatedly until X is greater than or equal to Y. This statement is equivalent to the WHILE modifier above.

An example of using WHILE or UNTIL in a FOR statement is

```
50 FOR X=1 WHILE X<=Y
```

which is equivalent to

```
50 FOR X=1 UNTIL X>Y
```

The subsequent FOR loop will be executed from the initial value of X in steps of 1 as long as X is less than or equal to Y. Note that X always will be compared to the current value of Y, even if the value of Y should change within the loop; this is not true when the more common form of the FOR statement is used. For example, when

```
50 FOR X=1 TO Y
```

is encountered for the first time, the final value of X is set permanently to the value of Y at that time. Any changes of Y within the loop will not change this final value.

A modified indirect statement can be included in a THEN or an ELSE clause as any other indirect statement. For example,

```
IF Z THEN A(I)=B(I) FOR I=1 TO 10 ELSE J=J↑3 WHILE J<N
```

FOR modifies only the statement A(I)=B(I) in the THEN clause; WHILE modifies only the statement J=J↑3 in the ELSE clause.

More than one modifier can be used to modify a single statement. The last modifier will be considered first, the next to the last modifier will be considered next, and so on.

Example 1

```
 85 GO TO 105 IF A=B UNLESS N=0
```

When this statement is executed, the condition N=0 is checked first. If N is zero, the command GO TO 105 will not be executed. If N is not zero, the condition A=B is considered. If A and B are equal, the program will transfer to line 105.

Example 2

```
PRINT Y(I) FOR I=1 TO 10 IF C(I)=P
PRINT Y(I) IF C(I)=P FOR I=1 TO 10
```

These two statements are not equivalent. The first statement first checks to see if C(I)=P with I previously defined. If this is true, the values of Y(1) to Y(10) will be printed. The second statement checks for each value of I whether C(I) is equal to P. Those values of Y(I) for which C(I)=P will be printed.

Example 3

```
 50 READ A(I,J) FOR I=1 TO 3 FOR J=1 TO 5
```

This statement is equivalent to

```
>50 FOR J=1 TO 5
>60 FOR I=1 TO 3
>70 READ A(I,J)
>80 NEXT I,J
```

First, J is set to 1 and values are read for A(1,1), A(2,1) and A(3,1); that is, for the first column of the array. Then J is set to 2 and so on, until finally, the last column is read in when J=5. If the values were to be read in row order instead of column order, the statement would be typed as

```
>50 READ A(I,J) FOR J=1 TO 5 FOR I=1 TO 3
```

J. Strings

String Variables

Instead of assigning a numeric value to a variable, the SUPER BASIC user may set a variable equal to a string of characters. String variables make it possible to accept names, addresses, mixed alphabetic and numeric identification, and similar data as input from files or from the terminal. SUPER BASIC accepts strings of any length.

A variable that is to be assigned a string value can be named in the same three ways as numeric variables: a single letter, a letter followed by a single digit, or a letter followed by $. Variable names for string arrays and arrays storing both strings and numbers can be, as for numeric arrays, a single letter or a letter followed by a $.

Assigning And Printing String Values

A string value, like a numeric value, can be assigned to a variable with either an assignment statement, an INPUT statement or a READ statement (including INPUT FROM a file, and matrix input instructions). Each string is enclosed in single or double quote marks.[19] Everything inside the quote marks is accepted except a Line Feed. A Line Feed indicates that the data is continued on the next line.

All forms of the PRINT command can be used to print strings. The effect of the comma, semicolon, and colon are the same for printing string variables as for printing any text enclosed in quote marks (explained in section 2(B)).

Example 1: Assignment, INPUT and PRINT

```
>10 A="STRING"
>20 INPUT B,C
>RUN
? "1234567","LA,999"
>PRINT A;B;C
STRING   1234567  LA,999
>
```

Although the string value of the variable B looks like a number, SUPER BASIC will not consider it as such. B will be treated as a group of characters having no numeric value.

---

[19] With exceptions when the variable is declared to be a string variable (explained below).

Example 2: READ, PRINT

```
>10 READ X,Y,Z
>20 PRINT X
>30 PRINT Y;Z
>40 DATA "FIRST STRING","SECOND","THIRD"
>RUN
FIRST STRING
SECONDTHIRD

>
```

Note that the colon in the second PRINT statement caused the values of Y and Z to be printed with no spaces between them.

Declaring String Variables

Variables or arrays can be assigned string values. This is accomplished by means of a STRING or TEXT statement, which may be executed either directly or indirectly. Although declaring string variables and arrays is not necessary, doing so will provide more efficient memory utilization and facilitate input of string values (as will be shown below).

Both arrays and non-subscripted variables can be declared in a STRING statement. As they are declared, the arrays are dimensioned exactly as they would be in a DIM statement. No previous dimensioning is necessary. For example,

```
10 STRING X,Y,A(5)
```

reserves space for array elements A(1) to A(5) and declares that the values assigned to X,Y and the array A will be strings.

A TEXT statement is used to declare string arrays only. For each array declared in a TEXT statement, the maximum number of characters of an element is specified for all elements. This maximum number may be a variable or an expression. For example,

```
20 TEXT A(12):10,B(3,5):M*N
```

reserves space for a 12 element string array A, each element of which can contain up to 10 characters, and a 15 element array B with maximum string length equal to the value of M*N.

Since dimensioning arrays declared in the STRING or TEXT statement is the same as dimensioning in a DIM statement, the following is permitted:

- Dimensions may be variables or expressions

```
50 TEXT J(Z):15,K(N+1,M+1):10
```

- The subscript base may be specified

```
70 STRING C(-1:1),D(0:5,20)
```

An array can contain both numbers and strings. In this case the array would be dimensioned in a DIM and not in a STRING or a TEXT statement since the latter declare that all data will be string values.

Assigning Declared String Variables

INPUT And READ Statements

When string variables or arrays are declared, data assigned to them by means of an INPUT or READ statement need not be surrounded by quote marks.

There are three exceptions; the following strings always must be surrounded by quote marks, even if the variable has been declared:

- A string containing a comma, such as "HART,S."
- A string containing leading spaces, such as " YES".
- In a DATA statement, a numeric string, such as "123" or "6E-3"

Example 1

```
>10 STRING Q,R,S,T
>20 READ Q,R,S,T
>30 PRINT Q:R:S:T
>40 DATA STRING,A23," SPACES ","MAY 3,1966"
>RUN
STRINGA23 SPACES MAY 3,1966

>
```

Quote marks were typed around the string " SPACES " so that its leading space would be accepted. Without the quote marks, the space would have been ignored. "MAY 3,1966" was enclosed in quotes so that embedded comma would be accepted as part of the string. Without the quote marks, SUPER BASIC would have stopped reading the value of T when it reached the comma; T would thus have been assigned the value MAY 3.

NOTE: Only commas and Carriage Returns (and not spaces) may be used to separate string values that are not surrounded by quote marks.

Example 2

```
>10 TEXT A(3):15
>20 INPUT A(I) FOR I=1 TO 3
>30 PRINT
>40 PRINT A(I) FOR I=1 TO 3
>RUN
? SMYTHE,ACCT. NO. 63794,"$1,630.75"

SMYTHE
ACCT. NO. 63794
$1,630.75

>
```

In the above example, array A is declared in a TEXT statement. The data need not be enclosed in quote marks. Quote marks were typed around the string "$1,630.75" to accept the embedded comma.

Since an array used to store both numeric and string data cannot be declared in a STRING or TEXT statement, input for string elements in such an array must be enclosed in quote marks. In the following:

```
>INPUT S(I) FOR I=1 TO 5
? 250,"A STRING",3.75,"XXX","13.69"
>
```

S(1) and S(3) are numeric variables; S(2),S(4) and S(5) are string variables.

Assignment Statement

Strings in an assignment statement must be surrounded by quote marks whether or not the string variables have been declared. For example,

```
>10 STRING A,B
>20 A="ONE"
>30 B="TWO"
>40 C="THREE"
>50 D=A
>60 PRINT A;B;C;D
>RUN
ONE   TWO   THREE   ONE

>
```

The Null String

While manipulating strings, a null string can be formed. This is the string "" , which contains no characters.

The VAR=ZERO command, which causes numeric variables to be initialized to zero, initializes string variables to the null string.

String Concatenation

Strings can be concatenated (joined together to form a new string) with a + sign, as illustrated below.

```
>10 X="XXX"
>20 Y="YYYY"
>30 A=X+Y
>40 B=X+"DEF"+Y
>50 PRINT A;B
>RUN
XXXYYYY  XXXDEFYYYY

>
```

Strings cannot be concatenated with numeric expressions; an error message will result.

A String Expression In The OPEN Statement

One particularly useful feature of the OPEN statement involves string variables or expressions. The name of the data file to be opened for input or output may be typed as a string variable or expression in the OPEN statement. In this way the file name can be assigned at the time the program is executed. For example, if the beginning statements of a program are

```
10 STRING A
20 PRINT "TYPE THE INPUT FILE NAME"
30 INPUT A
40 OPEN A,INPUT,1
```

the following will occur:

```
TYPE THE INPUT FILE NAME

? /XDATA/
```

and the file /XDATA/ will be opened for input as file 1 according to line 40.

String concatenation could be used to eliminate the need to type slashes around the file name; that is, line 40 could be

changed to

```
40 OPEN "/"+A+"/",INPUT,1
```

Then the file name could be typed simply as XDATA, and the slashes would be concatenated to this name in the OPEN statement itself.

String Functions

To aid the user in manipulating strings, SUPER BASIC has included a number of standard functions that operate on strings. These functions are explained below.

LENGTH (string)

This function returns a number equal to the number of characters in the specified string. For example,

```
>A="JONES"
>PRINT LENGTH(A)
 5
>
```

VAL (string)

This function takes a string of numeric information and returns a numeric value. For example,

```
>J="1234"
>K=VAL(J)
```

would set K to the numeric value 1234. The string used as an argument of this function can contain numeric information only. X=VAL("6E2") sets X to the value of 600, but X=VAL("A123") would cause an error message to be printed. In addition, spaces within the argument string are ignored; thus, Y=VAL("1.0 4") would set Y to the value of 1.04.

STR (numeric expression)

This function takes a numeric value and returns a string of numeric characters. For example, T=STR(99.6) sets T equal to a string variable with a string value of " 99.6". This string contains a leading space because of the omission of the + sign.

LEFT (string, numeric expression)

This function takes the number of characters specified by the second argument starting from the left side of the given string to form another string. For example,

```
>T="ABCDE"
>N=LEFT(T,2)
```

would give N the value of AB.

