Price $1.50

TYMSHARE MANUALS

INSTANT SERIES

SUPER BASIC

October 1968

TYMSHARE, INC.
525 UNIVERSITY AVENUE, SUITE 220
PALO ALTO, CALIFORNIA 94301

DIVISION OFFICES

Los Altos, California ■ Inglewood, California ■ Arlington, Virginia

DISTRICT OFFICES

Newport Beach, California ■ Englewood Cliffs, New Jersey
San Francisco, California ■ Dallas, Texas ■ Seattle, Washington

CM4

CONTENTS

CONTENTS (Continued)

TYMSHARE MANUALS
SYMBOL CONVENTIONS

The symbols used in this manual to indicate Carriage Return, Line Feed, and ALT MODE/ESCAPE are as follows:

Carriage Return:	⤶	
Line Feed:	↴	
ALT MODE/ESCAPE:	⊕	*NOTE: This symbol will be printed as many times as it is required to hit this key.*

Action At The Terminal

To indicate clearly what is typed by the computer and what is typed by the user, the following color code convention is used:

Computer: Black **User: Red**

PREFACE

This manual introduces both programmers and non-programmers to Tymshare SUPER BASIC and to the Tymshare time sharing system. The manual is designed as an introduction to programming in SUPER BASIC and includes the fundamental SUPER BASIC commands necessary for effective programming. Careful study of the manual will enable the non-programmer to solve a variety of problems using the computer.

SUPER BASIC is presented as an introductory programming language because, in addition to being easy to learn and use, it is also a very powerful language. SUPER BASIC is a conversational language; that is, it prompts you and tells you what is wrong if you make an error. Each program statement is checked immediately and an error diagnostic is returned if the statement is incorrect. The SUPER BASIC editing facilities then allow you to correct the statement immediately. When all of the program statements have been checked the program may be saved and then reused at any time.

SUPER BASIC may be used to perform any numeric computations and contains a full complement of mathematical and trigonometric functions to make programming easier. SUPER BASIC also may be used to manipulate alphanumeric strings. The information computed then may be output using the standard formats or using a special programmer defined "image" picture format.

SECTION 1
INTRODUCTION TO SUPER BASIC

As an introduction to programming in SUPER BASIC we have written a simple program which computes gas mileage when the initial and final odometer readings for any given amount of gasoline are known.

```
10 INPUT I,F,G
20 T = F–I
30 M = T/G
40 PRINT T;"MILES",M;"MILES/GAL"
```

We will now go through a step-by-step analysis of how this program was written. If you can understand the techniques used in writing this program, you should have no trouble writing many other programs which use these same basic techniques.

Defining The Problem

The first and probably most important step in programming is writing a clear, concise definition of the problem. A computer is designed to follow sets of simple commands which appear in logical sequence. It is during this first step that you should organize your problem into small sections that can be written in SUPER BASIC.

The easiest way to define a problem for programming is to separate the problem into the following three sections.

1. OUTPUT - What information is desired? This section includes the answer to our problem and anything that we wish to have printed, in this case, the gas mileage.
2. COMPUTE - What computations must be made to find the above information?

   Gas Mileage = Total miles travelled divided by the amount of gas used.

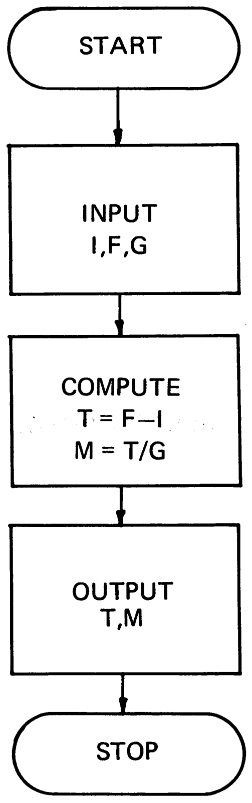
   Total Miles = Final odometer reading minus initial odometer reading.
3. INPUT - What information must be supplied to solve the problem?

   Initial odometer reading (in miles).

   Final odometer reading (in miles).

   Amount of gas used (in gallons).

Flowcharting The Problem



Entering The Computer And Calling SUPER BASIC

The process of calling the computer and telling it who you are is called logging in. After the connection has been properly made, the computer replies with:

PLEASE LOG IN: ↵	Type a Carriage Return.
ACCOUNT: A3 ↵	Type your account number (A3 in this case) followed by a Carriage Return.
PASSWORD: ↵	Type your password followed by a Carriage Return. The letters in the password will not be printed on the page.

USER NAME: JONES ↵	Type your user name followed by a Carriage Return.
PROJ CODE: K-123-X ↵	Type your project code if desired. *NOTE: A project code is optional. If no project code is wanted, simply type a Carriage Return in response to the system's request.*
READY 12/8 11:20	You are now properly connected with the computer and may proceed by calling the subsystem desired, which in this case is SUPER BASIC.
– SBASIC ↵	When the EXECUTIVE dash (–) appears, call SUPER BASIC by typing SBASIC and a Carriage Return. SUPER BASIC will respond with a >. You then may type any of the SUPER BASIC commands and start writing your program.
>10 INPUT I,F,G ↵	Now the steps of the program are typed. Each step is preceded by a line number. The program will be executed in the order specified by the numbers. Typing a statement with the same number as a preceding statement will cause the preceding statement to be replaced. The first step contains the command INPUT which, during execution, will request that the values of I, F, and G be entered from the terminal.
>20 T = F–I ↵ >30 M = T/G ↵	The arithmetic computations shown here are called replacement statements. The first replacement computes the total mileage and the second computes the gas mileage.
>40 PRINT T;"MILES",M;"MILES/GAL" ↵	The PRINT command will print on the terminal the values of the variables specified as well as any literal text that is enclosed in the double quote marks. (The difference between the comma and the semicolon will be explained later in this section.)
>RUN ↵	This command, which must be used without a line number, starts execution of the program at the lowest numbered line in the program.
? 1125.7 ↵ 1764.1 ↵ 40.9 ↵	The first statement of the program INPUT I,F,G is now executed. SUPER BASIC types a ? to indicate that it is waiting for terminal input. The first value, 1125.7, is typed and then terminated with a Carriage Return. The next two values are entered in the same manner.
638.4 MILES 15.608802 MILES/GAL	The values of T and M are computed. The PRINT statement causes the values of the variables and the literal text which was enclosed in the quote marks to be printed.
>QUIT ↵	Control will be returned to you with a > after all of the commands in the program have been executed. QUIT causes an exit from SUPER BASIC back to the EXECUTIVE.

```
– LOGOUT ↩
TIME USED 0:5:17
PLEASE LOG IN:
```

The EXECUTIVE command LOGOUT disconnects you from the computer and types the amount of time you have used, 5 minutes and 17 seconds in this case. PLEASE LOG IN: is repeated. Another person may then enter the system on that terminal, or you may hang up the phone.

Computer Programming

What is a computer program?

A computer program is a set of simple instructions written in a language the computer can understand which tells the computer how to solve a problem.

What type of language must be used?

The language used to write a program depends upon the problem to be solved and the computer being used. Before it can be executed, a program must be presented in a language understood by the computer. The actual machine language of a computer is generally very difficult to learn and use. Therefore, a group of higher level computer languages has been developed. SUPER BASIC is one of these languages, and is essentially a combination of simple English and elementary algebra.

How is a program written?

A program normally is written in two basic steps. The first step includes all of the preliminary defining of the problem, step by step analysis, flowcharting, and thinking that must be done before you can attempt the second step - the actual coding of the problem.

Defining the problem.

Before you can write a program to solve a problem, you must know exactly what the problem is and how to solve it. A computer can only follow your instructions - it has no intuitive knowledge. Your first task is to determine what output you need (what answers you wish to compute using the data available), what computations must be made, what alternatives exist in certain cases, and what original data you need and have to work with.

After you have decided what information is needed from your data and how to go about computing it, the next step is to try to place your information in a logical sequence so the computer can do the computations. With simple problems such as those first encountered in this text, this is easy. With complicated problems, the best approach probably is to break the problem into sections and work on it a section at a time.

Very often the clearest method of representing the logical sequence (flow) of a program is to picture it with a flowchart. Flowcharts consist of a number of boxes connected by lines. Within each box is a brief statement of an operation to be performed. The interconnecting lines, with arrows attached, show the various paths the solution may take. A flowchart shows clearly the logical flow of steps necessary to solve the problem. If many decisions are to be made or many alternatives exist, a graphical flowchart makes these alternatives easier to follow. Flowcharts may be simple, showing only the vaguest outline of the various alternatives, or very detailed. The greater the detail in the flowchart, the easier the actual programming will be.

Writing in SUPER BASIC

As yet you probably have not been introduced to SUPER BASIC. This manual is designed to teach it to you. SUPER BASIC is as much a language as any other that you have used. To achieve successful communication, the proper syntax and grammar must be used. If you do not "speak" SUPER BASIC correctly, the computer will not understand what you are saying and will tell you that you have made an error.

Checking your program (Debugging).

After the program has been written and put into the computer, your next task is to try to run the program. If the program runs, only one step remains and that is to make sure that the answers given are correct.

If the program will not run, SUPER BASIC will generally give you an error diagnostic telling you what is wrong. If your program runs but the answers are incorrect, there are two things that you may do. The first is to run the program part by part in sequential order. In this way you can isolate your problem. An alternate method is to try stepping through the program as the computer would using the simplest cases you can think of. Read each step in the program and execute it. Do only exactly what you have told the computer to do, not what you know should be done. As you step through the program you will probably find the errors.

Writing The Program In SUPER BASIC

A SUPER BASIC program is written in simple logical steps which are called statements. A statement may be up to 256 characters long and always must be terminated by a Carriage Return. Statements normally appear on a single physical line; however, they may be continued on the next line by pressing a Line Feed instead of a Carriage Return.

Direct And Indirect Statements

A statement may be either direct or indirect.

A direct statement does not have a line number. It is executed as soon as the Carriage Return is pressed, and is not saved after execution.

Example Using Direct Statements

```
>INPUT A ↵         INPUT statement executed
? 34 ↵             immediately.
>X = A↑2 ↵         X computed immediately.
>PRINT X ↵         PRINT statement executed
 1156              immediately.
>
```

Indirect statements have line numbers. Any integer within the range 0 to 999999 may be used as a line number. Indirect statements are always used when writing a program; they are stored in the program in line number sequence and are referred to by their line numbers. When the program is executed the computer will step through the entire set of indirect statements in the specified sequence. The entire program (all of the indirect statements) will be saved after execution so that the program may be executed again or modified.

