

HP 3000

BASIC FOR

BEGINNERS

HEWLETT  PACKARD

3000 computer systems

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP 3000

BASIC FOR

BEGINNERS



List of Effective Pages

Pages	Effective Date
Title	Nov. 1972
Copyright	Nov. 1972
iii to ix	Nov. 1972
1-1 to 1-10	Nov. 1972
2-1 to 2-9	Nov. 1972
3-1 to 3-8	Nov. 1972
4-1 to 4-8	Nov. 1972
5-1 to 5-13	Nov. 1972
S-1	Nov. 1972
A-1	Nov. 1972

Printing History

Part No.	Date	Update Package	Date
03000-90025	Nov. 1972		

About This Manual

HP 3000 BASIC for Beginners is written for those people who do not know how to program a computer but need (or desire) to learn how to program. Hewlett-Packard's BASIC/3000 programming language is the vehicle used.

The reader should use this manual while he is sitting at a keyboard terminal attached to the HP 3000 Computer System, as the examples in the manual are designed to be used as they are discussed.

An introduction to this manual covers the minimum features of the computer that the reader must learn. This is followed by five lessons, each describing some functional elements of BASIC/3000.

- Calculating Values
- Transferring Control
- Looping
- Functions and Subroutines
- Strings and Files

At the end of each lesson is a quick-reference summary. At the end of the manual are a summary description of additional BASIC/3000 features and an appendix which explains how to use BASIC/3000 without a terminal.

Since this manual covers only a portion of the BASIC/3000 programming language, the reader should refer to *HP 3000 BASIC Interpreter (03000-90008)* for additional information and a complete description of BASIC/3000.

CONTENTS

ABOUT THIS MANUAL	iii
INTRODUCTION	vi
PROGRAMMING SYSTEMS/LANGUAGES	vi
LOGGING ON	vi
LESSON I CALCULATING VALUES	1-1
ARITHMETIC WITH HEADINGS	1-2
FORMULAS	1-4
CORRECTIONS	1-6
DOCUMENTATION	1-6
INTERACTION	1-6
SCIENTIFIC NOTATION	1-8
MIN AND MAX	1-8
LESSON I SUMMARY	1-10
LESSON II TRANSFERRING CONTROL	2-1
GOTO STATEMENT	2-2
IF STATEMENT	2-4
END OF CONTROL	2-8
LESSON II SUMMARY	2-9

LESSON III	LOOPING	3-1	
	FOR STATEMENT	3-2	
	MATRICES	3-2	
	NESTING	3-4	
	INPUT/OUTPUT LOOPS	3-6	
	LESSON III SUMMARY	3-8	
LESSON IV	FUNCTIONS AND SUBROUTINES	4-1	
	PREDEFINED FUNCTIONS	4-2	
	SUBROUTINES	4-4	
	USER-DEFINED FUNCTIONS	4-6	
	LESSON IV SUMMARY	4-8	
LESSON V	STRINGS AND FILES.	5-1	
	STRING VARIABLES	5-2	
	STRING ARRAYS	5-6	
	FILES	5-8	
	FILE COMMANDS	5-12	
	LESSON V SUMMARY	5-13	
SUMMATION	S-1		
APPENDIX	HOW TO USE BASIC/3000 WITHOUT A TERMINAL	A-1	

INTRODUCTION

Before you start using the examples in this manual, you should

- a. Read this introduction.
- b. Be seated at a keyboard terminal which has access to BASIC/3000 through the Hewlett-Packard Multiprogramming Executive Operating System (MPE/3000).
- c. Know how to turn the keyboard terminal on and off, and use its keyboard.

PROGRAMMING SYSTEMS/LANGUAGES

Programming is the act of telling a computer what to do in a language that it can understand. Programming languages are artificial languages created to express computer programs; they have grammatical rules, like natural languages, but the rules are simpler and stricter than the rules of natural language. In English you often can be understood even though you communicate vaguely. In a programming language, such as BASIC/3000, you must say exactly what you mean; no more and no less.

Before you can program in BASIC/3000, you must identify yourself to the computer and prove that you are authorized to use it. Then you must request access to the BASIC/3000 Interpreter program. When you are done, you notify the computer. Each of these operations is done with a command typed into the computer through a keyboard terminal.

LOGGING ON

You must first have access to a terminal which is connected to an HP 3000 computer. When the computer is connected (and ready) it prints a colon (:) at the beginning of a line and you type in a HELLO command. BOBG and STUDENT are the user and account identification codes, respectively; these two names tell the computer to whose bookkeeping account the session is charged. You must next supply the passwords for the particular user and account identification codes.

:HELLO BOBG.STUDENT
USER PASSWORD?

MMMMMMMMMM
ACCT PASSWORD?

MMMMMMMMMM
SESSION NUMBER = #S172

THU, MARCH 29, 1973, 8:50 AM
WELCOME TO THE TIME-SHARING SYSTEM

See the man in charge of the computer for your names and passwords. These are only examples. Note that the computer types a mask, over which you are to type your passwords. The computer's operating systems, MPE/3000, types out these masks so other people will not use your passwords.