RIGHT (string, numeric expression)

This function takes the number of characters specified by the second argument starting from the right side of the given string to form another string. For example,

```
>PRINT RIGHT("ABCDE",3)
CDE
>
```

SUBSTR (string, numeric expression, numeric expression) OR
SUBSTR (string, numeric expression)

This function extracts a substring from the string given as the first argument. The function can have either two or three arguments. The number given as the second argument specifies which character of the string is the first character to be extracted. The number given as the third argument specifies how many characters of the string are to be extracted. If the third argument is omitted, the substring starts with the character specified by the second argument and continues to the end of the string. For example,

```
10 X="ABCDE"
20 Y=SUBSTR(X,2,3)
30 Z=SUBSTR(X,3)
```

will assign BCD to Y and CDE to Z.

INDEX (string,string)

If the second argument is a substring of the first argument, this function returns the character position of the second argument within the first; otherwise, it returns 0. For example,

```
10 X="ABCDE"
20 Y=INDEX(X,"BCD")
30 Z=INDEX(X,"E")
40 W=INDEX(X,"F")
```

will set Y to 2, Z to 5 and W to 0.

SPACE (numeric expression)

This function returns a string consisting of as many spaces as specified by the argument. For example,

```
>10 X="XX",Y="YYY"
>20 A=X+SPACE(3)+Y
>30 PRINT A
>40 M=2,N=4
>50 B=SPACE(M*N)+X
>60 PRINT B
>RUN
XX   YYY
        XX

>
```

Comparing Strings

Any of the relational operators (<,<=,=,>=,>,<>,#) can be used to compare strings. String characters are compared according to the following collating sequence which represents each character by a numeric code.

Code	Character	Code	Character
0	SPACE	40	@
1	!	41	A
2	"	42	B
3	#	43	C
4	$	44	D
5	%	45	E
6	&	46	F
7	'	47	G
10	(	50	H
11	)	51	I
12	*	52	J
13	+	53	K
14	,	54	L
15	-	55	M
16	.	56	N
17	/	57	O
20	0	60	P
21	1	61	Q
22	2	62	R
23	3	63	S
24	4	64	T
25	5	65	U
26	6	66	V
27	7	67	W
30	8	70	X
31	9	71	Y
32	:	72	Z
33	;	73	[
34	<	74	\
35	=	75	]
36	>	76	↑
37	?	77	←

Example

```
>A="JUNE",B="JULY"

>IF A>B THEN PRINT A:" > ":B
JUNE > JULY

>
```

The first two characters of the string values of A and B match, but since the letter N has a greater numeric code than the letter L, the string "JUNE" is greater than "JULY".

If the strings are of different lengths, the shorter string and the same number of characters from the longer string will be compared. If they match, the shorter string is taken to be the lesser of the two.

Example

```
>10 A="SUN"
>20 PRINT "VERIFIED" IF A<"SUNDAY"
>RUN
VERIFIED
>
```

Some other examples of statements using string comparison are:

```
15 IF A<>"PAID" THEN NEXT I
70 IF Z>="SMITH" THEN PRINT TAB(15):Z
130 PRINT "XXX" IF A+B<"MR. JONES"
GO TO 95 UNLESS RIGHT(X,2)="NG"
```

NOTE: Strings cannot be compared to numbers.

K. Complex Arithmetic

Complex Variables

Complex arithmetic can be performed easily in SUPER BASIC by using complex variables. A variable that is to be assigned a complex value must first be declared complex. To do this, type the variable name (or names, separated by commas) in a COMPLEX statement which can be executed directly or indirectly.

In the following example A and B are declared complex, assigned values by means of the INPUT command, and printed on the terminal.

```
>10 COMPLEX A,B
>20 INPUT A,B
>30 PRINT "A=";A,"B=";B
>RUN
? 5.6,-1.78,-300,15
A= 5.6,-1.78   B=-300, 15

>
```

Two numbers are required as input for each complex variable; namely, the real part and the imaginary part of the variable. When the value of a complex variable is printed, the real and imaginary parts are separated by a comma. The above example set A to 5.6-1.78i and B to -300+15i.

The COMPLEX statement also can be used to declare that an array will store complex values. For example,

```
10 COMPLEX R(0:20),S(M,N)
```

reserves space for a 21 element complex array R and an M by N complex array S. Each element of a complex array consists of two numbers, the real and the imaginary parts of the complex number.

The form of a complex number in a DATA statement is (A,B) where A and B are the real and imaginary parts of the complex number respectively. If B is zero, A may be typed alone in the DATA statement without parentheses. For example,

```
>10 COMPLEX X(3)
>20 READ X(I) FOR I=1 TO 3
>30 PRINT "X(1)=";X(1);"X(2)=";X(2);"X(3)=";X(3)
>40 DATA (5,4),3,(-4,1.7)
>RUN
X(1)= 5, 4  X(2)= 3, 0  X(3)=-4, 1.7

>
```

Complex Functions

CMPLX(A,B)

CMPLX(A,B) creates a complex value whose real part is equal to A and whose imaginary part is equal to B, where A and B can be any numeric expression. This function must be used to include a complex number in an assignment statement. For example,

```
>10 COMPLEX R,S
>20 R=CMPLX(1,5)
>30 N=4
>40 S=R+CMPLX(N+1,2)
>50 PRINT "R=":R,"S=":S
>RUN
R= 1, 5         S= 6, 7

>
```

If R and S had not been declared in the above example, only the real parts of their values would have been stored; the result would have been R=1 and S=6.

REAL (X)

This function returns the real part of the complex variable or expression.

```
>10 COMPLEX X,Y
>20 X=CMPLX(6,-1.1)
>30 Y=CMPLX(2.3,5)
>40 PRINT REAL(X),REAL(X+Y)
>RUN
 6               8.3

>
```

IMAG(X)

This function returns the imaginary part of a complex variable or expression.

```
>10 COMPLEX X,Y
>20 X=CMPLX(6,-1.1)
>30 M=IMAG(X)
>40 PRINT "M=":M
>RUN
M=-1.1
>
```

L. Picture Formatting

The user can specify his own format for output in addition to using the conventional SUPER BASIC forms of output. This feature, known as picture formatting, is useful in presenting calculated results in the form of tables and reports.

PRINT IN IMAGE Statements

The user may specify the exact format of his output by typing special characters in a string and using a PRINT IN IMAGE statement, as illustrated in the following example:

```
>10 INPUT A,B
>20 S="E FORMAT #######, INTEGER %"
>30 PRINT IN IMAGE S:A,B
>RUN
? 200,5.67
E FORMAT  .2E+03, INTEGER 6

>
```

In this example, S is a string variable which specifies the picture format to be used. The # signs in the string caused A to be printed in E format; the % signs caused the value of B to be rounded and printed as an integer. All other characters in the string (including spaces) were printed as specified. The format symbols # and %, which are explained below, cannot be printed as part of the picture format because of their special significance.

A picture format also may be used to write on a data file. For example,

```
PRINT ON 3 IN IMAGE S:X*Y,Z,W or
WRITE ON 3 IN IMAGE S:X*Y,Z,W
```

will print the values of X*Y,Z and W on file 3 in the format specified by the string variable S.

The picture format string can include any of the specifications listed below. The numeric fields will allow up to eleven significant digits of a number to be printed, depending on the number of symbols used in the format string. If the specified format cannot be used for the number to be printed (for example, if an insufficient number of places is specified), the message CANNOT FIT THIS FORM will be printed.

Integer Field

One or more % signs denote an integer field. One % sign must be typed for each digit of the number to be printed. Negative numbers require an additional % sign because of the preceding minus sign. A non-integer value will be rounded if an integer field is specified for it. For example,

```
>A=24,B=174.78
>PRINT IN IMAGE "%% %%% %%%":A,-A,B
24 -24 175
>
```

Note the alternate form of the PRINT IN IMAGE statement illustrated above. Instead of a string variable whose value specifies the format, the picture format string itself is typed after IN IMAGE.

Integer fields are right justified; that is, if more % signs are specified than are necessary, leading spaces will be printed before the number. For example, the format "%%%" would cause 24 to be printed with one space before it, and 4 to be printed with two spaces before it.

Decimal Field

One or more % signs with an embedded decimal point denote a decimal field. The number to be printed will be rounded to the specified number of decimal places. If the number is an integer or has fewer decimal places than the format specifies, trailing zeroes will be printed. Negative numbers require an additional % sign because of the preceding minus sign. For example,

```
>10 X=175.65,Y=11
>20 D="%%%.%% %%%%.%% %%.%"
>30 PRINT IN IMAGE D:X,-X,Y
>RUN
175.65 -175.65 11.0
```

A number that begins with a decimal point always will be preceded by a leading zero if a decimal field is specified. To allow for this leading zero, a % sign is needed before the decimal point in the format specification. For example,

```
>10 COMPLEX B
>20 B=CMPLX(.216,-.43)
>30 PRINT IN IMAGE "%.%%%  %%.%%":B
>RUN
0.216  -0.43
```

As shown above, a complex number requires two fields for output. Notice the two %% signs preceding the decimal point in the specified format for B's imaginary part. One % is for the minus sign, the other is for the leading zero.

Decimal fields are right justified; that is, if more % signs before the decimal point are specified than are necessary, leading spaces will be printed before the number.

NOTE: Whatever type of field is specified in SUPER BASIC picture formatting, no more than eleven significant digits of a number can be printed. If a number containing more than eleven significant digits is printed with a field of more than eleven symbols, the following will occur:

- Integer places past the eleventh significant digit will be filled with zeroes. For example, fourteen %'s will print the number 12345678901234 as 12345678901000.

- Decimal places past the eleventh significant digit will be replaced by blanks; for example, the field "%%%%%%%%.%%%%%" (in which eight %'s precede the decimal point and five follow it) will print the number 12345678.90123 as 12345678.901 followed by three blanks.

E Format Field

There are two forms for a field of E format:

1. A series of seven or more # signs.
2. One or more # signs, followed by a decimal point and a series of five or more # signs.

If the first form is used, the number printed begins with a decimal point. The second form allows the user to specify the number of digits before the decimal point. This is shown as follows:

```
>10 C=500
>20 PRINT IN IMAGE "#######":C
>30 PRINT IN IMAGE "##.#####":C
>RUN
 .5E+03
 50.E+01
>
```

In the first form of the E format field, a minimum of seven # signs is needed.

a) The first # is for the leading space or minus sign of the mantissa (the number to the left of E).

b) The second # is for the decimal point of the mantissa.

c) The third # is for the minimum of one digit for the mantissa.

d) The fourth # is for the character E.

e) The fifth # is for the plus or minus sign of the exponent.

f) The sixth and seventh #'s are for the two digit integer exponent.

In the second form of the E format field, the # signs are used as follows:

a) A minimum of one # before the decimal point is for the mantissa.

b) Four #'s after the decimal point are for the exponential part.

c) The last # is for the leading space or minus sign of the mantissa.

Notice that in the case of a positive number in E format, the leading space must be accounted for and always will be printed, while the integer and decimal fields allow this space to be suppressed.