Example Using Indirect Statements (Storing The Statements In A Program)

```
>10 INPUT A ↵      Stored
>20 X = A↑2 ↵      Stored
>30 PRINT X ↵      Stored
>RUN ↵             RUN starts execution of
                   the program at line 10.
? 34 ↵             INPUT statement exe-
                   cuted.
 1156              PRINT statement exe-
                   cuted.
>RUN ↵             Program executed again.
? 122 ↵            INPUT statement executed
                   again.
 14884             PRINT statement executed
                   again.
>
```

INPUT Statement

To input information (data) from the terminal, SUPER BASIC uses the command INPUT with a list of the variables to be given values. In our sample program the variables used were I, F, G.

10 INPUT I,F,G

An INPUT statement may be used either directly (without a line number) or indirectly (with a line number).

Variables

A variable is defined as a symbol (I,F,G), which is used to represent a numeric or string value which may be changed during execution of the program. Legal variables in SUPER BASIC include any single letter (A-Z), any single letter/single number combination (A0-Z9), and any single letter/dollar sign combination (A\$-Z\$).

Typing Numeric Data Into Variables

When an INPUT statement is executed, the system will respond with a ? to indicate that it is waiting for you to input a value. *NOTE: Do not confuse the ? in response to an INPUT statement with the ? error diagnostic.* You should then type the number you wish to store in the variable followed by a Carriage Return. If more than one variable is to be given a value in response to a single INPUT command, the value should be terminated by a Carriage Return if you wish the following value printed on the next line. If you wish the following value to be printed on the same line, type a space or a comma after the value. The system will wait until a value has been input for each variable specified in the INPUT statement and then proceed through the program.

Example

```
>INPUT A,B,C
? 123 ↵
700,12.4 ↵
```

If you make a mistake while typing the value, you may delete the preceding character by typing a Control A (A^C)[1] as long as you have not yet terminated the value with a Carriage Return, a comma, or a space. You may then type in the correct character. In like manner, you may delete the entire value by typing a Control W (W^C). You may then retype the entire number. *NOTE: After the terminating Carriage Return, comma, or space has been input, the value may not be changed by using the control characters A and W.*

You may restart input for the entire variable list in the INPUT statement by typing a Control Q (Q^C). A Q^C may not be used after the value of the last variable in the list has been terminated.

NOTE: A^C prints a ←, W^C a \, and Q^C an ↑.

Numbers

Numbers in SUPER BASIC may be represented in three ways. Numbers may be expressed as integers (whole numbers without a decimal point), as decimals (numbers with a decimal point), or in scientific notation. Scientific notation is used with very large or very small numbers. For example, if we wish to input the number of miles travelled by a beam of light in a year we could use 6E12 instead of 6,000,000,000,000. The E in this notation tells us that the number to the left of the E (which may be in integer or decimal form) is multiplied by 10 raised to the power of the number appearing to the right of the E (which may be a positive or negative integer).

SUPER BASIC will accept both positive and negative numbers as large as 5^{76} and as small as 5^{-76}. Internally, SUPER BASIC will retain 11 places of significance (all numbers are rounded to 11 significant digits). SUPER BASIC will output up to 8 significant digits if a PRINT statement is used, and up to 11 significant digits if a PRINT IN IMAGE statement is used. With a PRINT statement, SUPER BASIC will output the number as an integer if it is an integer value within the range ±1 through ±99999999, and as a decimal if it is a decimal value within the range ±0.1 through ±9999999.9. If the number does not fall within either of the ranges specified above, it will be output in scientific notation. Output using PRINT IN IMAGE will be explained later in the manual.

Examples

Number Input	**Number Output**
1123.45678	1123.4568
+.1234	.1234
33333333.33	33333333
6666666666666666	.66666667E+16
.00123456789	.12345679E–02
1234567890.E25	.12345679E+35
–5.9E+23	–.59E+24
.1234E04	1234
1234E–04	.1234

Strings

In SUPER BASIC a string of alphabetic and/or numeric characters also may be assigned to (stored in) a variable. This makes it possible to store names, addresses, mixed alphabetic and numeric text, and similar information in a single variable. Any legal variable may be used as a string variable. *NOTE: A variable may not be used to store both a numeric value and a string simultaneously.*

A string is indicated by enclosing the characters in double quote marks. A string may be of any length and may contain any group of characters. *NOTE: Blanks (space bar) are considered to be characters.* If the string is longer than 72 characters, a Line Feed may be used to continue the string on the next line. The Line Feed will not be accepted as part of the string. The following expressions are strings.

"G.J.JONES, 9350 ALPHA DR."
"123, RTS, 79549"
"PRINCIPAL $25000.00"

Typing Strings Into Variables

Strings are input in response to the INPUT command in almost the same manner as numeric values, except of course, that the string must be enclosed in double quote marks. When the INPUT statement is executed and the question mark appears, type the string enclosed in double quote marks followed by a Carriage Return, comma, or space.

Example

```
>INPUT A,B,C ↵
? "J.J.JONES", 2500 ↵
"DOWN PAYMENT $500 PAID 1/7/67 ↓
 BALANCE DUE 1/7/69" ↵
>
```

In the above example, the string J.J.JONES will be stored in the variable A, the numeric value 2500 will be stored in the variable B, and the string DOWN PAYMENT $500 PAID 1/7/67 BALANCE DUE 1/7/69 will be stored in the variable C.

If you make a mistake while typing a string into the variable you may use a Control A (A^C) to delete the preceding character, as long as you have not typed the end quote marks. The Control A may be used repeatedly to delete any number of characters back to

1 - To type a control character depress and hold the CTRL key and then type the desired character.

but not including the quote marks. If you wish to delete the quote marks you must use a Control Q (Q^C) which will restart the INPUT command. You must then input all of your data again.

Replacement Statements

Replacement statements are used to assign values to variables. The value assigned may be a numeric value, an arithmetic expression which will be evaluated to a single numeric value, or a string (enclosed in quotes). Replacement statements appear in the following form:

single variable = arithmetic or string expression

In our example there are two replacement statements. The first one computes the total mileage:

20 T = F–I

The second replacement statement computes the gas mileage:

30 M = T/G

A string replacement would appear as follows:

R = "G.R.BROWN"

The term replacement statement is used because the computer actually replaces the value stored in the variable on the left (T) with the computed value of the arithmetic expression (F–I). The equal sign (=) should not be read as "T is equal to" but as "The value stored in T is replaced by". In later examples we shall run into replacement commands that are mathematically invalid if the equal sign is interpreted to mean equality; that is, (T = T + 1) but which make sense if the statement is read T "is replaced by" T + 1.

A replacement statement may be used both directly (without a line number) and indirectly (with a line number).

Arithmetic Operators And Their Priority Of Execution

Six different arithmetic operations may be performed in SUPER BASIC.

Operation	Symbol	Precedence
Exponentiation (raising a number to a power)	↑	computed first
Unary minus (negative sign)	–	computed second
Division Multiplication	/ *	computed third
Subtraction Addition	– +	computed last

In general, SUPER BASIC follows the established arithmetic rules. SUPER BASIC does require, however, that an operator always be specified; that is, 4X is not a legal expression. In SUPER BASIC it must be written 4*X.

Priority Of Execution

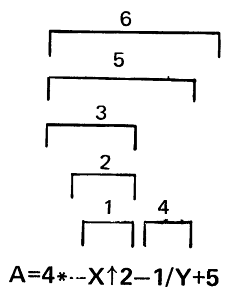
In normal algebraic expressions only one operation can be performed at a time. The computer also can perform only one operation at a time, and therefore since most expressions contain more than one arithmetic operator, some order or priority of computation (execution) must be established. The order of computation used in SUPER BASIC is the same one found in simple algebra.

All arithmetic expressions in SUPER BASIC are scanned from left to right. If any exponentiation is encountered, it is computed first. Next, all negative numbers are assigned a negative value (unary minus is computed).

The system then checks for any * or / which is executed from left to right. Finally the system will compute all + or –, again working from left to right.

Example

The operators in the expression
A = 4*–X↑2–1/Y+5
would be executed in the following order.



1 - Exponentiation	
2 - Unary minus	
3 - Multiplication	left-to-right
4 - Division	
5 - Subtraction	left-to-right
6 - Addition	

We have $A = 4(-(X^2)) - \frac{1}{Y} + 5$

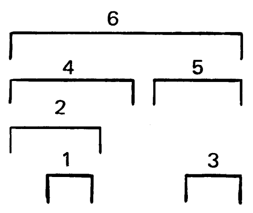
Parentheses

The normal order of execution may be altered by using parentheses in an expression. Anything that appears in parentheses must be evaluated first. The inner set of parentheses is always evaluated first.

Notice how parentheses in an equation alter the normal order of execution.

Example

A = (4*(–X))↑2–1/(Y+5) is executed starting with the inner set of parentheses.



A = (4*(-X))↑2-1/(Y+5)

1 - First set of parentheses, Inner pair (Unary minus)
2 - First set of parentheses, Outer pair (Multiplication)
3 - Second set of parentheses (Addition)
4 - Exponentiation
5 - Division
6 - Subtraction

in this case $A = (4(-X))^2 - \frac{1}{Y+5}$

PRINT Statement

To output information (print it on the terminal) SUPER BASIC uses the PRINT statement. The PRINT statement may be used with a list of variables, arithmetic expressions, and/or string expressions. When the PRINT statement is used with a variable list, the numeric or string value stored in each variable will be printed when the statement is executed. If the PRINT statement is used with an arithmetic expression, the expression will be evaluated and its value printed. When the PRINT statement is used with a string (which must be enclosed in quotes) the string will be printed when the statement is executed. PRINT statements may be used both directly and indirectly.