The BASIC/3000 Interpreter is a program controlled by the computer; it allows the user to write, test, run, and change programs written in the programming language BASIC/3000. This interpreter is entered with a :BASIC command. The BASIC/3000 Interpreter signals that it is in control of the terminal (and ready) by typing a > (greater than) character at the beginning of a line; this is the BASIC prompt character (as opposed to : (colon) and is printed at the beginning of each line. To leave the Interpreter after completing work with BASIC/3000, use the EXIT command.

When you finish a session, you tell the computer you are done by typing BYE. The computer reports how long the session took and how much of the computer's actual processing time was consumed. This command is given after returning the computer from within the BASIC/3000 Interpreter. The computer must print a colon (:) at the beginning of a new line before you can type BYE. If it is still typing greater than (>), type EXIT and wait for a colon to appear.

:BASIC

>

>EXIT

END OF PROGRAM

:

:BYE

THU, MARCH 29, 1973 9:56 AM
END OF SESSION



LESSON 1

Calculating Values

At this point, you should have connected your terminal to the HP 3000 Computer System, have been accepted by the computer, and have entered into the BASIC/3000 Interpreter. The Interpreter is ready to work when it types out a greater than sign (>).

This lesson covers how to calculate the result of any arithmetic formula which uses addition, subtraction, multiplication, division, raising to a power, and finding the minimum or maximum of two values. In addition, you will learn how to print out your answer on the terminal, how to correct errors, and how to enter numbers from the terminal into the computer.

PRINT
RUN
LIST
SCRATCH
LET
REM
INPUT
MIN, MAX
+, -, *, /



ARITHMETIC WITH HEADINGS

Type in this statement.

You now have a very simple, one-statement program. The first part of the statement is always the statement number. The second part always indicates what general type of action the statement is to perform (PRINT in this case). PRINT means to print what follows on the terminal. After the statement type comes the data; in this case, the value to be printed ($35 + 5$).

Type the RUN command and observe what happens. The operation of the program statement (printing) is done when the program is run, not when it is entered. RUN, however, is a command (since it has no statement number) and is executed immediately.

The computer prints $>$ when this 1 line program is done.

Now type the PRINT statement, but with quote marks around the $35 + 5$ (i.e., "35 + 5"). Note that the result is $35 + 5$ because the quote marks indicated a literal string (which can contain any characters). When you typed in the new statement 100 this automatically erased and replaced the old statement 100.

Try this statement to see the difference between a numeric operation and a literal string. PRINT has two items to print in this example. Since they are separated by a semi-colon, they are printed close together when the program is RUN.

Programs usually have more than one statement. The statements are executed in order, from lowest number to highest number. This two-statement program shows several new capabilities: Three new operators ($-$ for subtraction, $*$ for multiply, and $/$ for divide) and commas, to separate print items and to specify printing them farther apart in fixed column positions.

Another feature of BASIC/3000 is that blank spaces are ignored (except, of course, within literal strings). Thus, the three statements of this program are equivalent, since the spaces are only superficial differences.

If we list the current program (with the LIST command), we see that all three statements are listed identically.

The SCRATCH command erases the current working program and leaves a clean slate. If you now type LIST, you will see there is nothing to list. SCR is a shorter form of SCRATCH.

```
>100 PRINT 35+5
```

```
>RUN  
40
```

```
>
```

```
>100 PRINT "35+5"
```

```
>RUN  
35+5
```

```
>
```

```
>100 PRINT "35+5=";35+5
```

```
>RUN  
35+5= 40
```

```
>
```

```
>100 PRINT "35+5","35-5","35*5","35/5"
```

```
>200 PRINT 35+5,35-5,35*5,35/5
```

```
>RUN
```

35+5	35-5	35*5	35/5
40	30	175	7

```
>
```

```
>100 PRINT 35+5
```

```
>200PRINT35+5
```

```
>300 P R I N T 3 5 + 5
```

```
>RUN
```

```
40
```

```
40
```

```
40
```

```
>
```

```
>LIST
```

```
100 PRINT 35+5
```

```
200 PRINT 35+5
```

```
300 PRINT 35+5
```

```
>
```

```
>SCRATCH
```

```
>LIST
```

```
>
```

FORMULAS

The PRINT statement can print the result of any formula (combination of operations). The calculations in a formula are performed from left to right, except that multiply and divide are done before add and subtract. In this case, the result is 7 since the multiply is done first.

Parentheses are used to override the priority of operations; operators within parentheses are done first. Thus, the result is 9, since the add is done before the multiply.

What are the results of each of these statements?

What is A? When we run the program, we find an undefined value because A is a variable (a container which holds one number at a time) and we have not given A any value.

The LET statement puts numbers into variables. It consists of the word LET, a variable name, an equals sign (=), and a formula. The result of the formula is assigned to the variable and remains there until it is changed. (The word LET is optional.)