Field Of Strings

One or more % signs or # signs may be used to denote a string field. The number of symbols specified in the format determines how many characters of the string will be printed. For example, if A="STRING", the format "%%%%%%" or "######" may be used to print A. In the following example

```
>10 T="CODE XY"
>20 PRINT IN IMAGE "%%%%%%%":T
>30 PRINT IN IMAGE "%%%%":T
>RUN
CODE XY
CODE
```

the entire string is printed first; then only four characters of the string are printed.

A string field is left justified; that is, if more % or # signs are specified than the number of characters in the string, trailing spaces will be printed.

Descriptive Text In A Format

Any literal text may be included in the picture format string. Every character is printed exactly as it appears in the format, except for %,#, more than three $ or * symbols,[20] and decimal points. For example, the results of a program calculating the perimeter P and the area A of a triangle may be printed as follows:

```
110 S="PERIMETER IS %%.%, AREA IS %%%.%"
120 PRINT IN IMAGE S:P,A
```

Floating $ Field

This field is used to specify that a $ is to be printed immediately preceding an integer or decimal value (or a string). For example,

---

[20]The meaning of these symbols is explained below.

```
>R="$$$.$$  $$$.$$  $$$$"
>PRINT IN IMAGE R:2.045,.7,300
 $2.05   $0.70  $300
>
```

These formats printed the specified values as the % formats would have, except that the last of the preceding spaces is replaced by a $. The $ always floats to the position before the first digit. If the $ field is specified so that there are no preceding spaces (that is, no room for the $), SUPER BASIC prints an error message. For example, 23.06 cannot be printed with the format "$$.$$".

The $ field must consist of four or more $ signs. For example, "$$$" is not a legal field, nor is "$$.$", since each of these contains only three $ signs. If these illegal fields were included in a format string, the characters would be taken as literal text; that is, printed as specified. For example,

```
>PRINT IN IMAGE "$%.%%":2.33
$2.33
>
```

The * Field

The * field is used to specify that * symbols are to appear before the number (or string) in place of the usual preceding spaces. For example,

```
>S="****  **.**  ***.**"
>PRINT IN IMAGE S:23,8.625,3.2
**23  *8.63  **3.20
>
```

These formats printed the specified values as the % formats would have, except that each preceding space is replaced by a *. If the * field is specified so that there are no preceding spaces (no room for a *), SUPER BASIC prints an error message. For example, 19.72 cannot be printed with the format "**.**".

The * field has the same restriction as the $ field. A minimum of four symbols is necessary. In the following example, "***" is interpreted as literal text rather than a field specification and is printed as specified:

```
>PRINT IN IMAGE "***##":"NOTE"
***NO
>
```

The * field is useful for check protection; that is, preceding *'s instead of spaces will prevent anyone from adding to the beginning of the dollar amount on a check.

Image Repetition

Since the "picture" specified in an IMAGE format is the image of a line, a Carriage Return is supplied when the format is exhausted. Thus, if more values are to be printed than the number of fields specified, more than one line of the same image will result.

Example 1

```
>PRINT IN IMAGE "%%":16.3,19
16
19
>
```

Example 2

```
>10 W="%   %   %.%%"
>20 PRINT IN IMAGE W:I FOR I=1 TO 8
>RUN
1   2   3.00
4   5   6.00
7   8
>
```

NOTE: In addition to the types of examples illustrated thus far, a picture format can be specified by a string formed by concatenation, that is,

```
>G="%%%"
>F="%%.%"
>PRINT IN IMAGE F+G:16.3295
16.3295
>
```

PRINT IN FORM Statements

In addition to the line image type of picture format described above, SUPER BASIC provides a second type of format that uses IN FORM instead of IN IMAGE. The form of the output statements is similar, that is,

```
PRINT IN FORM S:A,B
PRINT ON 3 IN FORM S:X*Y,Z,W or
WRITE ON 3 IN FORM S:X*Y,Z,W
```

However, the format is field-oriented rather than line-oriented. The picture format string will not be an image of the printed line, but will specify fields for whatever will be printed, whether numbers, strings, descriptive text, or blanks.

Numeric, String, And Blank Fields

The symbols used to specify numeric and string fields are identical for IN FORM and IN IMAGE statements. One of the major differences between the two types of format statements is that when IN FORM is used, blanks typed between fields in the format string will not be printed as specified. For example, if M=12 and N=56.88, the statement

```
PRINT IN FORM "%% %%.%%":M,N
```

will print the values of M and N with no spaces between them. The blank in the above format serves only to separate the field for M from the field for N. To print blanks between numbers, use one or more B's to denote a field of blanks. Thus,

```
PRINT IN FORM "%% BBB %%.%%":M,N
```

will print the values of M and N with three spaces between them.

Character And Field Replication

When IN FORM is used, the picture format can be written in a "shorthand" notation; that is, replication of characters and fields is permitted by using a multiplier. The following chart gives several examples of IN FORM character replication:

The Format:	May Be Typed As:
"%%%"	"3%"
"%%%%.%%%"	"4%.3%"
"#######"	"7#"
"##.#####"	"2#.5#"
"%% BBBB %%.%"	"2% 4B 2%.%"
"**********.**"	"10*.2*"

The user also may specify the number of times a format field is to be used. The form of this field replication is

```
N (format field)
```

where N is the number of times the format field is to be used.

Example 1

The format

```
"2(3%.2% B)"
```

is equivalent to

```
"%%%.%% B %%%.%% B"
```

Example 2

```
>10 A=543.66,B=78.743,C=345.788
>20 G="2(3%.3% 4B) %%%"
>30 PRINT IN FORM G:A,B,C
>RUN
543.660      78.743     346
>
```

In this example, the field 3%.3% 4B is used twice (to print A and B); then the field %%% is used to print C.

Example 3

The format

"20(4%.2% B 4(3% B)/)"

illustrating two levels of field replication, may be used to print twenty lines, each with a decimal number and four integer numbers. A / generates a Carriage Return (see below). NOTE: Up to four levels of field replication are allowed in a format.

Field For Descriptive Text

When IN FORM is used, any literal text that is to be printed must be enclosed in single quote marks to denote a text field.[21] For example,

```
>10 D="'X EQUALS' B %.6%"
>20 X=PI/180
>30 PRINT IN FORM D:X
>RUN
X EQUALS 0.017453
>
```

Carriage Return In A Format

Unlike a format used in an IN IMAGE statement, no Carriage Return is given when the IN FORM format is exhausted. Thus if fewer fields are specified than the number of values to be printed, the format simply will be repeated on the same line as shown below.

```
>10 T="% 2B %.% 2B"
>20 PRINT IN FORM T:I FOR I=1 TO 5
>RUN
1  2.0  3  4.0  5
>
```

[21] If the format string is enclosed in single rather than double quote marks, the literal text to be printed is enclosed in double quotes.

A slash (/) can be used in a format to generate a Carriage Return. Consecutive slashes may be used to generate blank lines. Note the results when the format above is modified to end with a / instead of 2B:

```
>10 T="% 2B %.%/"
>20 PRINT IN FORM T:I FOR I=1 TO 5
>RUN
1  2.0
3  4.0
5
>
```

When printing a matrix IN FORM, use the / to generate a Carriage Return at the end of each row. For example,

```
>MAT INPUT A(3,3)
? 1,3,-6,8,11,9,4,2,1

>MAT PRINT IN FORM "3(%% 2B)/":A
 1   3  -6
 8  11   9
 4   2   1

>
```

IN FORM and IN IMAGE differ in another way concerning the Carriage Return in a format. Because the IMAGE format is the image of a line, the information printed always will be preceded and followed by a Carriage Return; that is, always will be a separate line. The IN FORM statement however, can cause its printout to be appended to printout from a previous line, as follows:

```
>10 PRINT "LINE 10 ";
>20 PRINT IN FORM "7%":"LINE 20"
>RUN
LINE 10 LINE 20
>
```

This cannot be done with PRINT IN IMAGE.

The Single

A single # may be used with PRINT IN FORM to specify what is known as "free field" format. Any number or string may be printed with this field. Up to eleven significant digits of a number will be printed. If the free field format is used to print a string, the entire string will be printed. For example,

```
>10 A="STRING"
>20 B=68.9
>30 C=666
>40 PRINT IN FORM "#":A,B,C,PI
>RUN
STRING  68.9  666.  3.1415926535
>PRINT IN FORM "#":1234567890123 45
  .12345678901E+15
>
```

M. Advanced Editing Features

The editing commands and characters described in section 1(F) are only a small part of the extensive editing features available in SUPER BASIC. Instead of retyping an entire line that needs changing, the user may let certain control characters do the editing for him. These control characters, which are the same as those available in the Tymshare EDITOR language, are summarized in the table on the adjacent page.

The first set of characters listed can be used at any time - while typing direct and indirect statements, file names, and even data input from the keyboard. The second set of characters is used to edit lines already typed, even if a syntax error was made in the line. The EDIT and MODIFY commands allow editing of any existing line in a program. Further explanation and examples of these editing features are given below.

Editing The Line Being Typed

In the following example, Control Q (Q^C) is used to delete the line being typed. While retyping the line, two incorrect characters are deleted with A^C's.

```
>40 FOR I=1 TOQ^C↑
40 PRINT I↑3 FOR I=1 TO 25A^C←A^C←50 Cr
>LIST 40 Cr
40 PRINT I↑3 FOR I=1 TO 50
>
```

The TABS Command

The tab stops which determine how many spaces I^C will type are initialized at 7, 15, and at every fifth position from 15 on. The direct command TABS allows the user to set any other tabs that he wishes. For example,

```
>TABS 10,20,30 Cr
```

sets the tab stops at the specified positions. A control I subsequently typed at the beginning of a line will space to position 10. NOTE: A maximum of ten tabs may be set with the TABS command.

EDITING CONTROL CHARACTERS		
Control Character	Symbol Printed	Function
Used at any time:		
		FOR DELETING
A^C or←	←	Deletes the preceding character typed.
W^C	\	Deletes the preceding word typed.
Q^C	↑	Deletes the entire line being typed.
		OTHER
I^C		Types spaces up to the next tab stop.
V^C and a character		Indicates that the control character that follows is to be accepted as any other character (it will not perform its editing function).
Used only during EDIT, MODIFY and edit of previous line:		
		FOR DELETING
S^C	%	Deletes the next character in the line being edited (the "old line").
K^C		Deletes the next character in the old line; prints the character it deletes.
P^C and a character	%	Deletes up to but not including the character typed after it.
X^C and a character	%	Deletes up to and including the character typed after it.
Carriage Return		Deletes the rest of the old line and ends the edit.
		FOR COPYING
C^C		Copies the next character in the old line.
O^C and a character		Copies up to but not including the character typed after it.
Z^C and a character		Copies up to and including the character typed after it.