Examples

```
>A = 12.57↵

>B = "GAL"↵

>PRINT "THE ANSWER IS";A;B↵
THE ANSWER IS   12.57   GAL

>PRINT 5*3↑2,(9+8)/7↑2↵
 45               .34693878

>
```

The variables, arithmetic expressions, and/or string expressions may be separated in a PRINT statement by either a comma (,), a semicolon (;), or a colon (:). The symbol used to separate the variables and expressions specifies the field format in which the values and/or strings are to be printed. The comma is used to specify a zoned field format which will output the data using five fifteen-space fields per line if possible. If the number or string is larger than 15 spaces, the data or string will be continued in the next field. A semicolon is used to specify a packed field format which will output the numbers or strings closer together than the zoned format. The exact format used depends upon the number of characters in the previous number or string. A colon (:) is used to specify a compressed field format which will print one value immediately after another leaving no space between the values. *If the second value printed is a positive number a space will be printed, because although the positive sign is not printed, a single space is left for it.*

Example

```
>A = 1.5↵

>B = 7965↵

>C = "STRING"↵

>G = 40.9↵

>PRINT A,B,G↵
 1.5            7965             40.9

>PRINT A;B;C↵
 1.5     7965     STRING

>PRINT A:B:C↵
 1.5  7965STRING

>
```

RUN And QUIT Commands

The RUN and QUIT commands are control commands. RUN is always used directly (without a line number). RUN is used to start execution of a program. When used, RUN erases all variable values stored in the computer memory and then transfers control to the lowest numbered line in the program. QUIT is used to return to the EXECUTIVE, and can be used either directly or indirectly.

SAMPLE PROGRAM - COST OF PRINTING

Define The Problem

The problem is to compute the total cost of printing a definite number of booklets and the cost per booklet, given the number of copies needed, the number of sheets printed on both sides, and the number of sheets printed on one side.

Input

1. Number of copies to be printed (N)
2. Number of pages printed on both sides (B)
3. Number of pages printed on one side (S)

Compute

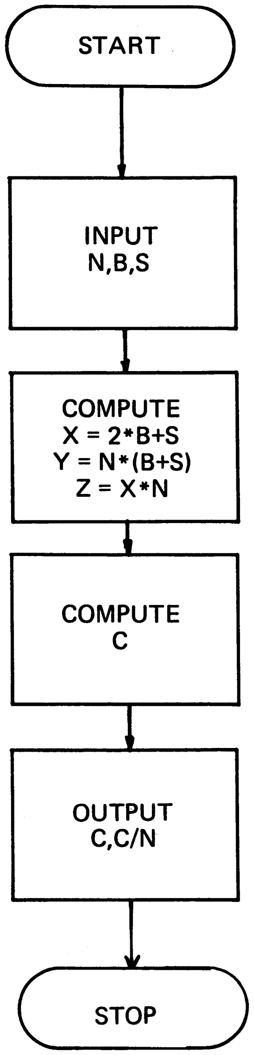
1. Total number of pages printed per booklet
   $X = 2B+S$
2. Total amount of paper needed
   $Y = N(B+S)$
3. Total number of pages printed
   $Z = X \cdot N$
4. Total cost of printing
   $C = 1.59X+.01765Y+.002883Z$
5. Cost per booklet
   C/N   Computed during the PRINT command.

Output

1. Total cost of printing (C)
2. Cost per booklet (C/N)

SUPER BASIC Program And Sample Execution

```
– SBASIC ↵
> 10 INPUT B,S,N ↵
> 20 X = 2*B+S ↵
> 21 Y = N*(B+S) ↵
> 22 Z = X*N ↵
> 30 C = 1.59*X+1.765E–02*Y+2.883E–03*Z ↵
> 40 PRINT "TOTAL COST = ";"$":C ↵
> 50 PRINT "COST PER BOOKLET = ";"$":C/N ↵
> RUN ↵
? 20,4,40 ↵
TOTAL COST =       $ 91.97808
COST PER BOOKLET =  $ 2.299452
> QUIT ↵
–
```



SECTION 2
ADDITIONAL SUPER BASIC FACILITIES

In this section we will introduce:

- The control command GO TO;
- A method of modifying commands using the IF modifier;
- Looping using the FOR and NEXT statements;
- The concept, use, and dimensioning of arrays;
- Picture formatting using the PRINT IN IMAGE statement;
- A method of storing data in your program in the form of a "data block", and then accessing the "data block";
- A method of putting comments in your program;
- A partial list of the standard mathematical functions available in SUPER BASIC.

GO TO Command

GO TO is a control command. When a GO TO statement is executed, control will be transferred to the line number specified in the statement. The general form of the statement is:

GO TO line number

GO TO may be used indirectly in a program (in which case it is preceded by a line number) or directly (no line number).

When an indirect GO TO command is executed in the program, control is transferred immediately to the line number specified, thus interrupting the normal sequential order of execution. GO TO may be used to "loop" back to a line which has been previously executed, or to skip a group of statements in the program. An indirect GO TO may be used with an IF modifier to transfer control only under certain conditions.

Used directly, GO TO starts execution of the program at the line number specified. A direct GO TO statement may be used in place of RUN to start execution. *NOTE: A direct GO TO will not erase the variable values stored in memory as does RUN.*

IF Modifier

The IF modifier defines the conditions under which the command it modifies will be executed. Any SUPER BASIC command modified by an IF modifier will be executed if and only if the conditions specified by the modifier are met.

```
10 GO TO 50 IF R = 20
```

The IF modifier used in this command will transfer control to line number 50 only if R = 20; otherwise the GO TO command will be ignored and the next statement in sequence will be executed.

IF is commonly used with GO TO to transfer control to another part of the program only under certain conditions. By using this type of conditional transfer, you can make decisions in your program based on previous numeric calculations. For example, you might check to see that a number is not negative before you try to compute its square root.

```
>10 INPUT A,B,C
>20 GO TO 50 IF B–C#0
>30 PRINT "DENOMINATOR IS ZERO,↴
  INPUT NEW DATA"
>40 GO TO 10
>50 PRINT A↑2/(B–C)
```

The Logical Operators

In addition to the equal relationship, five other relational operations may be specified in SUPER BASIC.

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

Relational expressions may be combined using AND or OR. If an AND is used to connect two relational expressions, both of the specified relationships must be true before the command can be executed. If the OR is used as a connector, only one of the relationships must be true.

```
>PRINT H IF H>=50 AND H<=75
```

The IF modifier in this command will limit the range of numbers printed from 50 through 75 inclusive.

Example

```
>10 INPUT H ↩
>20 PRINT H↑2 IF H>=50 AND H<=75 ↩
>RUN ↩
? 45 ↩        H is not within the specified range so H↑2
              is not typed.
```

```
> RUN ↵
? 55 ↵      H is within the specified range so H↑2 is
            typed.
 3025
>
```

FOR/NEXT Commands

Often you may wish to execute the same series of statements for a specific list of values. This type of repetition is common in programming and is referred to as a loop.

The FOR and NEXT commands create loops in SUPER BASIC. The general form of these commands is:

FOR variable = limit BY interval TO limit
statements to be repeated
NEXT variable

The FOR and NEXT commands are always used indirectly (with a line number), and always must be used together. FOR specifies the initial value to be used, the interval by which it is to be incremented, and the final or terminating value. NEXT ends the loop and transfers control back to the FOR statement which then increments the value of the variable and repeats the "loop" using this new value. The loop will be repeated using the new incremented value until the limit specified is reached, at which time control will be transferred to the statement following NEXT. *NOTE: If a NEXT statement is not used, the statements in the "loop" will be executed only for the initial value assigned to the variable.*

Example

```
> 10 FOR I = –1 BY .50 TO 1 ↵
> 20 PRINT "I = ";I,"I↑2 = ";I↑2 ↵
> 30 NEXT I ↵
> 40 PRINT "END" ↵
> RUN ↵
I =   –1          I↑2 =   1
I =   –.5         I↑2 =   .25
I =     0         I↑2 =   0
I =    .5         I↑2 =   .25
I =     1         I↑2 =   1
END

>
```

The variable used in the FOR and NEXT commands must be a non-subscripted variable. The limits and the interval may be specified using a numeric expression or predefined variable as well as a specific numeric value.

For example, the following is a valid FOR statement

FOR X$ = A–12 BY –.50 TO R↑3–5

Note that a negative interval was specified in the FOR statement above.

If the desired interval is 1, it need not be specified. The following statements are equivalent:

10 FOR R = 1 BY 1 TO 10
10 FOR R = 1 TO 10

NOTE: If the initial value exceeds the final value, the statements in the loop will be ignored and control transferred to the statement following the NEXT.

Two alternate forms of the FOR command also may be used. The first uses the same basic form but sets up a terminating condition using UNTIL and a conditional expression instead of TO with a definite final limit.

Example

```
> 10 FOR I = 2 BY 2 UNTIL I↑2>100 ↵
> 20 PRINT "I = ";I,"I↑2 = ";I↑2 ↵
> 30 NEXT I ↵
> RUN ↵
I =     2       I↑2 =     4
I =     4       I↑2 =     16
I =     6       I↑2 =     36
I =     8       I↑2 =     64
I =     10      I↑2 =     100

>
```

Any terminating condition may be set up. An AND or OR may be used to create a complex terminating condition. For example,

FOR I = X UNTIL ABS(Z↑2–3)>25 OR R = 0

The second alternate form lists the actual values to be assigned to the variables.

Example

```
> 10 FOR X = 1,7,11,99 ↵
> 20 PRINT X–3; ↵
> 30 NEXT X ↵
> RUN ↵
–2     4      8      96
>
```

NOTE: The above forms of the FOR statement may be combined. For example, FOR X = 1,3,5 TO 10,17,20 BY 5 UNTIL X = 50.

Nesting Loops

It is often advantageous to nest loops; that is, to place one loop within another. If loops are nested, the inside loop always must be closed before the outside loop is closed. Also, if more than one loop is open at one time, the loops must never have the same variable name. The following examples illustrate correct and incorrect nesting of loops.

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
```

If two nested loops are closed at the same time, you may use a single NEXT statement. The loop that is closed first must be listed first. The following statements are equivalent.