A formula can use many variables; the name for each variable is any capital letter of the Alphabet (A to Z) or any letter followed by a digit (e.g., A1, B9, Q7, S0, Z4, etc.). Each such name identifies one unique variable.

The value contained by a variable is accessed by using the variable name in a formula.

Here is a short program to calculate the area (A) of a circle. R is used for radius and P is used for π . ($\text{Area} = \pi(R^2) = P * R * R$).

Another way to do this is with the exponentiate operator (**). This operator raises the first value to the power of the second value. Exponentiate is done before multiply and divide.

>100 PRINT 1+2*3

>RUN

7

>

>100 PRINT (1+2)*3

>RUN

9

>

>SCR

>10 PRINT 5+5-2*3

>20 PRINT 5+(5-2)*3

>30 PRINT (5+5)-2*3

>40 PRINT 5+((5-2)*3)

>RUN

>SCRATCH

>100 PRINT A

>RUN

UNDEFINED VALUE IN LINE 100

>

>100 LET A=35+5

>200 PRINT A

>RUN

40

>

>SCR

>100 A=5

>200 LET B=A*20

>300 PRINT (A+2)*(B-2)

>RUN

>SCR

>100 R=10

>200 P=3.14

>300 A=P*R*R

>400 PRINT A

>RUN

314

>

>300 A=P*R**2

>RUN

314

>

CORRECTIONS

You have already seen how to make changes by retyping statements and how to erase the entire current program with a `SCRATCH` command. Now you will see what happens when you make an error in a statement.

Type in this invalid statement. The computer types back `ERROR` to indicate the statement is wrong. Following `ERROR` is an `@` symbol and a number. This means the error occurred at about the n th non-blank character after the line number. If you can see at a glance what the error is, simply type a *return*.

Retype the invalid statement. Assume now that you don't see what is wrong. The computer will provide you with more information about the error if you type any character.

If you see your error as soon as you make it, you can delete it by typing *Control H*. The computer responds with `"\"`. You can delete the last n characters by typing *Control H, n* times. Then you continue with the rest of the line. (Hold down `CONTROL` while striking `H`.)

What if you discover the error, but want to start over instead of correct the line? Typing a *Control X* cancels the current line. The computer responds with `"!!!"` and return/linefeed.

DOCUMENTATION

The `REM` (remark) statement allows you to provide comments in your program. Any message desired can appear after the word `REM`. This statement has no effect on the functioning of the program.

INTERACTION

So far you have worked only with fixed data that is internal to the program. Now we introduce a method that allows the program to interact with the person at the terminal so values for variables can change each time a program is run.

`INPUT` is a statement that reads one or more data items entered from the terminal and assigns them to variables. When the program is run, the computer prints out a question mark (?) and waits for the user to input his data.

A useful option with `INPUT` is the ability to print a string before inputting the items (a question mark is not printed). More than one item can be listed in an `INPUT` statement, separated by commas; you type the values, separating them with commas. If you do not type in enough values, the computer prompts with `??` to indicate more data is required. Write a program to input five numbers and print them out in even columns. Then run the program to practice inputting data.

>SCRATCH

>100 PRINT A
ERROR#3
>

>100 PRINT A
ERROR#3:
ILLEGAL STATEMENT TYPE
>

>100 PRINT\NT A
>LIST
100 PRINT A
>

>100 PRINT B!!!
LIST
100 PRINT A
>

>100 REM THIS IS A REMARK.
>200 REM 756URETHTHSHWG-:326X<>?
>300 REM ANYTHING GOES IN MY REM
>RUN

>

>SCR
>100 INPUT R
>200 P=3.14
>300 A=P*R**2
>400 PRINT "AREA IS: ";A
>RUN
?20
AREA IS: 1256

>
>100 INPUT "ENTER RADIUS:",R
>RUN
ENTER RADIUS:10
AREA IS: 314

>SR\CRATCH
>

SCIENTIFIC NOTATION

When a number is printed out (or entered) it is rounded to 6 (or 7) digits of accuracy because that is all of the accuracy the computer can store.

Although the computer won't allow more than 7 digits of accuracy, it will allow numbers with more than 7 digits to the left or right of the decimal point. That is, numbers can be very large (up to 77 places) but only 7 of those digits are significant. When a number too large is printed, it is scaled in a special way to avoid printing many zero digits. The format is a number (with only one digit to the left of the decimal place) followed by the letter E and an integer specifying the power of ten to multiply times the number. For example, 1.23457E+9 means $1.23457 \times 10^9 = 1234570000$.

For between zero and one, the exponent is negative. $1.23457E-07 = .0000001234567$. This notation allows a very wide range of numbers to be entered or printed in a compact form.

On input, the number before the E can be an integer, a decimal point followed by an integer, or an integer — decimal point — integer. Using this simple program, input the following numbers and observe the printed results.

1234560000.
1.0E-20
-1.0E20
1E2
23E-3
.001E3
1.1E5

Now try to find the largest possible number. (The computer will tell you when your number gets too big.)