EDITING CONTROL CHARACTERS (Continued)		
Control Character	Symbol Printed	Function
D^C		Copies the rest of the old line (printing it out) and ends the edit.
F^C		Copies the rest of the old line without printing it out and ends the edit.
H^C		Copies the rest of the old line (printing it out) and continues the edit at the end of the line.
Y^C		Copies the rest of the old line without printing it out and continues the edit at the beginning of the new line (same as F^C followed by MODIFY of the line as edited).
R^C		Copies and prints the rest of the old line plus the new line; continues the edit from where R^C was typed.
T^C		Same as R^C except that it aligns the rest of the old line and the new line.
U^C		Copies from the old line up to the next tab stop in the new line.
E^C text E^C	< >	FOR INSERTING Inserts text into the old line; first E^C prints <, second E^C prints >.
N^C		OTHER Backspaces in the old and in the new line.

File Name Editing

File names typed during the LOAD or SAVE command can be edited also. For example,

>SAVE /XYA^{C}←Z/ Cr

will save the program on a file named /XZ/.

To include a control character in a file name, precede the character by V^{C} so that no editing will occur. For example,

>LOAD /PV^{C}W^{C}R/ Cr

must be typed to load from a file named /PW^{C}R/.

Data Input Editing

The control characters A^{C}, W^{C} and Q^{C} have special properties when used to edit data typed in response to the INPUT command.

Control A will delete the preceding character unless that character is:

1) A comma (or space) used to separate data items. Once such a character is typed, the preceding value is stored in a variable and is not available for edit.

2) Either of the quote marks used to enclose a string data item. Once the first quote mark is typed, the user cannot delete it and type in a number instead of a string. As soon as the second quote mark is typed, the string is stored in a variable and is not available for edit.

For example,

```
>10 INPUT A,B,C Cr
>20 PRINT A;B;C Cr
>RUN Cr
? 123,56A^C←5,"ERA^C←A^C←STRING" Cr
 123   55   STRING
>
```

Once the comma was typed after 123, no editing could be done to that value. The first A^{C} deleted 6. The second and third A^{C}'s deleted ER; any more A^{C}'s typed there would not have been able to delete the leading quote mark.

Control W, which deletes the preceding data item, also has no effect on the characters which A^{C} cannot delete. For example,

```
>INPUT X,Y,Z Cr
? "SMYTW^C\SMITH",64,92W^C\93.8 Cr
>PRINT X:Y:Z Cr
SMITH 64 93.8
>
```

The first W^C deleted SMYT but not the leading quote mark. The second W^C deleted 92; another control W^C typed there would have done nothing, since 64 was already stored in the variable Y.

Control Q restarts the entire statement containing the INPUT command, causing SUPER BASIC to print another ?. Since direct statements are not saved and therefore cannot be re-started, Q^C applies only when the INPUT command was executed indirectly. For example,

```
>10 INPUT A(I) FOR I=1 TO 8 Cr
>RUN Cr
? 11.17,33.9,46.1,39,21.8,5.62 Cr
13.7Q^C↑
? 11.7,85,33.9,46.1,39,21.86 Cr
13.7,10.8 Cr
>
```

Note that the values for A(1) to A(6) were actually stored before the Q^C was typed, then the user typed in new input values. Thus, if the INPUT command were in a statement such as

```
>55 IF A=0 THEN INPUT A,B
    ELSE PRINT "NO"
```

the following might occur:

```
? 5,7.5Q^C↑
>
```

Statement 55 was restarted, but since A was actually assigned the value of 5 before Q^C was typed, A was no longer equal to zero and INPUT A,B was not executed.

Editing A Line Already Typed

EDIT And MODIFY

The direct commands EDIT and MODIFY allow the user to edit any statement in his program by using an extensive set of control characters. EDIT followed by a line number causes SUPER BASIC to print the specified line and wait for the user to edit. MODIFY (or MOD) is the same as EDIT except that the specified line is not printed.

Example 1

```
>EDIT 20 Cr
20 A=SQR(PI*M↑2)                    This is line 20.
Z^C*20 A=SQR(PI*ND^C↑2)             Z^C* copies up to and including the
                                    *.  The user typed N to replace
                                    the incorrect M, and D^C to copy
                                    the rest of the line.

>LIST 20 CR
20 A=SQR(PI*N↑2)                    This is the new line 20.
```

Example 2

```
>10 INPUT A(I) FOR I=1 TO 10 Cr
>20 GOSUB 100 Cr
>MODIFY 10 Cr                       Line 10 does not print.
3O^CA10 INPUT BF^C                  3 replaces 1 so that the edited
                                    line will be line 30.  O^CA copies
                                    up to but not including A.  The
                                    user types B to replace the A,
                                    and F^C which copies but does not
                                    print the rest of the line.

>LIST Cr
10 INPUT A(I) FOR I=1 TO 10
20 GOSUB 100
30 INPUT B(I) FOR I=1 TO 10
>
```

Editing The Previous Line

After the user types any indirect statement, that statement is immediately available for edit as though the EDIT or MODIFY command had been given. For example,

```
>45 IF Y=20 THEN NEXT I Cr
>Z^C245 IF Y=25D^C THEN NEXT I     Z^C and D^C are used to edit the
>LIST 45 Cr                        line just typed.  The 20 is
45 IF Y=25 THEN NEXT I             changed to 25.
>
```

This can be done even if a syntax error is made in the statement just typed.

Direct statements can be edited after they are typed only if a syntax error is made. Once the statement executes, it is no longer available for edit. For example,

```
>PRINT "AREA IS:A Cr
MISSING "                          This is a syntax error.
>ZCSPRINT "AREA ISEC<"EC>DC:A      The statement is edited.
AREA IS 35                         DC copies the rest of the line and
                                   causes the statement to be executed.

>PRINT "VOLUME IS":X Cr            This statement contained no syntax
VOLUME IS                          errors, so SUPER BASIC began to
VARIABLE HAS NO VALUE              execute it.  The variable X was
                                   not defined (a program error).
                                   Control characters will have no
>                                  effect here.
```

The RENUMBER Command

Renumbering To The End Of The Program

All or some of the statements in a program may be renumbered with a direct command which takes the form:

```
RENUMBER N1,N2,N3 or
REN N1,N2,N3
```

where N1 will be the first new line number, N2 is the number of the line in the program where renumbering will begin, and N3 is the increment to be used in assigning the new line numbers.

Example

```
>1 !THIS IS A TEST PROGRAM
>10 INPUT P,I,N
>11 M=P*(I+1)^N
>15 PRINT M
>20 GO TO 10
>RENUMBER 20,10,2
>LIST
1 !THIS IS A TEST PROGRAM
20 INPUT P,I,N
22 M=P*(I+1)^N
24 PRINT M
26 GO TO 20
>
```

In this example, the program is renumbered from line 10 to the end of the program, in steps of 2, with 20 as the first new line number. Line 1 remains unchanged. Notice that the line number referred to in the GO TO statement also has been changed correctly.

Certain words may be included in the RENUMBER command to help the user remember the order and meaning of the three arguments. For example,

```
RENUMBER 20,10,2
```

can be typed as

```
RENUMBER AS 20 FROM 10 BY 2 or
RENUMBER AS 20 FROM 10 INC 2
```

Any of these prompting words may be used or not as desired. AS is optional, and either FROM, BY, or INC may be replaced by a comma.

Renumbering A Range Of Lines

A range of lines may be specified for renumbering. For example,

```
RENUMBER 200,90-205,10
```

will renumber lines 90 to 205 as 200, 210, 220 and so on.

An additional prompting word may be included in this form of the RENUMBER command; namely, the dash used in indicating the line range may be replaced by the word TO.

When the RENUMBER command is given, SUPER BASIC first checks to see that after the requested renumbering is done, the renumbered line range will still have line numbers that are different from the rest of the program. If this is not the case, an error message will be printed, since it is impossible for two program lines to begin with the same number.

Omitting Parts Of The RENUMBER Command

One or more parts of the RENUMBER command may be omitted, with the following results:

Omitted	Result
N1	First new line number is assumed to be 100.
N2	Program is renumbered from the beginning (the lowest numbered statement).
N3	Increment is assumed to be 10.

Examples

RENUMBER	All three parts are omitted. RENUMBER 100,0,10 is assumed. (The 0 will cause renumbering to begin from the lowest numbered statement.)
RENUMBER,,5 or RENUMBER BY 5	First two parts are omitted. RENUMBER 100,0,5 is assumed.
RENUMBER 10 or RENUMBER AS 10	Last two parts are omitted. RENUMBER 10,0,10 is assumed.
RENUMBER 10,,5 or RENUMBER AS 10 BY 5	Second part is omitted. RENUMBER 10,0,5 is assumed
RENUMBER 150, 115-210 or RENUMBER AS 150 FROM 115 TO 210	Third part is omitted. RENUMBER 105,115-210,10 is assumed.

RENUMBER With ADD

There is another form of the RENUMBER command in which the numbers of the specified lines are increased by a certain amount. For example,

```
RENUMBER 150 ADD 10 or
RENUMBER FROM 150 ADD 10
```

will renumber from line 150 to the end of the program by adding 10 to every line number.

A range of lines may be specified, such as

```
RENUMBER 210-340 ADD 20 or
RENUMBER FROM 210 TO 340 ADD 20
```

which will add 20 to the line numbers 210-340 inclusive.

N. Control Of Running Programs

Control Commands

SUPER BASIC gives the user complete control of his running program. An indirect PAUSE or STOP statement causes program execution to be interrupted, as does pressing the ALT MODE/ESC key. The user then can enter direct statements which will, for example, assign or change variable values, print out values, or list parts of the program. He then may resume execution at the point of interruption or anywhere else in his program.

PROGRAM CONTROLS

COMMAND	EFFECT	TO CONTINUE
100 PAUSE	Interrupts the program at statement 100. The message PAUSE IN STEP 100 is printed. Direct statements can be entered.	a) Type GO to continue execution at the point of interruption. All information in the program before interruption is retained. NOTE: After interruption, if the user types an indirect statement or deletes a statement, GO will not continue execution. Any information about FOR loops or GOSUB commands is lost. b) Type GO TO line number to continue execution anywhere in the program. All information in the program is retained. c) Type RUN to reinitialize execution from the beginning of the program. No information is retained; that is, all values are reinitialized and all files are closed.[22]
ALT MODE/ESC	Finishes execution of the statement that was being executed when ALT MODE was pressed and prints the message INTERRUPTED BEFORE STEP , the step being the next statement. (Note exception below.)	Same as PAUSE

[22]Except the commands file.