```
FOR X = 1 TO 10        FOR X = 1 TO 10
FOR Y = 5 TO 10        FOR Y = 5 TO 10
NEXT Y                 NEXT Y,X
NEXT X
```

Arrays

Essential to an effective use of loops is an understanding of the concept and use of arrays. An array is simply a set or list of subscripted variables such as A_1, A_2,..A_n, which are known as array elements. Each subscripted variable (array element) is considered to be a unique variable, and thus each may be used to store a single value. An array element is identified by an array name and, in parentheses, a subscript which indicates the position of the element in the array. For example, the subscripted variables A(7) and R(4,3) both could represent array elements. Legal array names include any single letter A through Z, and any single letter/dollar sign combination, A$ through Z$. The subscripts generally are integers, although a non-integer value, a numeric expression, or a variable that has previously been given a value may be used, such as, A(11),B(I),R$(3–4*T).

Using arrays inside of loops enables you to save all of the computed values because, by changing the subscripts, you can change the variable each time the loop is executed.

Example

```
10 FOR I = 1 TO 3
20 A(I) = I↑2
30 NEXT I
```

When these statements are executed, A(1) will be given a value of 1, A(2) a value of 4 and A(3) a value of 9.

If more than one subscript is used, the subscripts must be separated by commas; for example, A(I,J). The number of subscripts indicates the dimension of the array. One subscript such as A(I) would indicate a one dimensional array which may be thought of as a list of items. An array with two subscripts such as A(I,J) may be thought of as a table or chart having both rows and columns where I specifies the row and J the column. To find a particular value in a table, both the row and column must be specified. For example, if the table appeared as follows:

		(J)		
		1	2	3
(I)	1	1.1	1.5	1.3
	2	1.7	1.9	1.4

the value 1.7 would be stored in the array element A(2,1) and the value 1.3 would be stored in the element A(1,3).

Dimensioning Arrays

All arrays, except those with subscripts with the range (1) through (10) or (1,1) through (10,10) must be dimensioned to reserve sufficient space in memory for all of the elements of the array. SUPER BASIC automatically reserves space for a one dimensional array with 10 elements; namely, A(1) through A(10), or a two dimensional array with 100 elements; namely, A(1,1) through A(10,10). The dimension statement DIM specifies the array name, indicates the number of memory locations needed, and specifies the first and last subscripts to be used as follows:

DIM array name (first subscript:last subscript)

For example, the statement

DIM A(50:100)

would reserve 50 places in memory for the array elements A(50) through A(100).

In SUPER BASIC, the subscript boundaries also may be specified by using an arithmetic expression or a predefined variable such as DIM I(J). If the first subscript is one, only the last subscript need be specified since SUPER BASIC assumes the first subscript to be one unless it is given a specific value. For example, the following two statements are equivalent:

```
DIM A(1:25)
DIM A(25)
```

A number of arrays may be dimensioned in a single DIM statement. The arrays must be separated by commas as follows:

DIM A$(12),B(−5:25),R(I)

Multidimensional arrays are dimensioned using an extended version of the DIM statement as follows:

DIM array name(first sub$_1$:last sub$_1$,first sub$_2$:last sub$_2$,...)

For example, the DIM statement

DIM R(0:6,−4:4,10)

would set up a three dimensional array with the first element R(0,−4,1) and the last element R(6,4,10).

Example Using Arrays And Loops

A company has 50 employees, numbered 1-50 (I). Each employee's gross pay is stored in the array element G(I). His net pay (20% tax deducted) is stored in the array element N(I).

```
>10 DIM G(50),N(50) ↵
>20 FOR I = 1 TO 50 ↵
>30 INPUT G(I) ↵
>40 N(I) = G(I)−.20*G(I) ↵
>50 PRINT I;G(I);N(I) ↵
>60 NEXT I ↵
>RUN ↵
? 250.00 ↵
 1     250        200
? 200 ↵
 2     200        160
? 360 ↵
 3     360        288
? 150 ↵
 4     150        120
? 175 ↵
 5     175        140
? 1222 ↵
 6     1222        977.6
? 790 ↵
 7     790        632
?   ⊕⊕             Terminated with ALT MODE/
                   ESCAPE key.
INTERRUPTED IN STEP 30
>
```

! (Comments)

To add a comment to your program, type a !, then your comment and a Carriage Return. Any combination of characters may be used in a comment. Comments are used primarily for program explanation and documentation and may be inserted at any point in the program. **Comments are not printed during execution.** Comments may follow any statement or they may occupy a separate line. Remember, however, that only comments in indirect statements will be saved on a file.

Example

```
>10 PRINT "THIS PROGRAM COMPUTES THE ↴
 AREA OF 3 CIRCLES ↵
>20 !THE STANDARD FORMULA FOR AREA ↴
 IS USED ↵
>40 PRINT " RADIUS           AREA" ↴
                        !THIS IS A HEADING ↵
>50 FOR R = 5 BY 5 TO 15 ↵
>60 A = 3.1416*R↑2 ↵
>70 PRINT R,A ↵
>80 NEXT R ↵
>RUN ↵
THIS PROGRAM COMPUTES THE AREA OF 3
                              CIRCLES
 RADIUS           AREA
 5                78.539816
 10               314.15927
 15               706.85835

>
```

PRINT IN IMAGE - Picture Formatting

In addition to the three conventional output formats, zoned (,), packed (;), and compressed (:), SUPER BASIC also contains picture formatting facilities with the PRINT IN IMAGE command. Using picture formatting you may output your data in any desired format with any descriptive text desired. You specify a "picture" of exactly how the output will look and the output is printed in that "image".

The general form of the statement is as follows:

PRINT IN IMAGE string:list of variables or expressions

The format in which the data is to be printed is stored in a string. The string may be included in the PRINT IN IMAGE statement or stored in a variable, and that variable referred to in the PRINT IN IMAGE statement.

For example, the statement

```
10 PRINT IN IMAGE "THE ANSWER IS ↴
 %%%%%%.%          %%%%":A,B
```

is equivalent to the statements

```
10 C = "THE ANSWER IS %%%%%%.% ↴
        %%%%"
20 PRINT IN IMAGE C:A,B
```

NOTE: A Line Feed was used to continue the statement on the next line. The Line Feed will, however, be ignored when the statement is executed, and the specified image will be printed on a single line; therefore, the string used to specify the image should never be longer than 72 characters.

When the PRINT IN IMAGE statement is executed, all characters (except % and # and ") will be printed exactly as they appear in the image string with the first character in the string (following the quote marks) appearing as the first character on the output line. The symbols % and # are used to specify the field formats and can never be printed. The double quote marks are used to delimit the string and, therefore, may never appear within the string.

```
> 10 INPUT A ↵
> 20 S = "%%%%%  %%%%.%%  ########↓
     ##.######" ↵
> 30 PRINT IN IMAGE S:A,A,A,A ↵
> RUN ↵
? 6666.666 ↵
 6667  6666.67  .67E+04  66.7E+02

>
```

If fewer fields are specified than there are variables, the "image" will be repeated until all of the variable values have been output. Each time the "image" is repeated, a new line will be used.

Example

```
> PRINT IN IMAGE "%%%%%":123,456,789 ↵
   123
   456
   789
```

Field Format Specifications

The field formats specify how the values stored in the variables are to be printed. Using the symbols % or # you must specify the number of characters that you wish printed; and, in the case of numeric values, the form in which the number is to be printed; namely, integer, decimal, or exponential.

Integer Fields

To specify integer fields, a % must be typed for each digit of the variable that you wish to print. If there is any chance that the variable will be negative, an extra % must be specified for the minus sign. If the variable is not an integer, it will be rounded to an integer, dropping any decimal places when it is printed. A maximum of 11 significant integer digits will be printed. If the number contains more than 11 significant integer digits, 11 significant integer digits will be printed and the rest of the number filled with zeros. For example, the number 123456789123456 would be printed as 123456789120000. If the field specification is too large; that is, more %'s are specified than needed, the number will be right justified. If the field specification is too small, an error diagnostic will be given.

Decimal Fields

Decimal fields also use a % sign to specify the number of digits needed. With decimal output, however, a decimal point must be specified. For example, the numeric field specified by %%%%.%% will print up to four integer digits and two decimal digits. As with integer fields, if there is any chance that the number will be negative, an extra % sign must be specified for the minus sign. If the decimal begins with a decimal point such as .66, one extra % must be specified for the minus sign. If the number begins with a decimal point such as .66, one extra % must be specified (%.%%) since a leading zero (0.66) is always printed. If more %'s than necessary are specified in the integer part of the field specification, the number will be right justified (leading blanks will be printed). If the decimal part of the field specification is too small, the number will be rounded. If the integer part of the field specification is too small, an error diagnostic will be given.

Up to 11 significant digits will be printed. If more than 11 significant digits are specified, the output will be rounded to 11 significant digits and zeroes used to fill the field.

Exponential Fields (Scientific Notation)

An exponential field may be specified either with or without a decimal point. With scientific notation a # sign is used to specify each place needed. When a decimal point is not included in the field specification, a minimum of seven # signs must be specified. The minimum specification includes a place for:

1. The sign of the number.
2. The decimal point.
3. A minimum of one decimal digit.
4. The E.
5. A plus or negative sign to indicate the sign of the exponent.

6. and 7. Two places for the exponent.

If the exponential field is specified without a decimal point, the number printed always will begin with a decimal point; for example, .5E+06.

If the specification contains a decimal point, the number will be printed with the number of integers specified. In this case, a minimum of five # signs are required after the decimal point. Four of these are used for the exponential part (4,5,6, and 7 above), and one for the sign of the number, which in this form is included as the last # sign specified even though the sign of the number is printed at the beginning of the number.

In both forms up to 11 significant digits may be specified. If the specification indicates more than 11 significant digits, blanks will be supplied to fill these places. The number will be rounded to 11 significant digits and be right justified. If the field specification is too small, the decimal part of the number will be rounded if possible or an error message will be printed.

```
>100 R = 1.23497E-09 ↵
>200 PRINT IN IMAGE "############":R ↵
>300 PRINT IN IMAGE "###.########":R ↵
>RUN ↵
  .123497E-08
  123.497E-11

>
```

String Fields

Either a % or a # may be used to specify a string field. One character will be printed for each % or # sign specified. The entire string need not be specified; only the first part of the string can be printed.