MIN AND MAX

MIN and MAX are two more operators that can be used in formulas. In the priority of operations, MAX and MIN come below add and subtract.

MIN finds the minimum of two values and MAX finds the maximum of two values.

What are the results printed in lines 50, 60, and 70 of this program? Test your answers.

```
>100 PRINT 12345.6789123
>RUN
12345.7
```

```
>100 PRINT 1234567890.00
>RUN
1.23457E+09
```

```
>
```

```
>100 PRINT .00000001234567
>RUN
1.23457E-07
```

```
>
```

```
>100 INPUT R
>200 PRINT R
>RUN
?
```

```
>SCRATCH
>10 PRINT 100 MIN 200
>20 PRINT 100 MAX 200
>RUN
100
200
```

```
>
```

```
>10 A=35
>20 B=56
>30 C=A MIN B
>40 D=A MAX B
>50 PRINT "MAXIMUM="C+A MAX D-B
>60 PRINT "MINIMUM="C+A I\MIN D-B
>70 PRINT A MIN B MAX C-D
>RUN
```

LESSON I

Summary

Variables

Letter (A to Z)

Letter-Digit (A0 to A9, . . . , Z0 to Z9)

Operator Hierarchy

** (done first)

* /

+ -

MAX, MIN (done last)

Parentheses override hierarchy.

Scientific Notation

fraction E $\pm$ *integer* (1.5067E+7)

Prompt Characters

> BASIC/3000 ready for input

? Program requests data

?? Program needs more data

??? Last input was not a legal command (this is printed which BASIC/3000 cannot understand what you have typed in).

Corrections

Replace line by retyping

After ERROR @ *n* return to suppress added message, any character not to suppress.

Delete last character with *Control H*.

Delete line with *Control X*.

Statements

Statement Number PRINT *list of formulas and/or strings*
, as separator means spread out.
; as separator means print close together.

Statement Number LET *variable = formula*

Statement Number REM *any sequence of characters*

Statement Number INPUT *list of strings and/or variables*

Commands

RUN Run the current working program

LIST List the current working program

SCRATCH Erase the current working program (Also SCR)

LESSON II

Transferring Control

This lesson covers how to repeat calculations in BASIC/3000, using the GOTO statement, IF statement, relational operators, and AND/OR. It also teaches how to stop doing something.

READ/DATA/RESTORE
GOTO
IF - THEN
<, >, =, <=, >=, <>, #
AND/OR/NOT
END
STOP

Suppose you want the program to calculate and print the area of many circles. One way to do this in BASIC/3000 is to use READ/DATA statements. The DATA statement contains a list of values; each time a variable is requested with the READ statement, the next value is taken.

This program reads the three radius values (1, 2, 3), calculates the area and prints the results. However, you can see that we had to repeat the essential statements three times (i.e., lines 600-700-800 and 900-1000-1100 duplicate lines 300-400-500). When the number of repetitions becomes large, this becomes very inconvenient. What we would like to do is go back to line 300 after completing line 500.

GOTO STATEMENT

Here is the program rewritten using the GOTO statement. This statement unconditionally (always) transfers control to the statement number specified. When the program is run, it repeats the read-calculate-print operation until it runs out of data. When the READ statement attempts to read data after all the values have been used, the program is stopped and "OUT OF DATA IN LINE *nnn*" is printed.

Suppose you wanted to print the area of a circle for integer radius values starting with 1. This program performs that function admirably, except that it will execute interminably. When you have seen enough printing you stop the program by striking *Control Y*.

If you rewrite this program to use an INPUT statement instead of a READ statement it can determine the area for any radius if you enter the radius from the terminal. This program is also never-ending, but typing *Control Y* during input will stop it.

```

>SCRATCH
>100 DATA 1,2,3
>200 P=3.14
>300 READ R
>400 A=P*R**2
>500 PRINT "FOR RADIUS ";R;" AREA IS ";A
>600 READ R
>700 A=P*R**2
>800 PRINT "FOR RADIUS ";R;" AREA IS ";A
>900 READ R
>1000 A=P*R**2
>1100 PRINT "FOR RADIUS ";R;"AREA IS ";A
>1200 REM TOO MUCH WORK! AND I MADE A MISTAKE IN LAST LINE.
>RUN
>SCR
>100 DATA 1,2,3,4,5,6,7
>200 P=3.14
>300 READ R
>400 A=P*R**2
>500 PRINT"FOR RADIUS ";R;" AREA IS ";A
>600 GOTO 300
>1200 REMARK THIS IS EASY NOW!!!
>RUN
FOR RADIUS 1      AREA IS 3.14
FOR RADIUS 2      AREA IS 12.56
FOR RADIUS 3      AREA IS 28.26
FOR RADIUS 4      AREA IS 50.24
FOR RADIUS 5      AREA IS 78.5
FOR RADIUS 6      AREA IS 113.04
FOR RADIUS 7      AREA IS 153.86
OUT OF DATA IN LINE 300
>
>SCR
>100 R=1
>200 PRINT"FOR RADIUS ";R;" AREA IS ";3.14*R**2
>300 R=R+1
>400 GOTO 200
>RUN
FOR RADIUS 1      AREA IS 3.14
FOR RADIUS 2      AREA IS 12.56
FOR RADIUS 3      AREA IS 28.26
FOR RADIUS 4      AREA IS 50.24
FOR RADIUS 5      AREA IS 78.5
>100 P=3.14
>200 INPUT R
>300 PRINT "FOR RADIUS ";R;" AREA IS ";P*R**2
>400 GOTO 200
>RUN
?.1234
FOR RADIUS .1234      AREA IS 4.78145E-02
?10
FOR RADIUS 10      AREA IS 314
>