ALT MODE/ESC (Twice)	To interrupt execution of an INPUT statement or a statement with an infinite loop (such as PRINT A WHILE A>1), press ALT MODE twice. The message INTERRUPTED IN STEP is printed.	Same as PAUSE except that GO will not resume execution reliably.
Normal end of program	Terminates program execution.	a) Type GO TO line number to continue execution anywhere in the program except inside a FOR loop or a subroutine. All information in the program is retained. b) Type RUN to reinitialize execution from the beginning of the program. No information is retained; that is, all values are reinitialized and all files are closed.[23]
35 STOP or 35 END	Termination program execution at statement 35.	
Program execution error	Terminates program execution and prints the message ERROR IN STEP : followed by an error diagnostic.	
>QUIT or >Q or 100 QUIT	Returns to the EXECUTIVE. Closes all files.[23]	a) Type -CONTINUE (or -CON) to return to SUPER BASIC and continue. b) Type -SBASIC to reinitialize SUPER BASIC.

NOTE: Although RUN normally retains no information, a VAR=ZERO, VAR=UNDEF, or BASE command will be retained when the RUN command is given.

[23]Except the commands file.

Commands Files: New version being implemented.

SECTION 3

SUMMARY OF SUPER BASIC

All commands can be executed both directly and indirectly unless specified otherwise.

1. VARIABLES AND ARRAYS

Variable Names

single letter
single letter followed by single digit
single letter followed by $

Subscripted Variable (Array) Names

single letter
single letter followed by $

Variable Initialization

Variables ordinarily are not initialized.
VAR=ZERO initializes variables to 0.
VAR=UNDEF nullifies VAR=ZERO.

Value Types

Type	May Be Declared As
Real Number Integer; e.g.,15,7 Decimal; e.g.,13.6, -.03 E Notation; e.g.,6E2 (E2 meaning times 10^2)	REAL INTEGER
Logical Value All variables with a numeric value have a logical value as well. TRUE if the numeric value≠0. FALSE if the numeric value=0.	LOGICAL If declared, value returned is: 1 for TRUE 0 for FALSE
Complex Number Declaration is required. NOTE: The logical value of a complex variable is set to the logical value of its real part.	COMPLEX
String Value Any combination of characters.	STRING TEXT (arrays only)

DIM And Declaration Statements

DIM reserves space for array elements which may be integer, real, or string. Defines no elements. Arrays with a subscript greater than 10 or with more than two dimensions (subscripts) require DIM.

```
DIM A(20),  B(6)          Subscript base 1 is implied.
DIM A(0:20),  B(-6:6)     Base other than 1 is specified.
```

BASE n specifies subscript base n; for example,

```
BASE 0
DIM C(2,4)
```

reserves space for a 3x5 matrix C.

Delcaration statements dimension arrays that are declared, using the same form as DIM; for example,

```
INTEGER A(20), B(-6:6)
```

2. OPERATORS

Type	Operators		Operate On
Arithmetic	↑	exponentiation	Numeric variables and expressions.
	-	unary minus	
	MOD	modulo	
	*,/	multiplication, division	
	+,-	addition, subtraction	
Relational	<	less than	String or numeric variables and expressions.
	<=	less than or equal to	
	=	equal to	
	>=	greater than or equal to	
	>	greater than	
	<> or #	not equal to	
Logical	NOT		Logical values of numeric variables and expressions.
	AND		
	OR		
	IMP	implication	
	EQV	equivalence	
String	+	concatenation	String variables and expressions.

3. FUNCTIONS

Standard Functions

Function	Brief Description
	Mathematical Functions
ABS(X)	Absolute value of X.
ATN(X) or ATAN(X)	Arctangent (in radians, over the range $-\pi/2$ to $+\pi/2$) of X.
ATN(Y,X) or ATAN(Y,X)	Arctangent (in radians, over the range $-\pi$to$+\pi$) of Y/X.
COS(X)	Cosine of X (X in radians).
EXP(X)	Natural exponential of X, e^X.
INT(X) or IP(X)	Greatest integer not exceeded by X.
FIX(X)	X truncated: equal to X-IP(X).
FP(X)	Fractional part of X: equal to SGN(X)*INT(ABS(X)).
LOG(X)	Natural logarithm of X.
LOGT(X) or LOG10(X)	Logarithm of X (base 10).
PI	Mathematical constant π.
RND(X)	Random number generator. NOTE: RND(0) may be typed as RND.
SGN(X)	Sign function (1 for positive X, 0 for X=0 and -1 for negative X).
SIN(X)	Sine of X.
SQR(X) or SQRT(X)	Square root of X.
TAN(X)	Tangent of X (X in radians).
	Print Functions
POS	Position of print head (terminal output).
POS(X)	Position of print head (output on file X). NOTE: For binary files, POS(X) is word position.
TAB(X)	Tab to print positon X (used with PRINT).
	String Functions
(S denotes string argument; N denotes numeric argument)	
INDEX(S_1,S_2)	Position of S_2 within S_1 (e.g., INDEX ("ABC","B")=2).
LEFT(S,N)	Substring of S; N characters, starting from left.
LENGTH(S)	Length of string S.
RIGHT(S,N)	Substring of S; N characters, starting from the right.
SPACE(N)	String of N spaces.
STR(N)	String of the characters comprising N, (e.g., STR(3)=" 3").

Function	Brief Description
SUBSTR(S,N_1)	Substring of S, from N_1th character to end of S.
SUBSTR(S,N_1,N_2)	Substring of S; N_2 characters, starting from N_1th character.
VAL(S)	Numeric value of S, where S must be a numeric string (e.g., VAL("-6")=-6).
	Complex Functions
CMPLX(X,Y)	Complex number with real part X and imaginary part Y.
IMAG(C)	Imaginary part of the complex argument C.
REAL(C)	Real part of the complex argument C.

Programmer Defined Functions

DEF (indirect only) defines a function with name FN followed by a single letter; for example,

```
80  DEF FNS(X,Y)=2*SIN(X)-FNA(2)
100 DEF FNK=2.165*R↑2
```

4. INPUT/OUTPUT STATEMENTS

Fundamental Input/Output Statements

Method of Input or Output	Example			
	Real	Complex (must be declared)	Undeclared String	Declared String
Assignment Statement assigns values to variables (LET is optional).	LET A=6 X,Y,Z=0 B=3,K=C*N↑2	A=CMPLX(6,X)	X="DOUBLE" Y='SINGLE'	Same as undeclared string.
INPUT prints ?, accepts input typed in reply.	INPUT A,B ? 4.5,6	INPUT A ? 11,-4.1	INPUT A ? "STRING" or 'STRING'	INPUT A ? STRING (but quotes must enclose strings with leading spaces or commas)
READ reads data from DATA statements. (DATA is indirect only) RESTORE causes rereading from beginning of DATA statements.	10 READ A,B 50 DATA 4.5,6	10 READ A 50 DATA (11,-4.1)	10 READ A 50 DATA "STRING" or 50 DATA 'STRING'	10 READ A 50 DATA STRING (but quotes must enclose strings with leading spaces or commas, and numeric strings).
PRINT prints numbers, text, values of variables and expressions. PRINT zones: , normal (15 spaces) ; packed : concatenated	A=6 PRINT A;A/2 6 3	A=CMPLX(2,3) PRINT "A=":A A= 2, 3	X="XX",Y='YY' PRINT X,Y XX YY	

Data File Input/Output Statements

Statement Model	Remarks
OPEN/file name/FOR [SYMBOLIC or BINARY] [INPUT or OUTPUT] AS FILE n Short form: OPEN /file name/, [SYMBOLIC or BINARY] [INPUT or OUTPUT] , n	Three files may be open concurrently.* File number n may be any positive numeric expression. File name may be a string variable. Output data file need not exist previously.
INPUT FROM n: variable list	n is file number of data input file.
[PRINT or WRITE] ON n: list of variables or expressions	n is file number of data output file; usual PRINT functions and zones apply.
CLOSE n	Closes data file n (automatic after RUN, DEL ALL and return to EXEC) Also: CLOSE $n_1,n_2,..$

* Also "TELETYPE" (or "TEL") to denote the terminal.

Examples

```
30 OPEN /DATA/,INPUT, 1
65 INPUT FROM 1:X,Y,Z
85 WRITE ON M*N:R,S,T↑2
210 CLOSE 3,B-2
```

Picture Formatted Output

PRINT IN IMAGE string: { list of values,
PRINT IN FORM string: { variables or expressions.

Format string contains field specification symbols:

TYPE OF FIELD	FIELD SPECIFICATION		EFFECT		REMARKS
	IN IMAGE or IN FORM	IN FORM	PRINTS	AS	
INTEGER	"%%%"	"3%"	432	432	Leading space can be suppressed.
DECIMAL	"%%%.%"	"3%.%"	-16.39	-16.4	
E NOTATION	"#######" "#.#####"	"7#" "#.5#"	400	.4E+03 4.E+02	Leading space cannot be suppressed.
STRING	"%%%%%%" "###"	"6%" "3#"	STRING	STRING STR	
FLOATING $ * FIELD	"$$$.$$" "***.**"	"3$.2$" "3*.2*"	9.4	$9.40 **9.40	IN IMAGE: less than 4 $'s or *'s will not be interpreted as a field specification.
DESCRIPTIVE TEXT and SPACES	IN IMAGE: Characters other than above symbols will be printed as specified (including spaces); e.g., "X IS %%" IN FORM: Spaces are used to separate field specifications. Text to be printed is enclosed in single quotes. Spaces are denoted by B's; e.g., "'X IS' B %%"		12	X IS 12	Note IMAGE exception above. (Less than 4 $'s or *'s will be printed as specified.) IN FORM descriptive text will be enclosed in double quotes if format string is in single quotes.
CARRIAGE RETURN	IN IMAGE: Supplied at end of image; e.g., "%" IN FORM: Denoted by /; e.g., "%/"		1,2,3	1 2 3	

IN FORM:

Field replication (FORM "3(2% B)" is equivalent to "%% B %% B %% B")

Free field format (a single #, prints any string or number, up to 11 significant digits).

Example of picture formatted output to a data file:

```
PRINT ON 3 IN IMAGE S:A,B,C
```

NAME	EXAMPLE	REMARKS
	INPUT	
MAT READ	MAT READ A,B,C, MAT READ K(15),L(-1:1,3)	Matrices are read in row order (i.e., second subscript varies more rapidly).
MAT INPUT	MAT INPUT A,B,C MAT INPUT R(2,3),S(0:M) MAT INPUT FROM 1:A(N+1)	
MAT PRINT	OUTPUT MAT PRINT A,B;C MAT PRINT ON 2:R;S; MAT WRITE ON 2:R;S;	May be picture formatted. Matrices are printed in row order.
	MATHEMATICAL OPERATIONS	
Addition	MAT C=A+B	
Subtraction	MAT C=A-B	
Multiplication	MAT C=A*B	Matrices only. MAT A=A*B is illegal.
Scalar Multiplication	MAT C=(X-5)*A MAT C=A	
Transpose	MAT C=TRN(A)	Matrices only. MAT A=TRN(A) is illegal.
Inverse	MAT C=INV(A)	Square matrices only. Uses Gauss-Jordan method.
	MATRIX INITIALIZATION	
ZER	MAT C=ZER MAT C=ZER(M) MAT C=ZER(15,N)	Sets all elements to zero.
CON	MAT C=CON MAT C=CON(M) MAT C=CON(15,N)	Sets all elements to one.
IDN	MAT C=IDN MAT C=IDN(M,M)	Square matrices only. Sets identity matrix.