```
>10 T = "BROWN W.W., 945 HUSTON" ↵
>20 PRINT IN IMAGE "%%%%%%%%%%":T ↵
>RUN ↵
BROWN W.W.
```

A string field is left justified; that is, if more % or # signs are specified than there are characters in the string, the blanks will be supplied at the end.

Standard Functions

The standard functions are commonly used relationships which are included in SUPER BASIC as a convenience for you. The standard functions are used by specifying the function name followed by the desired argument in parentheses.

Standard Function	Arithmetic Expression	SUPER BASIC Expression	Special Information
Mathematical Constant pi (π)	pi - π	PI	
Sine	sin(a)	SIN(A)	Argument must be in radians.
Cosine	cos(a)	COS(A)	Argument must be in radians.
Tangent	tan(a)	TAN(A)	Argument must be in radians.
Arctangent	arctan(a)	ATAN(A)	Angle whose tangent is A. The answer is in radians.
Natural Logarithm	ln(a)	LOG(A)	Argument must be positive.
Common Logarithm	$\log_{10}(a)$	LOG10(A)	Argument must be positive.
Exponential	e^a	EXP(A)	
Absolute Value	$\lvert a \rvert$	ABS(A)	
Square Root	$\sqrt{a}$	SQRT(A)	Argument must be positive.

Example

```
SQRT(25)
```

The argument of any standard function may be a numeric value, a predefined variable, or any numeric expression. The standard functions are treated as numeric expressions and may be used in replacement statements, PRINT statements, etc.

Example

```
10 INPUT A
20 R = SIN(A–PI)+5*COS(A)
30 PRINT R;TAN(R)
```

The table on the previous page includes the most commonly used SUPER BASIC functions. *NOTE: Not all of the functions built into SUPER BASIC are included here. For a complete list see the Tymshare SUPER BASIC Manual, Reference Series.*

DATA Statement, READ, And RESTORE Commands

Occasionally you may wish to store certain of your data values within the program. Data may be stored in the program in a "data block" which is created by using the indirect statement DATA. The data values stored in the "data block" may be accessed with the command READ together with the list of variables into which the values are to be read.

DATA Statement

A "data block" is created with indirect DATA statements (DATA statements must have a line number) which have the following general form:

line number DATA value,value,"string",value

Example

```
110 DATA 18.3,1E6,"EXP","STG",3
```

The data is stored in a "data block" in the order in which it appears in the DATA statement. If more than one DATA statement is used in a program, the statement with the lowest line number will be stored in the "data block" first. It does not matter where the DATA statements appear in your program; they may appear together at the beginning or end of the program, or they may be scattered throughout the program. Only numeric values and strings enclosed in quotes may appear in DATA statements. An expression or variable may not be used. A DATA statement may contain any number of values, separated by commas, as long as the entire statement is not longer than 256 characters. Any number of DATA statements may be included in the program.

READ Command

A "data block" is read with the command READ which may be used either directly or indirectly, and takes the following form:

READ variable list

Each time a READ command is executed, one value from the "data block" will be assigned to each variable in the variable list. If a second READ command is executed, the next previously unassigned value in the data block will be stored in the variable specified.

Example

```
10 READ A,B,C
20 DATA 1.59, "GAL"
30 READ D
40 DATA 2.57, –3E09
```

When executed A = 1.59, B = "GAL", C = 2.57, and D = –3E+09.

RESTORE Command

Normally, each value in the "data·block" will be read only once. If you wish to reread the "data block" you must use the command RESTORE. RESTORE may be used both directly and indirectly. RESTORE tells SUPER BASIC to return to the beginning of the "data block" and start rereading it. RESTORE may be used at any time during program execution; the entire block need not have been read; however, there is no way to start rereading the "data block" at any place other than at the beginning of the block.

Example

```
10 READ A,B,C
20 DATA 1,2,3,4,5
30 RESTORE
40 READ D,E,F,G
```

When executed, A=1, B=2, C=3, D=1, E=2, F=3, G=4.

NOTE: When the program is executed with the RUN command or a **direct** *GO TO, the data always will be read starting with the first value of the data block. However, an* **indirect** *GO TO statement within a program will read data starting with the first previously unassigned data value in the block.*

Example

```
10  PRINT "NAME BALANCE DUE"
20  FOR I = 1 TO 5
30  READ N,P,M,N$
40  B = P–M*N$
50  PRINT N; "$ ":B
60  NEXT I
70  DATA "JONES",21956,172.59,12
80  DATA "SMITH",350,21.95,16
90  DATA "MARCH",7650,99.78,5
100 DATA "ROBERTS",1800,50.51,9
110 DATA "EDWARDS",2100,171.00,2
```

Programming Hint: If you wish to reuse a program using varying sets of data, you might reserve one specified range of line numbers such as 100-299 for the program, and one for DATA statements such as 0-99. Your program could be saved on the disk and reused (see Section 3), and your data could be stored in DATA statements and read into SUPER BASIC from paper tape (see Section 5).

SAMPLE SUPER BASIC PROBLEMS

The following sample problems show problem definition, flowcharting, coding, and program execution.

DOUBLE DECLINING BALANCE DEPRECIATION

Define The Problem

The problem is to compute the double declining balance depreciation on any given asset over any specified number of years.

Input

1. Cost of the asset (C).
2. Estimated useful lifetime (L).

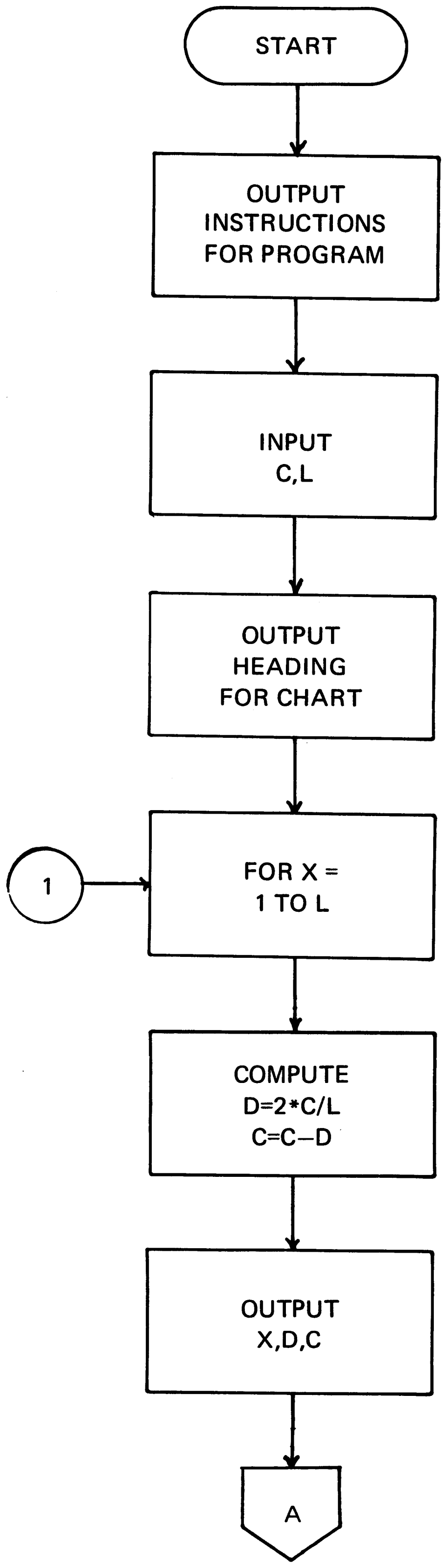
Compute

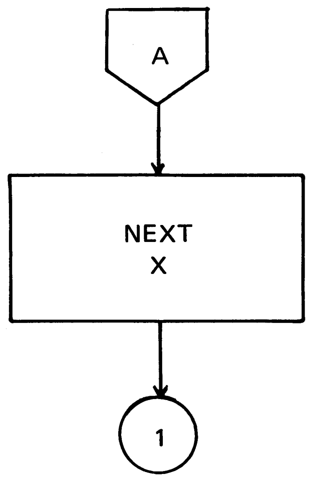
1. Depreciation $D = 2\frac{C}{L}$
2. Book value $C = C-D$

Output

For the entire range of years.
1. Year (X)
2. Amount of depreciation (D)
3. Book value (C)

Flowchart





SUPER BASIC Program And Sample Execution

```
– SBASIC ↵
> 10 PRINT "PROGRAM TO CALCULATE DOUBLE DECLINING BALANCE DEPRECIATION" ↵
> 20 PRINT             ! SKIPS ONE LINE ↵
> 30 PRINT "WHEN THE QUESTION MARK IS TYPED, INPUT THE COST OF THE" ↵
> 40 PRINT "ASSET IN DOLLARS AND ITS ESTIMATED USEFUL LIFETIME IN YEARS." ↵
> 45 INPUT C,L ↵
> 46 PRINT ↵
> 50 PRINT" ↴
YEAR            DEPRECIATION            BOOK VALUE" ! HEADING FOR CHART ↵
> 60 FOR X = 1 TO L ↵
> 80 D = 2*C/L ↵
> 90 C = C–D ↵
> 100 PRINT IN IMAGE" ↴
%%%             $ %%%%%%.%%           $ %%%%%.%%": X,D,C ↵
> 110 NEXT X ↵
> RUN ↵
PROGRAM TO CALCULATE DOUBLE DECLINING BALANCE DEPRECIATION

WHEN THE QUESTION MARK IS TYPED, INPUT THE COST OF THE
ASSET IN DOLLARS AND ITS ESTIMATED USEFUL LIFETIME IN YEARS.
? 100000,20 ↵

YEAR            DEPRECIATION            BOOK VALUE
  1             $    10000.00         $  90000.00
  2             $     9000.00         $  81000.00
  3             $     8100.00         $  72900.00
  4             $     7290.00         $  65610.00
  5             $     6561.00         $  59049.00
  6             $     5904.90         $  53144.10
  7             $     5314.41         $  47829.69
  8             $     4782.97         $  43046.72
```

```
   9          $        4304.67          $        38742.05
  10          $        3874.20          $        34867.84
  11          $        3486.78          $        31381.06
  12          $        3138.11          $        28242.95
  13          $        2824.30          $        25418.66
  14          $        2541.87          $        22876.79
  15          $        2287.68          $        20589.11
  16          $        2058.91          $        18530.20
  17          $        1853.02          $        16677.18
  18          $        1667.72          $        15009.46
  19          $        1500.95          $        13508.52
  20          $        1350.85          $        12157.67