```

IF STATEMENT

Suppose you want to use READ-DATA, but want to perform some action after the data is exhausted. The program must be able to determine when it is out of data and refrain from asking for more. BASIC/3000 provides the IF statement (conditional transfer of control) for this purpose. It consists of a test, $N < 11$ (N less than 11), and a statement number. If the test is true, control transfers to the statement specified. If the test is false, control transfers to the next statement in line.

Here is a modification of the previous program which uses IF, READ, DATA, and a new statement — RESTORE. This program demonstrates an additional feature of DATA: all of the DATA statements in a program are effectively a single list that starts with the lowest numbered DATA statement and proceeds upward. The READ statement takes the next value in the list and adjusts a pointer so that the subsequent READ will obtain the next value. When a program begins, the pointer is set to the first value in the first DATA statement. The purpose of the RESTORE statement is to reset this pointer so that the data can be read again. This program also demonstrates a feature of the IF statement. The test part of an IF statement can be any numerical expression, including a simple variable; if the result is not zero, the test is true; if the result is zero, the test is false. Note that the variable G is used to enter 0 for no, 1 for yes in line 1000. Then it is tested in line 1200.

The testing mechanism of the IF statement is even more powerful than we have shown. The following relations can be tested:

=	(equals)
<	(less than)
>	(greater than)
>=	(greater than or equal)
<=	(less than or equal)
<> or #	(not equal)

```

>100 DATA 7,45,1,.025,8,4.5
>200 DATA .1,.2,.3,.5,.4
>300 N=0
>400 READ R
>500 N=N+1
>600 PRINT "FOR RADIUS ";R;" AREA IS ";3.14*R*R
>700 IF N<11 THEN 400
>800 PRINT "WE ARE OUT OF DATA . STOP ."
>RUN
FOR RADIUS 7 AREA IS 153.86
FOR RADIUS 45 AREA IS 6358.5
FOR RADIUS 1 AREA IS 3.14
FOR RADIUS .025 AREA IS 1.96250E-03
FOR RADIUS 8 AREA IS 200.96
FOR RADIUS 4.5 AREA IS 63.585
FOR RADIUS .1 AREA IS .0314
FOR RADIUS .2 AREA IS .1256
FOR RADIUS .3 AREA IS .2826
FOR RADIUS .5 AREA IS .785
FOR RADIUS .4 AREA IS .5024
WE ARE OUT OF DATA . STOP .

```

>

```

>800 PRINT"WE ARE OUT OF DATA."
>900 PRINT"DO YOU WISH TO PRINT THEM AGAIN?"
>1000 INPUT "ENTER 0 FOR NO, 1 FOR YES:",G
>1100 RESTORE
>1200 IF G THEN 300
>RUN

```



The result is true if the relation holds, false if the relation does not hold. In addition, there are three other logical operators:

NOT	(NOT true is false, NOT false is true)
AND	(true only if both sides are true)
OR	(true if either or both sides are true)

These operators can be combined with numerical operators because "true" is merely a non-zero result, "false" is a zero result. The normal hierarchy of operators is

** (Exponentiate)	<i>(Done First)</i>
NOT	
*,/, MOD (Multiply, divide, and modulo—take the remainder of a divide)	
+, − (Addition and subtraction)	
MIN, MAX	
=, <, >, <=, >=, <>, #	
AND	
OR	<i>(Done Last)</i>

Parentheses are used to override the normal hierarchy of execution. With all these tools available it is possible to test for very complex situations.

This program tests whether a specified month, day, and year are a legal data (taking into account how many days per month and leap year). Statement 800 performs a preliminary test to see if the data is outside of certain limits (e.g., month greater than 12). Statement 900 skips to statement 1200 if the month is not February. Statement 1000 sets L to the number of days allowed in February (28 + (1 or 0)). The test $Y \text{ MOD } 4 = 0$ tells whether the year is a leap year. Statement 1200 sets the day limit to 30 or 31. Statement 1500 tests whether the day is between 1 and L inclusive. Using statements like these you can write programs to test data for any conceivable combination of circumstances.

Note that the program was LISTed. The LIST command does not put a ">" in front of each statement.