6. CONTROL STATEMENTS

See chart below for FOR and NEXT.

Statement Model	Remarks
END	Not needed at end of program.
GO	Direct only.
GO TO line number	When used directly, retains all previous information.
GOSUB line number	Be sure to isolate subroutine from main program.
IF logical expression THEN statement IF logical expression THEN statement ELSE statement	The statement after the THEN or ELSE clause can be any indirect statement except DATA, REM or !
ON numeric expression GO TO $line_1$, $line_2$,... ON numeric expression GOSUB $line_1$, $line_2$,...	Value of numeric expression will be truncated if not an integer.
PAUSE	Indirect only.
QUIT (or Q Cr when used directly)	Also can be used indirectly.
RETURN	
RUN	Direct only.
STOP	Equivalent to END.

FOR and NEXT

These commands are indirect only:

Example of FOR	Remarks
30 FOR X=1 TO 10	Implied step of 1.
30 FOR X=1 TO 10 STEP 2	Step specified as 2.
30 FOR X=10 TO 1 STEP -2	Negative step specified.
30 FOR X=10 TO 1 BY -2 30 FOR X=10 STEP -2 TO 1 30 FOR X=10 BY -2 TO 1	Alternate forms of above example.
30 FOR X=1,2,7,8	Values listed.
30 FOR X=1,2,6 TO 18 STEP 3,50	Values and range listed.
30 FOR X=N*Q TO Q/3 STEP N	Variables used in defining range.
30 FOR X=1 WHILE X<=Y 30 FOR X=1 UNTIL X>Y 30 FOR A=10 STEP 2 WHILE A<Y	WHILE and UNTIL used to specify final value. Change in Y within loop will alter final value.
30 FOR X=1 TO Y	Change in Y within loop will not alter final value.

FOR statement is accompanied by NEXT.

80 NEXT X

80 NEXT I,J (equivalent to { 80 NEXT I / 81 NEXT J })

7. STATEMENT MODIFIERS

Most direct statements and all indirect statements except DATA can be modified.

Modifier	Example	Effect of Modifier
IF	INPUT N IF M=SQR(7)	INPUT N executed only if M equals SQR(7).
UNLESS	INPUT N UNLESS M=SQR(7)	INPUT N executed only if M does not equal SQR(7).
FOR	PRINT X↑2 FOR X=1 TO 10	Equivalent to: 10 FOR X=1 TO 10 20 PRINT X↑2 30 NEXT X FOR modifier takes the same forms as the FOR statement.
WHILE	X=2*X WHILE X<Y	X=2*X executed repeatedly as long as X is less than Y.
UNTIL	X=2*X UNTIL X<Y	X=2*X executed repeatedly as long as X is greater than or equal to Y.

WHILE and UNTIL also may be used with FOR to specify the final value.

8. LOADING AND SAVING THE PROGRAM

These commands are direct only.

Command	Example	Purpose
LOAD	>LOAD /A/ >LOAD (A3JIM)/@PAUL/ >LOAD "SIMEQN"	To load program statements saved on a file.
SAVE	>SAVE /FILE/ >SAVE /XY/,1-15,30,70-100	To save all or part of a program.
TAPE	>TAPE (If D^C was not punched at end of tape, type D^C after tape is read.)	To load program statements from paper tape.

9. EDITING AND UTILITY COMMANDS

All of these commands are direct only except REM and !

Command	Example	Remarks
DELETE or DEL	>DELETE 10 >DEL 8-10,70 >DELETE ALL	DELETE ALL has the same effect as returning to EXEC and recalling SBASIC.
EDIT	>EDIT 25	The line to be edited will be printed out.
LIST	>LIST 25 >LIST 10,65-90 >LIST	LIST alone lists the entire program.
MODIFY or MOD	>MOD 10	The line to be edited will not be printed out.
REM and !	>10 REM PRINT A >! SUBROUTINE >55 A=A+1 ! ADD 1	Only ! can append comments to statements (see the last example).
RENUMBER or REN	>RENUMBER 20,10,5 >REN AS 20 FROM 10 BY 5 >REN 20,10-95,5 >REN AS 20 FROM 10 TO 95 INC 5 >RENUMBER BY 5 >REN >REN 30 ADD 10 >REN FROM 30 TO 65 ADD 10	When omitted, first new line number is assumed to be 100, first old to be 0 (program is renumbered from the beginning), and increment to be 10.
TABS	>TABS 5,10,15,20	Tabs are initialized at 7,15 and steps of 5 from 15 on. I^C spaces to next tab stop.

SECTION 4

SAMPLE SUPER BASIC PROGRAMS

Listing Stocks

This program reads up to 100 items of string and numeric data from a file in which the last item is known to be the string "END". The data is printed on the terminal with a picture format, followed by the sum of the numeric information. Note that the left justification of strings and the right justificatcn of numbers is an extremely useful feature of picture formatting.

```
-COPY /BSTOCKS/ TO TEL

ABBOT LABS.,100,AMPEX,100,BECKMAN INSTRUMENT,100,BRISTOL MYERS,20
COLGATE PALMOLIVE,100,CONTINENTAL BAKING,100,FOREMOST MCKESSON,200
"GRANT, W.T.",100,HALLIBURTON,100,HOWMET,200,INT. TEL. & TEL.,12
LITTON IND.,17,MC CALL,100,NATIONAL CAN,100,NORWICH PHARMACAL,64
OLIN MATHIESON CHEM.,100,SQUIBB BEECH NUT,66,UNITED FRUIT,100
END

-SBASIC

>LOAD /STOCKS/
>LIST
10 S=0
20 STRING C(100)
30 DIM N(100)
40 OPEN /BSTOCKS/,INPUT,1
50 FOR I=1 TO 100
60 INPUT FROM 1: C(I)
70 IF C(I)<>"END" THEN INPUT FROM 1: N(I) ELSE 100
80 S=S+N(I)
90 NEXT I
100 PRINT
110 PRINT IN FORM "27% 3%/": C(J),N(J) FOR J=1 TO I-1
120 PRINT
130 PRINT "TOTAL NO. OF SHARES IS":S
140 CLOSE 1
>RUN

ABBOT LABS.                 100
AMPEX                       100
BECKMAN INSTRUMENT          100
BRISTOL MYERS                20
COLGATE PALMOLIVE           100
CONTINENTAL BAKING          100
FOREMOST MCKESSON           200
GRANT, W.T.                 100
HALLIBURTON                 100
HOWMET                      200
INT. TEL. & TEL.             12
LITTON IND.                  17
MC CALL                     100
NATIONAL CAN                100
NORWICH PHARMACAL            64
OLIN MATHIESON CHEM.        100
SQUIBB BEECH NUT             66
UNITED FRUIT                100

TOTAL NO. OF SHARES IS 1679

>
```

Percentage Bar Chart

The DATA statement in the following program lists the frequency counts for ten class intervals denoted by the numbers 1-10. The program calculates the percentage frequency of each class interval (expressed as a percent of total). Each percentage frequency is rounded to the nearest integer and plotted on a bar chart.

This program demonstrates the usefulness of the statement modifiers (FOR and IF), print functions (TAB), and logical operators (OR).

```
-SBASIC

>LOAD /CHART/
>LIST
10 READ Y(I) FOR I=1 TO 10
20 N=0
30 N=N+Y(I) FOR I=1 TO 10
40 P(I)=100*Y(I)/N,S(I)=INT(P(I)+.5) FOR I=1 TO 10
50 PRINT
60 PRINT TAB(8):"PERCENTAGE BAR CHART"
70 FOR Y=25 TO 1 STEP -1
80 PRINT Y: IF Y=5 OR Y=10 OR Y=15 OR Y=20
90 PRINT TAB(3*I):"XX":IF S(I)>=Y FOR I=1 TO 10
100 PRINT
110 NEXT Y
120 PRINT
130 PRINT IN FORM "9(3%) 4%":I FOR I=1 TO 10
140 DATA 1,5,17,19,30,40,25,21,7,2
>RUN

        PERCENTAGE BAR CHART

                    XX
                    XX
                    XX
                    XX
 20                 XX
                    XX
                XX  XX
                XX  XX
                XX  XX
 15             XX  XX  XX
                XX  XX  XX
                XX  XX  XX  XX
                XX  XX  XX  XX
            XX  XX  XX  XX  XX
 10      XX XX  XX  XX  XX  XX
         XX XX  XX  XX  XX  XX
         XX XX  XX  XX  XX  XX
         XX XX  XX  XX  XX  XX
         XX XX  XX  XX  XX  XX
 5       XX XX  XX  XX  XX  XX
         XX XX  XX  XX  XX  XX  XX
      XX XX XX  XX  XX  XX  XX  XX
      XX XX XX  XX  XX  XX  XX  XX
   XX XX XX XX  XX  XX  XX  XX  XX  XX

   1  2  3  4  5  6  7  8  9  10
>
```

Directory Of Addresses

The file /DIR/ contains the names and addresses of a number of California residents. This program asks the user whose address he wants. The user may type in any part of the person's name (for example, DALE, DALE MOSS or MOSS) and that person's full name and address will be printed. If the name typed is not found, the program will print the last string in the data file which tells the user that the address is not listed there.