>
```

MEAN AND STANDARD DEVIATION

Define The Problem

The problem here is to compute the mean and standard deviation of a group of data. The mean is computed using the following formula:

$$M = \frac{\sum_{i=1}^{N} X_i}{N}$$

The standard deviation is computed using the following formula.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (X_i - M)^2}{N-1}}$$

Input

1. The total number of data items (N).
2. The data (placed in the array A(I)).

Compute

1. Mean
2. Standard Deviation

Output

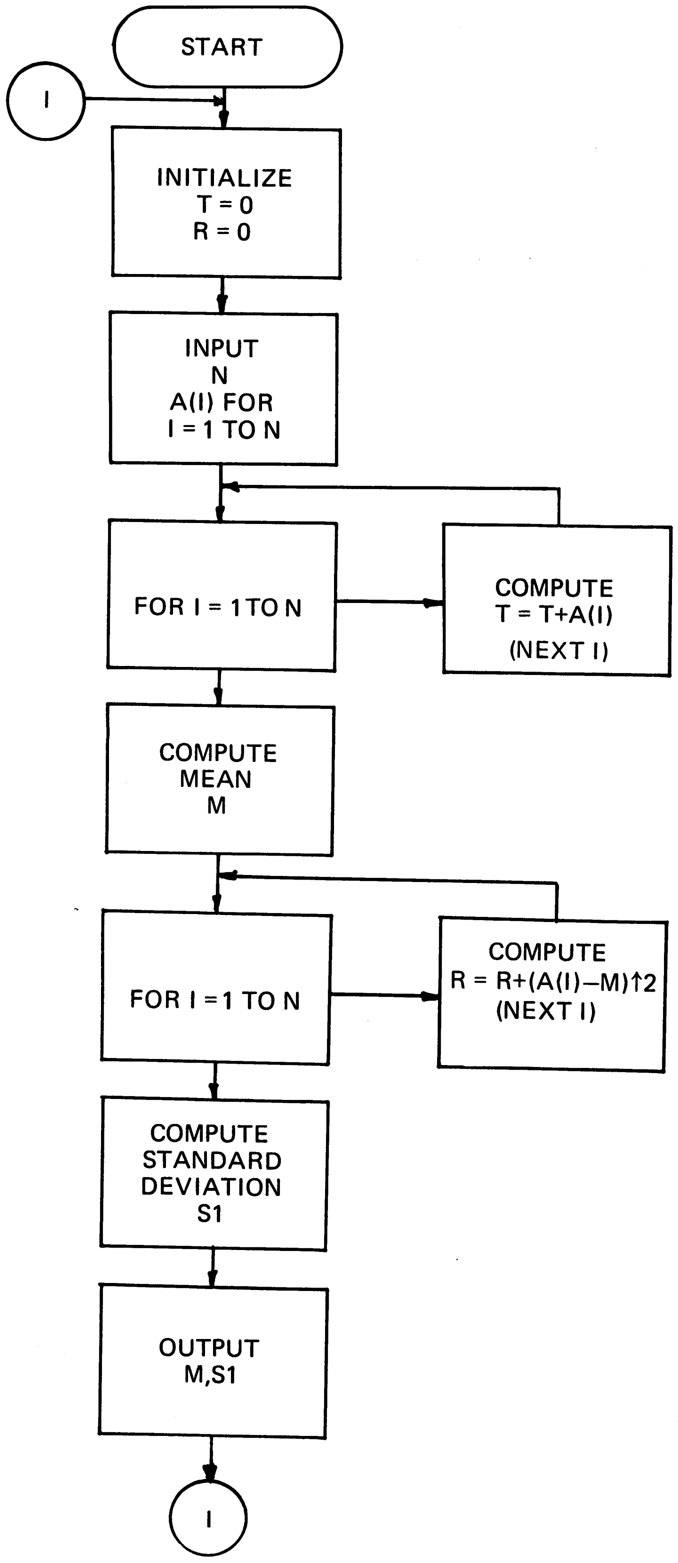
1. Mean (M)
2. Standard Deviation (S1)

Flowchart: *See next page.*

SUPER BASIC Program and Sample Execution

```
-SBASIC ↵
>10 T=0 ↵
>20 R=0 ↵
>25 PRINT "TOTAL PIECES OF DATA"; ↵
>30 INPUT N ↵
>32 DIM A(N) ↵
>35 PRINT "TYPE IN THE DATA SEPARATED BY COMMAS" ↵
>37 !VALUES SUPPLIED AND MEAN COMPUTED ↵
>40 FOR I=1 TO N ↵
>50 INPUT A(I) ↵
>60 T=T+A(I) ↵
>70 NEXT I ↵
>80 M=T/N ↵
>85 ! STANDARD DEVIATION COMPUTED ↵
>90 FOR I=1 TO N ↵
>100 R = R+(A(I)-M)↑2
>120 NEXT I ↵
>130 S1=SQRT(R/(N-1)) ↵
>140 PRINT "MEAN =";M,"STANDARD DEVIATION =";S1 ↵
>150 GO TO 10 ↵
>RUN ↵
TOTAL PIECES OF DATA? 5 ↵
TYPE IN THE DATA SEPARATED BY COMMAS
?  1,6,4,5,7 ↵
MEAN  =     4.6        STANDARD DEVIATION  =        2.3021729
TOTAL PIECES OF DATA?           ⊕⊕
INTERRUPTED IN STEP 30
>
```

Flowchart


START
I
INITIALIZE
T = 0
R = 0
INPUT
N
A(I) FOR
I = 1 TO N
FOR I = 1 TO N
COMPUTE
T = T+A(I)
(NEXT I)
COMPUTE
MEAN
M
FOR I = 1 TO N
COMPUTE
R = R+(A(I)–M)↑2
(NEXT I)
COMPUTE
STANDARD
DEVIATION
S1
OUTPUT
M,S1
I

STOCK PROGRAM

Define The Problem

Given the sales, cost of sales, number of shares, and price-to-earnings ratio, determine the gross income, the net income, earnings per share, and what the stock price should be.

Input: The sales, cost of sales, number of shares, and price earnings ratio: S,C,H,R.

Compute: The gross income, the net income (at 50% income tax), the earnings per share, and what the stock price should be: G,N,E,P.

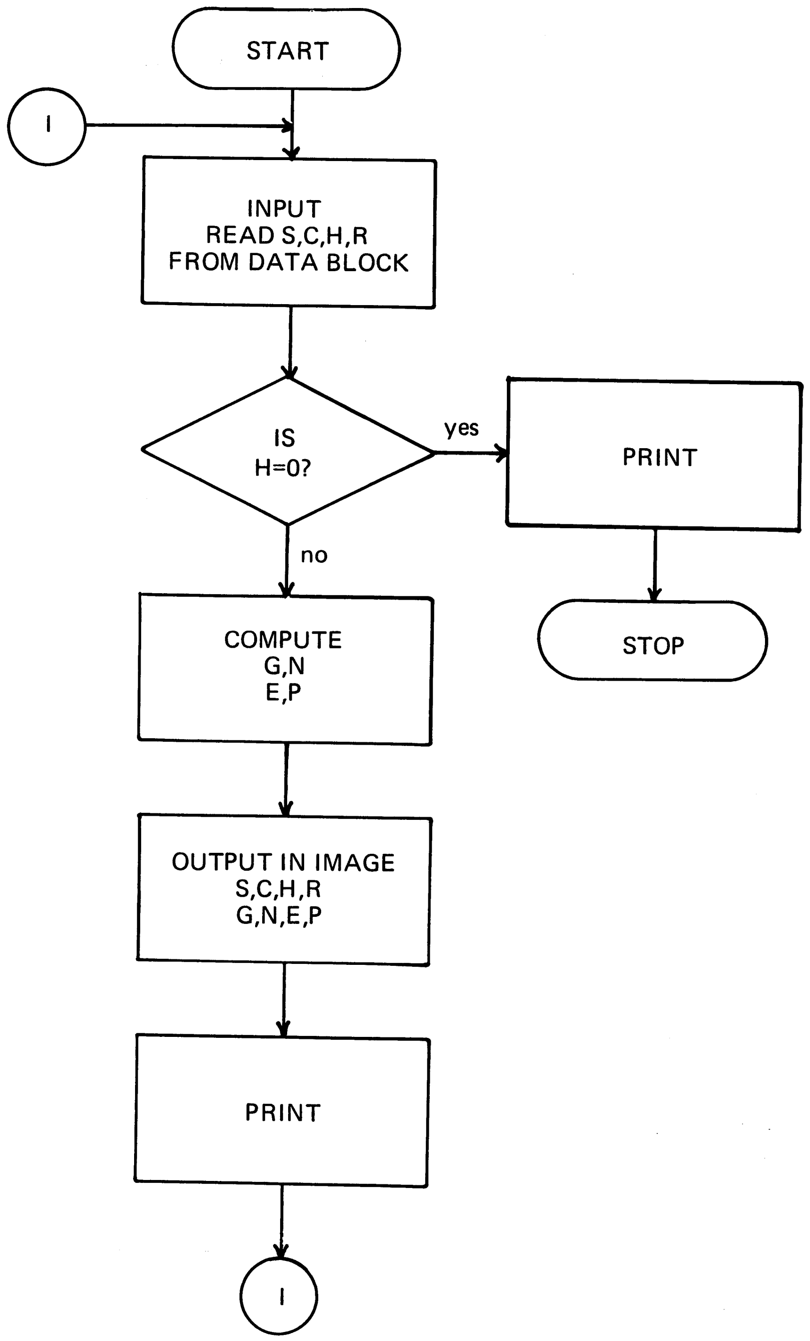
Output: The results.

Flowchart: *See next page.*

SUPER BASIC Program And Sample Execution

```
–SBASIC ↵
>100 READ S,C,H,R ↵
>110 GO TO 200 IF H=0 ↵
>120 G=S–C ↵
>130 N=.5*G ↵
>140 E=N/H ↵
>150 P=R*E ↵
>155 Q="S=  $ %%%%%%%%  C= $ %%%%%%%%  H= %%%%%%  R=     %%%%" ↵
>156 W="G=  $ %%%%%%%%  N= $ %%%%%%%%  E= $%%.%%   P= $%%%.%%" ↵
>160 PRINT IN IMAGE Q: S,C,H,R ↵
>170 PRINT IN IMAGE W: G,N,E,P ↵
>180 PRINT ↵
>190 GO TO 100 ↵
>200 PRINT ↵
>210 DATA 10000000,8000000,500000,15 ↵
>220 DATA 117110,93690,7730,72 ↵
>230 DATA 2189300,1641980,91220,20 ↵
>240 DATA 2448700,1910000,112230,43 ↵
>250 DATA 0,0,0,0 ↵
>RUN ↵
S=  $ 10000000  C= $   8000000  H= 500000  R=     15
G=  $  2000000  N= $   1000000  E= $ 2.00  P= $  30.00