>LIST

```
100 REM DATE VERIFICATION PROGRAM
200 INPUT "ENTER MONTH, 1 TO 12, OR -1 TO STOP?",M
250 IF M=-1 THEN STOP
300 INPUT "ENTER DAY, 1 TO 31?",D
400 INPUT "ENTER YEAR, 00 TO 99?",Y
500 IF NOT (M>=1 AND M<=12) OR (D<=0) OR (Y<=0) THEN 5000
600 IF M<>2 THEN 1200
700 L=28+Y MOD 4
800 GOTO 1500
1200 L=30+NOT ((M=9) OR (M=4) OR (M=6) OR (M=11))
1500 IF NOT (D>=1 AND D<=L) THEN 5000
1600 PRINT "GOOD DATE."
1700 GOTO 200
5000 PRINT "BAD DATE."
5100 GOTO 200
>
```

>100 REM DAT

>

>RUN

```
ENTER MONTH, 1 TO 12, OR -1 TO STOP?1
ENTER DAY, 1 TO 31?31
ENTER YEAR, 00 TO 99?79
GOOD DATE.
ENTER MONTH, 1 TO 12, OR -1 TO STOP?2
ENTER DAY, 1 TO 31?29
ENTER YEAR, 00 TO 99?41
GOOD DATE.
ENTER MONTH, 1 TO 12, OR -1 TO STOP?6
ENTER DAY, 1 TO 31?13
ENTER YEAR, 00 TO 99?47
GOOD DATE.
ENTER MONTH, 1 TO 12, OR -1 TO STOP?-1
```

>

END OF CONTROL

A program can normally end in three ways:

- By executing the last statement.
- By being stopped by the user with a *Control Y*.
- By executing an END or STOP statement.

The END and STOP statements terminate execution of a program:

Statement number END

Statement number STOP

END is usually the last statement in a program (but in BASIC/3000, a final END statement is not required). STOP can appear anywhere in a program to provide additional stopping points. The differences between END and STOP are relevant only in advanced programming situations and are described in the reference manual.

LESSON II

Summary

Control Keys

Control Y: (Break; stop printing or input and end program.)

Operators – Complete Hierarchy

**	(Exponentiate)
NOT	(Reverse truth)
*, /, MOD	(Multiply and divide, and modulo)
+, -	(Add and subtract)
MIN, MAX	(Minimum and maximum)
=, <, >, <=, >=, <>, #	(Relations)
AND	(True, if both true.)
OR	(True, if either true.)

Statements

Statement Number READ *list of variables*

Assign values from DATA statement sequentially to variables.

Statement Number DATA *list of values*

Statement Number RESTORE

Reset pointer to first DATA statement.

Statement Number GOTO *Statement Number*

Transfer control to statement specified.

Statement Number IF *expression* THEN *Statement Number*

Transfer control to statement specified if *expression* result is true (non-zero). If result is false (zero), continue with next statement.



LESSON III

Looping

One of the most common occurrences in programming is repetition — the performance of the same task more than one time. This lesson covers the method in BASIC/3000 of looping (the FOR and NEXT statements) and a form of data storage which is often used with looping (MATRICES).

FOR
NEXT
DIM
A (I)
A (I,J)
MAT INPUT
MAT READ
MAT PRINT



Here is a simple case of repetition or looping. Statements 400, 500, and 600 are repeated until $N \geq 11$.

FOR STATEMENT

Since this type of program structure is needed so often, a special looping statement is provided — the FOR statement. Here is the same program rewritten with a FOR statement. The FOR statement consists of:

Statement Number FOR variable = expression TO expression

The variable is assigned the value of the first expression, then tested to see if it is greater than the expression after TO (the limit). If it is not, the following statements are executed until the NEXT statement. The NEXT statement consists of

Statement Number NEXT variable

Where *variable* matches the variable in the FOR statement. When the NEXT occurs, the variable in the FOR statement and the NEXT statement is incremented by 1 and tested against the limit again. This continues until the limit is exceeded; at this point, control transfers to the statement after the NEXT. Here the loop (300, 400, 500, 600) is executed 11 times. Run the program.

MATRICES

A matrix (or array, as it is sometimes called) is a set of variables referenced by a common name. Individual variables within a matrix are accessed by specifying a subscript number (fifth element of matrix S is $S[5]$); thus, matrix variables are sometimes called subscripted variables. The subscripts of a matrix always start with 1 and go up to N (where N is the number of variables).

The size of a matrix is defined by a DIM (dimension) statement. The number of elements in each matrix is specified after the matrix name in brackets (or parentheses). Any matrices not defined in a DIM statement are assumed (in BASIC/3000) to have 10 elements. The elements of the matrix are accessed by typing the matrix name, followed by a number in parentheses, which specifies the element desired. This number must be between 1 and the size of the matrix, since elements are numbered 1, 2, 3, . . . n . In this program, a matrix, D, is filled with 10 values from the keyboard and these values are used as radii to calculate the area of circles. In statement 300 optional feature of FOR is demonstrated—the STEP expression. This allows the FOR statement variable to be modified by any number after each execution of the loop instead of just incrementing by 1.