Note the value of using the unique string function INDEX in this example (line 110). Since INDEX searches each name in the directory for whatever is typed by the user, any part of a name is acceptable for input. INDEX conveniently returns 0 if it does not find the string for which it has searched.

```
-COPY /DIR/ TO TEL

MR. JOHN B. CAREY,285 COTTLE AVENUE,CAMPBELL
MRS. LESLIE FISHER,1964 HAMPTON DRIVE,DANVILLE
MR. CARL LARSON,985 SOUTH 9 STREET,SAN JOSE
MR. DALE MOSS,1650 SARATOGA AVENUE,SARATOGA
MR. JOHN REY,106 FORMAN STREET,CAMPBELL
MR. DANIEL TORRES,24 SCHARF AVENUE,LOS GATOS
MISS DONNA WILKES,315 SOUTH 3 STREET,SAN JOSE
MR. MICHAEL YOUNG,60 WILSON ROAD,CHESTER
MR. HENRY C. ZIMMER,15 JACKSON STREET,PALO ALTO
THE ADDRESS IS NOT LISTED HERE.

-SBASIC

>LOAD /ADDR/
>LIST
10 STRING N1,N,A,C
20 OPEN /DIR/,INPUT,2
30 PRINT
40 PRINT "ADDRESS OF":
50 INPUT N1
60 PRINT
70 INPUT FROM 2:N
80 IF LEFT(N,3)="THE" THEN PRINT N ELSE 100
90 GO TO 140
100 INPUT FROM 2:A,C
110 IF INDEX(N,N1)=0 THEN 70 ELSE PRINT N
120 PRINT A
130 PRINT C:", CALIFORNIA"
140 CLOSE 2
150 GO TO 20
>RUN
```

(Continued)

```
ADDRESS OF? JOHN REY

MR. JOHN REY
106 FORMAN STREET
CAMPBELL, CALIFORNIA

ADDRESS OF? ZIMMER

MR. HENRY C. ZIMMER
15 JACKSON STREET
PALO ALTO, CALIFORNIA

ADDRESS OF? PEARSON

THE ADDRESS IS NOT LISTED HERE.

ADDRESS OF? DONNA

MISS DONNA WILKES
315 SOUTH 3 STREET
SAN JOSE, CALIFORNIA

ADDRESS OF?
INTERRUPTED IN STEP 50
>
```

Cube Root

This program uses the approximation method to compute the cube root of any number typed by the user. The first approximation is A = N/3, which is compared to the next (closer) approximation $A1 = (2A^3+N)/3A^2$. Each time through the loop (line 40), A stores the last value of A1 and a new approximation is calculated, with the last value of A1 replacing A in the formula. As soon as A1 is equal to A when rounded to eight significant digits (i.e., ABS(A1-A)<1E-9), the program prints the cube root (A1) and the number of passes through the approximately loop (I-1).

Two important characteristics of FOR when used with UNTIL (or WHILE) are illustrated here:

1. A and A1 must be initialized (line 30) because the terminating condition is checked before the loop is entered. Thus, if A had not been initialized, SUPER BASIC would not have been able to define ABS(A1-A) upon first encountering the loop.

2. The value of I upon exit from the loop is that value which caused the exit to occur (i.e., 1 more than the value of I the last time through the loop). For this reason, the number of iterations is I-1 and not I.

```
-SBASIC

>LOAD /ROOT/
>LIST
10 PRINT "TYPE THE NUMBER"
20 INPUT N
30 A=0,A1=N/3
40 A=A1,A1=(2*A↑3+N)/(3*A↑2) FOR I=1 UNTIL ABS(A1-A)<1E-9
50 PRINT "CUBE ROOT:":A1
60 PRINT "NUMBER OF ITERATIONS:":I-1
70 PRINT
80 GO TO 10
>RUN
TYPE THE NUMBER
? 7777
CUBE ROOT: 19.812413
NUMBER OF ITERATIONS: 17

TYPE THE NUMBER
? 0
CUBE ROOT: 0
NUMBER OF ITERATIONS: 0

TYPE THE NUMBER
? -45.9
CUBE ROOT:-3.5804496
NUMBER OF ITERATIONS: 9

TYPE THE NUMBER
?
INTERRUPTED IN STEP 20
>
```

Fundamental Frequency

This program uses the formula

$$F = \frac{.467T}{R^2} \sqrt{\frac{Y}{D\ (\ 1 - P^2)}}$$

to find F, the fundamental frequency of a circular clamped plate. D,Y, and P (the density, Young's modulus, and Poisson's ratio) are read from a DATA statement, and the value of T (the thickness) is requested. Using this data, the program caluclates F for a range of radii (R) from .1 to 1 in steps of .1, from 1 to 10 in steps of 1, and from 10 to 100 in steps of 10. Picture formatting is used to print the results on a file.

Line 90 in this example illustrates the number of instructions that may be given in one statement. A number and the value of a programmer defined function are printed with picture formatting on a file for three distinct range of values.

```
-SBASIC

>LOAD /FREQ/
>LIST
10 READ D,P,Y
20 PRINT "WHAT IS THE THICKNESS OF THE DRUM MATERIAL":
30 INPUT T
40 DEF FNF(X)=(.467*T/X↑2)*SQR(Y/D*(1-P↑2))
50 OPEN /X/,OUTPUT,1
60 PRINT ON 1:"RADIUS","FUND. FREQ."
70 PRINT ON 1
80 A="%%%.%          %%%%%%%%.%%%"
90 PRINT ON 1 IN IMAGE A: R,FNF(R)
   FOR R=.1 TO .9 BY .1,1 TO 9 BY 1,10 TO 100 BY 10
100 DATA 7.8,.3,20E11
>RUN
WHAT IS THE THICKNESS OF THE DRUM MATERIAL? .672

>Q

-COPY /X/ TO TEL

RADIUS          FUND. FREQ.

  0.1          15159139.386
  0.2           3789784.847
  0.3           1684348.821
  0.4            947446.212
  0.5            606365.575
  0.6            421087.205
  0.7            309370.192
  0.8            236861.553
  0.9            187149.869
  1.0            151591.394
  2.0             37897.848
  3.0             16843.488
  4.0              9474.462
  5.0              6063.656
  6.0              4210.872
  7.0              3093.702
  8.0              2368.616
  9.0              1871.499
 10.0              1515.914
 20.0               378.978
 30.0               168.435
 40.0                94.745
 50.0                60.637
 60.0                42.109
 70.0                30.937
 80.0                23.686
 90.0                18.715
100.0                15.159
```

Gross Pay

This program reads from a file up to 100 items of string data, each string consisting of an employee's name and hourly rate. The user is asked to type after each employee's name the number of hours worked by that employee. After extracting numeric information (the hourly rate) from the strings in the data file, the program calculates each employee's gross pay and charts the results.

Note the varied purposes served by the strings read from the file. They are used to request for input (line 90), to compute the gross pay (line 110) and to print the output (line 180). Using the string functions LEFT, VAL, and SUBSTR makes this possible.

```
-COPY /RATES/ TO TEL

ADAMS           $1.50/HR
BENTLEY         $2.75/HR
BROWN           $3.00/HR
DEARBORN        $1.75/HR
FIELD           $1.50/HR
GREER           $4.75/HR
LAMONT          $1.25/HR
MEADOWS         $3.50/HR
MITTY           $5.50/HR
RUFOLO          $3.25/HR
SMITH           $1.50/HR
SOUTHERN        $2.50/HR
SWAN            $2.00/HR
UNDERWOOD       $3.00/HR
END

-SBASIC

>LOAD /GROSPAY/
>LIST
10 TEXT A(100):25
20 DIM H(100),G(100)
30 OPEN /RATES/,INPUT,1
40 FOR I=1 TO 100
50 INPUT FROM 1:A(I)
60 IF A(I)="END" THEN PRINT "TYPE NUMBER OF HOURS WORKED BY:"
   ELSE NEXT I
70 CLOSE 1
80 FOR J=1 TO I-1
90 PRINT LEFT(A(J),15):
100  INPUT H(J)
110 G(J)=H(J)*VAL(SUBSTR(A(J),17,4))
120 NEXT J
130 PRINT
140 PRINT "EMPLOYEE:","HOURLY RATE:","HOURS WORKED:","GROSS PAY:"
150 PRINT
160 F="2%.2% ' HRS' 6B 4$.2$/"
170 FOR J=1 TO I-1
180 PRINT A(J):TAB(31):
190 PRINT IN FORM F:H(J),G(J)
200 NEXT J
>RUN
```

(Continued)

```
TYPE NUMBER OF HOURS WORKED BY:
ADAMS            ? 35.5
BENTLEY          ? 40
BROWN            ? 40
DEARBORN         ? 38.25
FIELD            ? 40
GREER            ? 35
LAMONT           ? 37.5
MEADOWS          ? 40
MITTY            ? 40
RUFOLO           ? 40
SMITH            ? 37
SOUTHERN         ? 32
SWAN             ? 35.25
UNDERWOOD        ? 40

EMPLOYEE:       HOURLY RATE:    HOURS WORKED:  GROSS PAY:

ADAMS           $1.50/HR        35.50 HRS       $53.25
BENTLEY         $2.75/HR        40.00 HRS      $110.00
BROWN           $3.00/HR        40.00 HRS      $120.00
DEARBORN        $1.75/HR        38.25 HRS       $66.94
FIELD           $1.50/HR        40.00 HRS       $60.00
GREER           $4.75/HR        35.00 HRS      $166.25
LAMONT          $1.25/HR        37.50 HRS       $46.88
MEADOWS         $3.50/HR        40.00 HRS      $140.00
MITTY           $5.50/HR        40.00 HRS      $220.00
RUFOLO          $3.25/HR        40.00 HRS      $130.00
SMITH           $1.50/HR        37.00 HRS       $55.50
SOUTHERN        $2.50/HR        32.00 HRS       $80.00
SWAN            $2.00/HR        35.25 HRS       $70.50
UNDERWOOD       $3.00/HR        40.00 HRS      $120.00

>
```

APPENDIX A

ALPHABETIC LIST OF ALL SUPER BASIC STATEMENTS AND CHARACTERISTICS

The following is an alphabetic list of all SUPER BASIC statements.

D - A direct statement
I - An indirect statement
B - Either a direct or an indirect statement
Y - Statement may be modified by statement modifiers
N - Statement may not be modified by statement modifiers

Statement	Statement Type	Modification Possible
BASE	B	Y
CLOSE	B	Y
COMMANDS	B	Y
COMPLEX	B	Y
DATA	I	N
DEF	I	Y
DELETE	D	N
DIM	B	Y
EDIT	D	N
END or STOP	B	Y[1]
FOR	I	Y
GO	D	N
GOSUB	B	Y
GO TO	B	Y
IF	B	Y[2]
INPUT	B	Y
INTEGER	B	Y
LET (Assignment)	B	Y
LIST	D	N
LOAD	D	N
LOGICAL	B	Y
MAT	B	Y
MODIFY	D	N
NEXT	I	Y
ON	B	Y
OPEN	B	Y
PAUSE	I	Y
PRINT	B	Y
QUIT	B	Y
READ	B	Y
REAL	B	Y
REM or !	B	N
RENUMBER	D	N
RESTORE	B	Y
RETURN	B	Y
RUN	D	N
SAVE	D	N

Statement	Statement Type	Modification Possible
STRING	B	Y
TABS	D	N
TAPE	D	N
TEXT	B	Y
VAR=UNDEF	B	Y
VAR=ZERO	B	Y
WRITE	B	Y

1 - But not by FOR
2 - Not applicable.

APPENDIX B

DECLARATION STATEMENT STORAGE ALLOCATION

Declaration Statement	Declares	Words Of Storage Per Variable Or Per Element Of An Array
INTEGER A, B(0:100)	Integer variable or arrays	1*
REAL X,Y(10), Z(N)	Real variables or arrays	2
DIM Z(5), A(2,3)	Integer, real, and string arrays	2
COMPLEX A, B, C(12)	Complex variables or arrays	4
LOGICAL D, G, F(50)	Logical variables or arrays	1/24
STRING M, N, A(2,3)	String variables or arrays	1/3 word per character with a minimum of 2 words.**
TEXT A(20):15, B(3,2):12	String arrays; specifies maximum element length	1/3 per character.**

* The maximum number of words (elements) for an integer array is about 8000 in a program.