S=  $   117110  C= $     93690  H=   7730  R=     72
G=  $    23420  N= $     11710  E= $ 1.51  P= $109.07

S=  $  2189300  C= $   1641980  H=  91220  R=     20
G=  $   547320  N= $    273660  E= $ 3.00  P= $  60.00

S=  $  2448700  C= $   1910000  H= 112230  R=     43
G=  $   538700  N= $    269350  E= $ 2.40  P= $103.20

>
```

Flowchart


START
I
INPUT
READ S,C,H,R
FROM DATA BLOCK
IS
H=0?
yes
PRINT
STOP
no
COMPUTE
G,N
E,P
OUTPUT IN IMAGE
S,C,H,R
G,N,E,P
PRINT
I

SECTION 3
PROGRAM FILES

Saving And Reusing A Program

A program written on the terminal in SUPER BASIC may be saved and reused at any time by storing it on a disk file.

To Save A Program

To save a program which has just been written, rewritten, or read in from paper tape, use the SAVE command as follows:

>SAVE /File Name/ ↵

NOTE: With SAVE, any single character or group of characters except commas may be used in a file name. It is suggested however that file names be short (1 - 4 characters).

SUPER BASIC will respond with either NEW FILE or OLD FILE as follows:

NEW FILE ↵

This message indicates that the file is a new one; that is, you do not already have a file by that name. Press the Carriage Return.

OLD FILE ↵ or **N ↵**

This message indicates that you are trying to write over (change) an old file. If you wish to change the old file, hit the Carriage Return to indicate that you realize the file is an old one. If you happened to pick a file name that is already in use, and you would like to save that file, type N ↵ and repeat the above procedure using a new file name.

NOTE: Direct commands and values stored in variables will not be saved on disk files created with the SUPER BASIC SAVE command.

To Reuse A Program

To reuse a program which has been saved on the disk, use the LOAD command. This command copies the file from the disk; the file is not erased.

To use the LOAD command with a saved program, type LOAD and the file name enclosed in slashes as follows:

>LOAD /File Name/ ↵

When the computer returns the >, the program is ready for use. Only statements with errors in them will be printed. If you would like a complete listing of the program use the command LIST when the > is returned. If you wish to start execution of the program, use the RUN command. If you wish to edit the program in any way, make the changes and then use the SAVE command to write the edited program back on the file. Remember that nothing done on the terminal will be saved unless it is saved on the file.

Using A Tymshare Library Program

>LOAD "Library File Name" ↵

Any library program which is written in SUPER BASIC may be called by using the LOAD command and enclosing the file name in double quote marks as shown above. *NOTE: Most SUPER BASIC library programs are self-starting.*

Removing A File

To remove a previously saved file from the disk, use the EXECUTIVE command DELETE (see Page 30).

SECTION 4
SUPER BASIC EDITING FEATURES

If you have made an error in your program, there are three courses of action open to you. You may:

1. Add a statement.
2. Delete a statement.
3. Change a statement.

Statements may be added at any time. Statements may be changed or deleted while they are being typed using a Control A (A^C) and/or a Control Q (Q^C); or immediately following the incorrect line by using the Control Z (Z^C) and/or the Control D (D^C).

The system is always in the edit mode both when a statement is being typed, and also immediately following the Carriage Return after an indirect statement has been typed. Statements also may be edited immediately after an error message or the error diagnostic ? is returned.

Example

```
>50 PRINT A;B;C ↵
>Z^CA50 PRINT A,D^CB;C ↵
>60 PRING R,S ↵
?
>Z^CN60 PRINT D^CR,S ↵
>
```

If you wish to delete or change a line other than the current or last line typed, you must use either the DELETE or EDIT command. *NOTE: Control characters are explained below.*

Adding A Statement

To add a statement to a SUPER BASIC program, merely type the statement with a line number that will position it at the appropriate place in the program. SUPER BASIC orders the program according to the statement line numbers ignoring entirely the order in which the statements are typed. If you cannot add a statement where you need one; for instance, if you need to add a statement between line number 10 and 11, you must renumber your program before you add the statement.

A program may be renumbered very easily with the command RENUMBER (abbreviated REN). You must specify the first new line number of the program, the first line to be renumbered, and the increment desired between the lines as follows:

RENUMBER AS new line # FROM old line # BY increment

The RENUMBER command always must be used directly (without a line number). When RENUMBER is used, the specified statements in the program will be renumbered. The program itself will not be changed, only the line numbers and references to line numbers will be changed.

If, for example, we wish to add a statement between 0 and 1 to print RADIUS =, the program first must be renumbered; then the statement may be added.

```
>0  PRINT "AREA OF A CIRCLE" ↵
>1  INPUT R ↵
>10 A = PI*R↑2 ↵
>11 PRINT "AREA =";A ↵
>12 GO TO 1 ↵
>RENUMBER AS 20 FROM 0 BY 10 ↵
>LIST ↵
20 PRINT "AREA OF A CIRCLE"
30 INPUT R
40 A = PI*R↑2
50 PRINT "AREA =";A
60 GO TO 30
>25 PRINT "RADIUS=": ↵
```

Deleting A Statement

The easiest way to delete a statement is to type the statement line number followed immediately by a Carriage Return. The statement and the line number will be removed from the program.

Statements also may be deleted with the direct command DELETE (which may be abbreviated DEL) followed by the line numbers of the statements to be deleted separated by commas. For example, the statement

```
> DELETE 10,20,11,9,5
```

will delete statements 5, 9, 10, 11, and 20.

A range of statements may be deleted by placing a dash (–) between the line numbers. When a dash is used, all statements with line numbers within the range specified will be deleted from the program.

For example, the statement

```
> DELETE 0,20–40,99
```

will delete statement 0, statements 20 through 40 inclusive, and statement 99.

To delete everything that you have done (the program, the values stored in variables, etc.), use the command DELETE ALL. Whenever you start a new program it is a good idea to use DELETE ALL after saving your program, otherwise you may pick up statements or variable values from the old program.

Changing A Statement

The simplest way to change a statement is to retype it correctly using the same line number. However, a program may be changed more quickly using the EDIT command in conjunction with the editing control characters. The EDIT command always must be used directly with a single line number. When the command is executed, the system types the statement specified and then waits for you to edit it. For example, if statement 10 were PRINT A;B the editing might proceed as follows:

```
>EDIT 10↵
10 PRINT A;B          Statement typed.
$Z^C$A10 PRINT A↵     Statement edited.
```

Statement 10 is now PRINT A. Notice how the control characters make editing easier.

Editing With Control Characters

The SUPER BASIC editing control characters provide the most flexible means of editing available. To generate a control character you hold the CTRL key down and press the character you desire. Four of the control characters used in editing will be introduced in this manual. The control characters may be used when a statement is first typed as well as when the EDIT command is used. *NOTE: Control characters are indicated as follows: A^C, D^C, etc. Some control characters print a symbol, others print nothing.*

Deleting A Character (Control A)

This control character deletes the preceding character. A^C can be used repeatedly to delete several preceding characters. Whenever an A^C is used, a ← is printed.

Deleting A Line (Control Q)

A Control Q deletes the entire statement currently being typed and allows you to retype the entire statement from the beginning. When a Q^C is used, an ↑ is printed.

Copying To A Character (Control Z)

A Control Z searches either the preceding line or the statement specified by an EDIT command for the character specified after the Z^C. If Z^C cannot find the character, Z^C does nothing to the line, rings the bell, and waits. If the character is found, Z^C copies all of the characters up to and including the character specified and prints the characters on the terminal. Now make your corrections and continue typing the line. When you want to end the statement, hit the Carriage Return.

Copying The Rest Of A Line (Control D)

The Control D copies the rest of the old line to the new line (including the Carriage Return). The new line is checked for errors and control is returned to you with a >.

Listing A Statement

To list a statement (have it typed out), use the command LIST with the line number of the statement. If you wish to list a group of statements, simply separate the line numbers with commas; for example, LIST 1,3,7. If you wish to list a range of statements, use a dash between the limits of the range such as LIST 10-20. If you wish to list the entire program, use the command LIST followed by a Carriage Return. For example, the statement

```
> LIST 0,10–40,97
```

will type the statements with line numbers 0,10 through 40 inclusive, and 97.

Example Using EDIT, DELETE, LIST And The Control Characters

```
>10 INPUT A,B↵
>20 T = B↑2–SA$^C$←A↵
>LIST 20↵
20 T = B↑2–A
>20 S = T+AQ$^C$↑
25 S = T+A↑2↵
>30 I = "ANSWRA$^C$←ER = "↵
>40 PRINT S↵
>50 PRINT A,B↵
>LIST↵
10 INPUT A,B
20 T = B↑2–A
25 S = T+A↑2
30 I = "ANSWER = "
40 PRINT S
50 PRINT A,B
>EDIT 50↵
50 PRINT A,B
$Z^C$A50 PRINT A;B,I:S↵
>DELETE 40↵
>LIST↵
10 INPUT A,B
20 T = B↑2–A
25 S = T+A↑2
30 I = "ANSWER = "
50 PRINT A;B,I:S
>
```

SECTION 5
PAPER TAPE

On line time may be minimized by punching your program on paper tape off line. You may then log in, call SUPER BASIC, and, using the SUPER BASIC command TAPE, read the entire tape directly into SUPER BASIC.

Punching A Paper Tape Off Line

To punch a paper tape off line, turn the knob on the front of the terminal to LOCAL and push the ON button on the paper tape punch. All characters typed on the keyboard will be punched on the paper tape as well as printed on the terminal. The program statements are punched on paper tape in the same manner as they are typed on line except that you must terminate each statement with a Carriage Return and then a Line Feed. If a statement is longer than one physical line, reverse this process; end the line to be continued with a Line Feed and then type a Carriage Return.

Errors on the paper tape may be corrected immediately after they are made in one of two ways.

1. You may correct an error off line by first punching the Back Space key on the paper tape punch controls and then hitting the RUB OUT key on the terminal; or,
2. You may use the two editing characters available. The back arrow (←) may be used to delete the previous character, and the up arrow (↑) immediately followed by a Carriage Return may be used to delete the entire line being typed.

CAUTION: When punching paper tape off line, always type both a Carriage Return and a Line Feed in the order indicated in the examples below.

Example

Turn the paper tape punch on.

```
10 INPUT A ↩↴
20 X = A↑2–8 ↩↴
30 PRINT IN IMAGE" ↴↩
A↑2–8 = ###########":X ↩↴
```

Turn the paper tape punch off.

Reading The Program Into SUPER BASIC

The program may be loaded directly into SUPER BASIC using the direct command TAPE as follows:

```
– SBASIC ↩
> TAPE ↩
```

Place the paper tape on the tape reader and turn the reader to START.

```
10 INPUT A
20 X = A↑2–8
30 PRINT IN IMAGE"
A↑2–8 = ###########":X
```

D^C *The D^C may be punched at the end of the paper tape or may be typed from the terminal after the entire tape has been read.*

The Control D terminates the TAPE command and control is returned to you. All statements with errors will be printed with the appropriate diagnostics after the Control D has been typed. You should now retype any incorrect statements as incorrect statements are not retained in the program. After the program has been read into SUPER BASIC and the errors corrected, you should continue as though you had just written the program on the terminal.

SECTION 6
THE EXECUTIVE

Entering The System

The process of calling the system and telling the computer who you are is called logging in.

After you make the connection properly, the system replies with:

PLEASE LOG IN: ↵

Now type a Carriage Return and the system replies with:

ACCOUNT: A3 ↵

Type your account number followed by a Carriage Return. The system then types:

PASSWORD: ↵

Type your password followed by a Carriage Return. *NOTE: The password does not print.* The system next types:

USER NAME: JONES ↵

Type your user name followed by a Carriage Return. The system next asks for a project code.

PROJ CODE: K-123-X ↵

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
–
```

Now call any of the subsystems such as SUPER BASIC, or give commands in the EXECUTIVE.