What was used in the last example is called a one-dimensional array—it consisted of a list of variables, each one uniquely specified by a single subscript. BASIC/3000 also allows two-dimensional arrays matrices consisting of rows and columns, where each variable must be defined by two subscripts (row number and column number).

```

>SCR
>100 DATA 7,45,1,.025,8,4.5
>200 DATA .1,.2,.3,.5,.4
>300 N=0
>400 READ R
>500 N=N+1
>600 PRINT "FOR RADIUS ";R;"AREA IS ";3.14*R*R
>700 IF N<11 THEN 400
>800 PRINT "WE ARE OUT OF ALL DATA."
>
>

>100 DATA 7,45,1,.025,8,4.5
>200 DATA .1,.2,.3,.5,.4
>300 FOR N=1 TO 11
>400 READ R
>500 PRINT "FOR RADIUS ";R;"AREA IS ";3.14*R*R
>600 NEXT N
>700 PRINT "WE ARE OUT OF ALL DATA."

```

```

>SCR
>100 DIM D(10)
>200 PRINT "ENTER TEN NUMBERS PLEASE."
>300 FOR I=1 TO 10 STEP 1
>400 INPUT D(I)
>500 NEXT I
>600 FOR I=1 TO 10
>700 PRINT"FOR RADIUS";D(I);"AREA IS";3.14*D(I)**2
>800 NEXT I
>900 PRINT "DO YOU WIS \H TO CONTINUE?"
>1000 INPUT "I FOR YES, 0 FOR NO.",Q
>1100 IF Q=1 THEN 200
>RUN

```

A two-dimensional matrix of 20 rows and 2 columns is used to hold 20 pairs of values. The first value of each pair is a part number and the second value is the amount of that part in stock. The matrix is filled using READ/DATA. The program inputs a part number from the terminal and searches through the first row of matrix P to see if there is a match. If there is no match, the part is not stock. If there is a match, the corresponding stock amount is printed from the second column of the same row. This program is listed. Note that LIST uses brackets ([]) for matrices. Brackets and parentheses are interchangeable in all contexts.

NESTING

FOR loops can be nested in BASIC/3000; that is, one FOR loop can contain another FOR loop imbedded within it.

This program accepts up to 20 values from the terminal and sorts them into increasing size using a nested loop. The outer loop steps through all the values one at a time; for each value, an inner loop searches through the rest of the values, exchanging the values whenever the first is greater than the second. The inner loop is terminated by statement 1500; the outer loop by statement 1600. In this way the smaller values are "ripple-sorted" to the beginning of the array. The entire sorted array is then printed out. Note in statement 2000 that LIN is used to skip lines on the terminals (the number in parentheses specifies the number of lines to skip). RUN THE PROGRAM!

```

>LIST
100 REM PARTS INVENTORY CHECKER
200 DIM P[20,2]
300 FOR I=1 TO 20
400 READ P[I,1]
500 NEXT I
600 FOR I=1 TO 20
700 READ P[I,2]
800 NEXT I
900 INPUT "WHAT PART NUMBER,0 TO STOP.",N
1000 IF N=0 THEN 3000
1100 FOR I=1 TO 20
1200 IF P[I,1]=N THEN 2000
1300 NEXT I
1400 PRINT "PART NOT IN STOCK."
1500 GOTO 900
2000 PRINT P[I,2];" ITEMS IN STOCK."
2100 GOTO 900
2200 REM PART NUMBERS
2300 DATA 200,300,401,789,451,100,205,888,123,681
2400 DATA 114,109,910,771,452,476,305,666,900,496
2500 REM AMOUNTS IN STOCK
2600 DATA 0,123,4560,11,789,4512,24,250,123,9,8
2700 DATA 7,6,5,4,3,2,1,0,100
3000 END
>RUN
WHAT PART NUMBER,0 TO STOP.771
5 ITEMS IN STOCK.
WHAT PART NUMBER,0 TO STOP.452
4 ITEMS IN STOCK.
WHAT PART NUMBER,0 TO STOP.0
>
>LIST
100 REM ARRAY SORTER
200 DIM S[20]
300 INPUT "HOW MANY ITEMS TO SORT?",N
350 IF N>20 OR N<0 THEN STOP
400 FOR I=1 TO N
500 INPUT S[I]
600 NEXT I
650 REM START OF ACTUAL SORT
700 FOR I=1 TO N
800 FOR J=I+1 TO N
900 IF S[I]<=S[J] THEN 1500
1000 T=S[I]
1100 S[I]=S[J]
1200 S[J]=T
1500 NEXT J
1600/NEXT I
1700 REM END OF ACTUAL SORT
2000 PRINT LIN(1);"SORTED ITEMS";LIN(2)
2100 FOR I=1 TO N
2200 PRINT S[I]
2300 NEXT I
>RUN

```

INPUT/OUTPUT LOOPS

In all of our examples of looping with matrices we have used a simple combination of three statements many times:

```
FOR statement
INPUT, READ, or PRINT matrix element
NEXT
```

Because this method is so common, two short-hand looping constructs are provided in BASIC/3000.