** If the number of characters of a STRING or TEXT element is not evenly divisible by 3, the remaining characters of the string occupy one full word.

APPENDIX C

THE EXECUTIVE SYSTEM

Entering The System

To gain access to the Tymshare time sharing system, you must first log in. As soon as the connection to the Tymshare computer is made, the system will type:

PLEASE LOG IN:

Type a Carriage Return. The system replies with:

ACCOUNT:A3 Cr

Type your account number (A3 in this case) followed by a Carriage Return. The system then types:

PASSWORD: Cr

Type your password followed by a Carriage Return. NOTE: The password does not print. The system next types:

USER NAME: JONES Cr

Type your user name followed by a Carriage Return. The system next asks for a project code.

PROJECT CODE: K-123-X Cr

Type your project code followed by a Carriage Return. NOTE: Project codes are optional. If no project code is wanted, simply type a Carriage Return in response to the system's request.

After you have entered the requested information correctly, the system will type:

READY 12/8 11:20
-

The dash in the left hand margin indicates that you are now in the EXECUTIVE. You can call SUPER BASIC or give any EXECUTIVE command.

Calling SUPER BASIC

To call SUPER BASIC, type the EXECUTIVE command

-SBASIC Cr

SUPER BASIC will reply with a > when it is ready to accept a command.

Returning To SUPER BASIC

If for some reason you return to and work in the EXECUTIVE and then wish to continue from where you left off in SUPER BASIC, you can use the CONTINUE command. The program and data that you worked with in SUPER BASIC were not destroyed by the return in the EXECUTIVE.

Example

```
-SBASIC Cr
>
●
●                The user types part of a SUPER
●                BASIC program.
>QUIT Cr
 -
●
●                He does some work in the EXECUTIVE.
●

-CONTINUE Cr
SBASIC
>                He continues to type the program.
```

If the user had typed SBASIC Cr or called any other language instead of giving the CONTINUE command, all of his previous work would have been destroyed.

Listing Files

When the EXECUTIVE command

```
-FILES Cr
```

is given, a complete listing of all your files will be printed, and the type of file will be indicated (SYM for symbolic, BIN for binary).

Example:

-FILES Cr

```
SYM /MORTGAGE/
SYM /JUNK/
SYM /DATA
BIN /BDATA/
SYM /VEN/
SYM /ABC/
```

Deleting Files

If there is no further use for a particular file, delete it by typing:

-DELETE /file name/ Cr

Example

-DELETE /ABC/ Cr
-

A single DELETE command may be used to delete more than one file. The file names must be separated by commas as follows:

- DELETE /PGM/,/JUNK/,/VEN/ Cr
-

Leaving The System

To exit from the Tymshare system, you first must be in EXECUTIVE. To return to the EXECUTIVE from SUPER BASIC, type:

>QUIT Cr

The EXECUTIVE dash will appear in the left margin. Now type

-LOGOUT Cr

followed by a Carriage Return. The system then will type

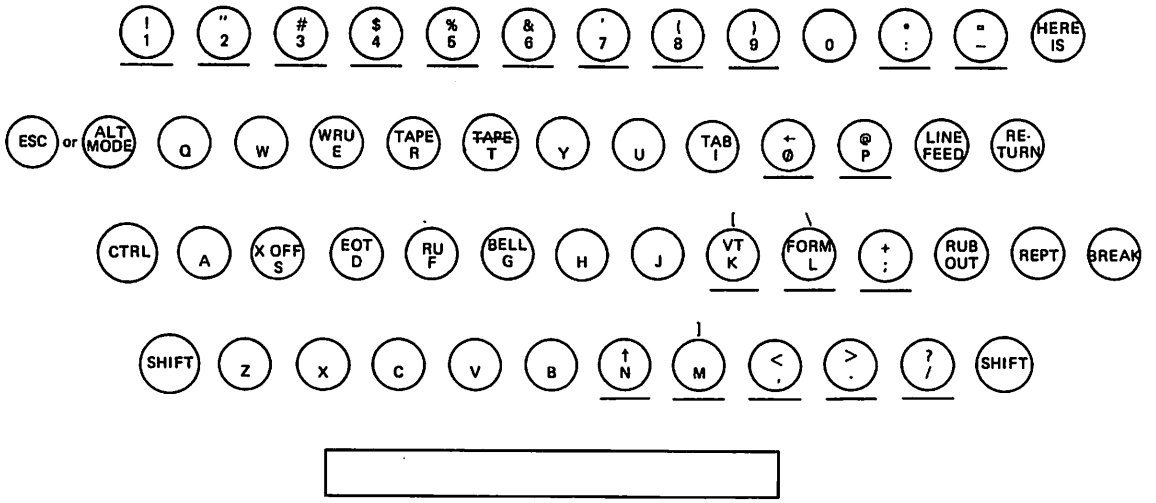
TIME USED 0:37:12

PLEASE LOG IN:

You now may disconnect the line or let another user log in.

APPENDIX D
THE TERMINAL

The Keyboard[1]



SHIFT
Only those keys which are underlined in the keyboard diagram have a shift position. The SHIFT key operates in the manner of an ordinary typewriter. The SHIFT characters are printed as they appear on the upper half of these keys, with the following exceptions:

SHIFT K = [
SHIFT L = \
SHIFT M =]

CTRL (Control)
Any alphabetic key may be pressed in conjunction with this key. The resulting character, called a control character, does not always print on the terminal. Control characters serve a variety of purposes depending on when they are typed. Some languages, for example, use control characters as editing instructions to the computer. In the Tymshare manuals, a superscript c is used to designate control characters; for example, Control D is shown as D^C. Note the following special control characters:

J^C = Line Feed
M^C = Carriage Return

ALT MODE or ESCAPE.
This key is used to abort a command, interrupt the execution of a program, and/or return to the EXECUTIVE. *NOTE: On machines not having either the ALT MODE or the ESCAPE key, use SHIFT K^C.*

HERE IS
Not used in the Tymshare system.

LINE FEED
Advances the paper one line each time it is pressed. When the user is connected to the computer, the system automatically supplies a Carriage Return after every Line Feed.

RETURN (Carriage Return)
Returns the print head to the beginning of a line. The print head goes to the beginning of the **next** line only when the user is connected to the computer; that is, the system automatically supplies a Line Feed after every Carriage Return.

RUB OUT
Used in conjunction with the B.SP. button on the paper tape punch to delete characters punched in error.

REPT (Repeat)
Repeats any character on the keyboard (including a space) when pressed in conjunction with the desired character.

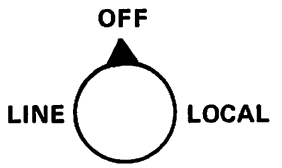
BREAK
DO NOT press this key; it causes a transmission interrupt and possible loss of program and data.

NOTE: Maximum line width is 72 characters.

1 - This is the standard terminal keyboard. On individual machines, some keys may not exist or may be located differently than shown in this diagram.

The ON/OFF Controls

The standard ON/OFF control is a three-position dial located on the front of the terminal and to the right of the keyboard.



Standard ON/OFF Control

LINE
The terminal is ON and ready to be connected to the computer via the phone line. When the connection is made, the terminal is said to be "on line".

OFF
The terminal is OFF.

LOCAL
The terminal is ready for local ("off line") operations; that is, operations to be performed when the terminal is not connected to the computer. *Paper tape may be punched off line.*

The Paper Tape Controls

When the terminal is equipped with a paper tape punch and reader, the controls are on the left side of the terminal.

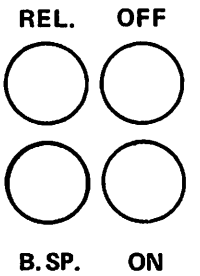
REL.
Releases the paper tape so that the user can pull it through manually.

OFF
Turns the punch OFF.

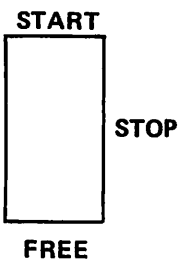
ON
Turns the punch ON to punch the paper tape.

B.SP.
Back spaces the paper tape one frame each time the button is depressed. Used in conjunction with the RUB OUT key on the keyboard to delete erroneous characters.

Punch Controls



Reader Controls



START
Starts and continues paper tape reading.

STOP
Stops paper tape reading.

FREE
Frees the tape reader mechanism so the user can pull the tape through manually.

How To Punch Paper Tape Off Line

The user can punch a paper tape while not connected to the computer. Later the program can be read into SUPER BASIC by means of the TAPE command. The contents of a data file can be punched on tape and read into EDITOR or the EXECUTIVE.

To punch paper tape off line, turn the dial on the front of the terminal to LOCAL and depress the ON button on the paper tape punch controls. Then punch the tape from the keyboard. Note the following special rules:

- Always follow a Line Feed with a Carriage Return.
- Always follow a Carriage Return with a Line Feed.

Example

On line, type:	Off line, type:
20 IF A THEN 100 Lf	20 IF A THEN 100 Lf Cr
ELSE 200 Cr	ELSE 200 Cr Lf

In case of a typing error in a SUPER BASIC statement, delete the incorrect character by typing a ← immediately. Repeat the ← to delete as many characters as required. In addition, an ↑ followed by a Carriage Return deletes an entire line typed in error.

The above editing characters are accepted only in SUPER BASIC. If the tape contains data to be read in EDITOR or the EXECUTIVE, delete an incorrect character by depressing the B.SP. button on the punch controls and then the RUB OUT key on the keyboard. To delete several incorrect characters, press B.SP. as many times as necessary and then RUB OUT the same number of times.

TYMSHARE MANUALS

Instant Series

CAL
SUPER BASIC
EDITOR

Reference Manuals

EXECUTIVE
CAL
SUPER BASIC
EDITOR
FORTRAN IV
FORTRAN II
LIBRARY
COGO
ECAP
ARPAS/DDT
BRS

TYMSHARE™

TYMSHARE, INC.

SAN FRANCISCO
745 Distel Drive
Los Altos, California 94022
Telephone: 415/961-0545

LOS ANGELES
334 East Kelso Street
Inglewood, California 90301
Telephone: 213/677-9142

SAN DIEGO/ORANGE COUNTY
4630 Campus Drive, Suite 209
Newport Beach, California 92660
Telephone: 714/540-5940

NEW YORK
464 Hudson Terrace
Englewood Cliffs, New Jersey 07632
Telephone: 201/567-9110

DALLAS
2355 Stemmons Bldg., Suite 1010
Dallas, Texas 75207
Telephone: 214/638-5680

SEATTLE
2200 6th Avenue, Suite 810
Seattle, Washington 98121
Telephone: 206/MA 3-8321