Calling SUPER BASIC

SUPER BASIC may be called at any time from the EXECUTIVE (whenever the EXECUTIVE dash appears). To call SUPER BASIC, type SBASIC and a Carriage Return. The system will respond with a > to indicate that it is waiting for a SUPER BASIC statement. You may then give any of the SUPER BASIC commands.

```
– SBASIC ↵
> Any SUPER BASIC command may now be typed.
```

Continuing After Interruption

– CONTINUE ↵

If for some reason you return to the EXECUTIVE and then wish to return to your SUPER BASIC program, you can continue your program where you left off by typing CONTINUE or CON. The system will recall SUPER BASIC and give you a > to let you know that you are back in SUPER BASIC and have control.

All of the work that you had done before you returned to the EXECUTIVE will be saved as long as you have not called another language (CAL, EDITOR, etc.). CONTINUE applies to the last language called.

```
– SBASIC ↵
> INPUT A ↵
? 79 ↵
> QUIT ↵
– CONTINUE ↵
SBASIC
> PRINT A ↵
 79
>
```

The value stored in the variable A was saved by using CONTINUE. If you had reentered SUPER BASIC by typing –SBASIC, you would have lost everything that had been done up to that point that had not been stored on a disk file.

Listing Files

– FILES

When this command is typed in the EXECUTIVE a complete listing of all your current files will be printed, and the type of file will be indicated (SYM for symbolic, BIN for binary, or DUM for dump). Almost all SUPER BASIC files will be symbolic.

```
– FILES ↵
SYM    /MORTGAGE/
SYM    /MATRIX/
SYM    /MILE/
SYM    /GAS/
SYM    /TRO/
SYM    /SIM/
SYM    /ST/
SYM    /DEPR/
```

Deleting Files

– DELETE /File Name/

This command is used to delete (erase) files from the disk. More than one file may be deleted at a time by placing a comma between the file names as follows:

– DELETE /MORTGAGE/,/TRI/,/JR/ ↵

Leaving The System

When you are ready to leave the system, you must first be in the EXECUTIVE, characterized by the dash in the left hand margin. *NOTE: To return to the EXECUTIVE, type QUIT* ↵. Then type:

– LOGOUT (or –LOG) ↵

followed by a Carriage Return. The computer then will type:

TIME USED 0:37:12
PLEASE LOG IN:

Now disconnect the line, or let another user log in.

GLOSSARY

ARGUMENT
A variable upon whose value the value of a function depends. The arguments of a function are listed in parentheses after the function name whenever a function is used; for example, SIN(A).

CODE
A system of symbols for representing data or instructions (commands) in a computer. The act of translating or writing information in the form of abbreviations and specific notation which can be understood by the computer is usually called coding.

COMMAND
The part of a statement which specifies the type of operation to be performed by the computer. For instance, PRINT indicates an output operation.

COMPILE
To convert the program from a non-machine language such as SUPER BASIC to a machine language. SUPER BASIC checks each statement for errors before compiling and will return the appropriate diagnostic if the syntax is incorrect.

COMPUTE
Perform the operation(s) indicated.

DISK
A magnetic storage device on which data, programs, and text may be stored.

EXECUTE
To carry out an instruction or command, or set of commands (program).

FILE
A program or data which has been stored on the disk. Files are referred to by file names. Each file name must be unique.

Binary File
A file in which the machine instructions or data are stored in numbers to the base two. Binary files can be read only by a computer program and cannot be displayed in readable form on the terminal.

Dump File
A special binary file that contains a program to restore the operating system under which the program was run.

Symbolic File
A file which is stored using the image of the characters as they appear on the terminal. Most files used in SUPER BASIC are symbolic files.

FORMAT
A defined arrangement of characters, fields, lines, punctuation, headings for a desired clear presentation of data or printout.

HARDWARE
The mechanical, magnetic, and electrical components of a computer.

INPUT
Information or data supplied to the computer by the user. Input ordinarily is supplied from the terminal keyboard or from paper tape when using the Tymshare system.

LOOP
The repeated execution of an instruction or a series of instructions.

OFF LINE
Operations performed while not connected with the computer (such as punching paper tape).

ON LINE
Operations performed while connected directly with the computer.

OPERATOR
The what-to-do part of an arithmetic expression (for example, add, subtract). A symbol denoting the mathematical or logical operation to be performed.

OUTPUT
Information or data printed by the computer. Computer results such as answers to mathematical problems, statistical or accounting figures. Information or data written on tape also is considered output.

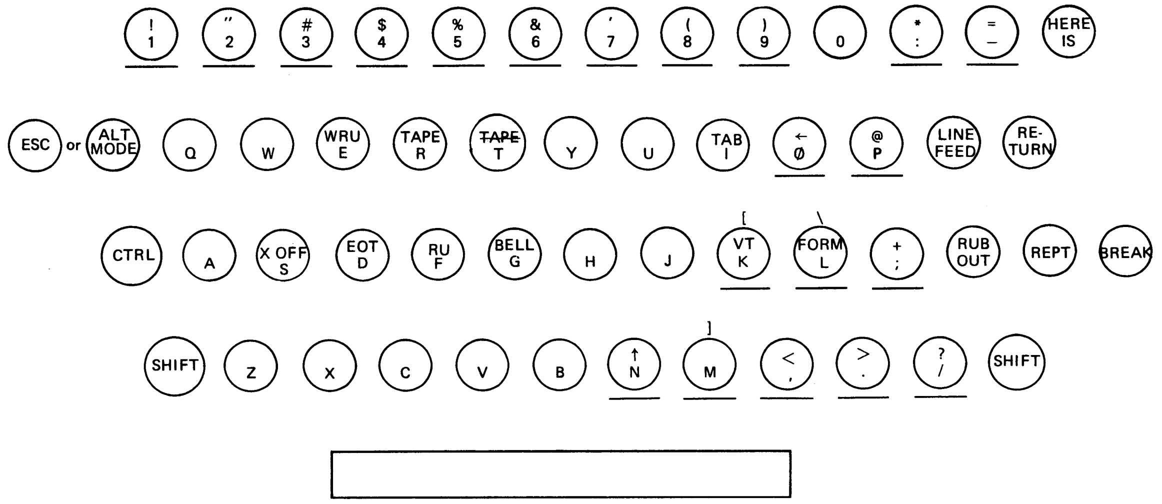
PUBLIC LIBRARY
A library of programs stored on disk files which may be used by any Tymshare user.

SOFTWARE
The features in a computer which are not hardware (the programs, compilers, languages, etc.).

STATEMENT
A single program step which consists of a command and any modifiers.

STORAGE
A general term for any device able to retain information.

APPENDIX
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.

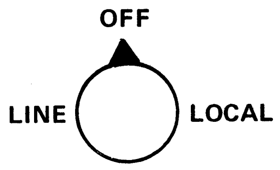
BREAK
DO NOT press this key; it causes a transmission interrupt and possible loss of program and data.

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.

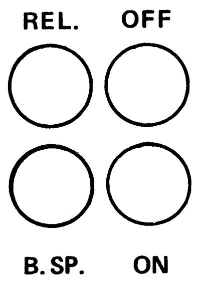
REL.
Releases the paper tape so that the user can pull it through manually.

OFF
Turns the punch OFF.

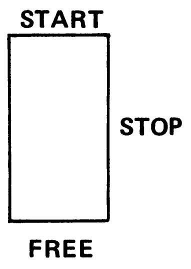
ON
Turns the punch ON to punch the paper tape.

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.

SUPER BASIC SUMMARY

NUMBERS

Integer (without decimal point)	e.g., 357940
Decimal (with decimal point)	e.g., 35.7940

Scientific Notation 3.57E23 (where E23 means x10^{23})

VARIABLES

Legal Variables	A-Z,A\$-Z\$,A0-Z9
Subscripted Variables	A(1),A(2),Z(L,M,N), A\$(I)-Z\$(I)

ARITHMETIC OPERATORS

In order of precedence

↑	Exponentiation
–	Unary minus
*,/	Multiplication, Division
+,–	Addition, Subtraction

RELATIONAL AND LOGICAL OPERATORS

=	Equal
#	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
AND	
OR	

MATHEMATICAL FUNCTIONS

ABS(A)	Absolute value of A
SIN(A)	Sine of A
COS(A)	Cosine of A
TAN(A)	Tangent of A
ATAN(A)	Angle whose tangent is A
EXP(A)	e to the power A
LOG(A)	Natural logarithm of A
LOG10(A)	Base 10 logarithm of A
SQRT(A)	Positive square root of A
PI	Mathematical constant π

STATEMENT FORMS

Unless indicated, the following statements may be used both directly (without a line number), and indirectly (with a line number).

Input/Output Statements

INPUT variable list
READ variable list (reads data from DATA statements)
PRINT variable, expression, "string"
PRINT IN IMAGE "image string":variable, expression, "string"

Replacement Statement

Single variable = arithmetic or string expression
A = PI*R↑2 S = "STRING1"

Control Statements

GO TO line number
RUN (direct only)
QUIT
RESTORE (used with DATA statements)

FOR Loop Statements (Indirect Only)

FOR variable = value list
FOR variable = limit BY interval TO limit
FOR variable = limit BY interval UNTIL terminating condition
NEXT variable

Miscellaneous

DIM array name (first sub_1 :last sub_1 ,first sub_2 : last sub_2)
DATA value list (indirect only)
! Comments
LIST (direct only)
TAPE (direct only)

Modifiers

IF expression

FILES (Direct Only)

SAVE /file name/
LOAD /file name/
LOAD "library program"

EDITING COMMANDS (Direct Only)

DELETE line number
DEL line number
DELETE ALL
EDIT line number
RENUMBER AS new line number FROM old line number BY increment

Control Characters

A^C	*Prints a ← and deletes preceding character.*
Q^C	*Prints an ↑ and deletes the statement being typed.*
Z^C Character	*Copies up to and including the character typed after it.*
D^C	*Copies and prints out the rest of the line. Ends the edit.*
W^C	*Prints a \ and deletes preceding word.*