The first method is the input/output loop – a FOR statement in parentheses followed by a comma and a list of variables to be input, read, or printed as many times as the loop specifies.

This is the parts checker program, rewritten using input/output loops. The first column of array P was filled by statement 300 and the second *column* by statement 400. These two statements can be consolidated into one statement by using nested FOR statements:

```
300 INPUT (FOR J=1 TO 2, (FOR I=1 TO 20, P[I,J] ))
400 REM NOT NEEDED ANYMORE
```

REMEMBER: The first subscript indicates the *row* and the second the *column*.

RUN THE PROGRAM!

The second method of input/output loops is the transfer of an entire matrix. There are three statements for this purpose: MAT INPUT, MAT READ, and MAT PRINT. For two-dimensional arrays, rows are transferred, element by element.

Here is the sorting program rewritten to use MAT INPUT and MAT PRINT. Note the dimension N of S in statement 400. This redefines the size of S and causes MAT INPUT to read N values from the terminal. A matrix can be redimensioned to any number of rows and columns, as long as the total number of elements does not exceed that originally allocated to the matrix and the number of dimensions is not altered. The MAT PRINT in statement 1800 prints the one-dimensional array S as a row, 5 values per line. (If a semi-colon had appeared after S, the values would have been packed together.)

RUN THE PROGRAM!

>LIST

```
100 REM CONDENSED PARTS CHECKER
200 DIM P[20,2]
300 READ (FOR I=1 TO 20,P[I,1])
400 READ (FOR I=1 TO 20,P[I,2])
500 INPUT "WHAT PART NUMBER.0 TO STOP.",N
600 IF N=0 THEN 3000
700 FOR I=1 TO 20
800   IF P[I,1]=N THEN 2000
900 NEXT I
1000 PRINT "PART NOT IN STOCK."
1100 GOTO 500
2000 PRINT P[I,2];" ITEMS IN STOCK."
2100 GOTO 500
2200 REM PART NUMBERS
2300 DATA 200,300,401,789,451,100,205,888,123,681
2400 DATA 114,109,910,771,452,476,305,666,900,496
2500 REM AMOUNTS IN STOCK
2600 DATA 0,123,4560,11,789,4512,24,250,123,9,8
2700 DATA 7,6,5,4,3,2,1,0,100
3000 END
>
```

>LIST

```
100 REM CONDENSED SORTER
200 DIM S[20]
300 INPUT "HOW MANY ITEMS TO SORT.",N
350 IF N>20 OR N<0 THEN STOP
400 MAT INPUT S[N]
500 FOR I=1 TO N
600   FOR J=I+1 TO N
700     IF S[I]<=S[J] THEN 1500
800     T=S[I]
900     S[I]=S[J]
1000    S[J]=T
1500  NEXT J
1600 NEXT I
1700 PRINT LIN(1);"SORTED ITEMS";LIN(2)
1800 MAT PRINT S
>RUN
HOW MANY ITEMS TO SORT.5
?5,4,3,2,1
```

SORTED ITEMS

1

2

3

4

5

>

LESSON III

Summary

Statement Number FOR *variable* = *expression* TO *expression* STEP *expression*
 (FOR A = 1 TO 100) OPTIONAL

Statement Number NEXT *variable*

(Variable matches FOR statement variable)

Statement Number DIM *list of dimensioned matrices*

one-dimension = D(20)

two-dimension = D(20,20)
 ↑ ↑
 rows, columns

Statement Number INPUT
Statement Number READ
Statement Number PRINT

These can have (*FOR* statement, *variable list*)
 in variable list.

Statement Number MAT INPUT *Matrix list*
Statement Number MAT READ *Matrix list*
Statement Number MAT PRINT *Matrix list*

Subscripts change
 Matrix size

(Rows transferred first.)

LESSON IV

Functions and Subroutines

Often in programming it is necessary to perform the same function or task several places within a program. Instead of repeating the statements necessary to perform the task, it is most convenient to list the statements only once and refer to them only when they are needed. BASIC/3000 supplies two methods of reaching this goal — functions and subroutines.

A function is an operation or set of operations which can be called upon in an expression; it receives one or more values which are called parameters and produces a result which is the function value. Many functions are provided by the BASIC/3000 Interpreter itself (e.g., find the square root of a value, etc.), but the user can define others for himself.

A subroutine is a position in the program to which the program transfers with a GOSUB statement. (GOSUB transfers control in a similar fashion to GOTO). When a RETURN statement is encountered in the subroutine, control transfers back to the statement following the last GOSUB statement. This allows the program to use one collection of statements in many places.

INT, ABS, SGN, SIN, etc.
GOSUB
RETURN
DEF
GOTO-OF

PREDEFINED FUNCTIONS

A predefined function is identified by a three-letter name and is called by using the name with the parameter(s) in parentheses. Here are four common functions:

INT(X) Largest integer less than or equal to X .
ABS(X) Absolute value of X .
SGN(X) Sign of X ; returns 1 if $X > 0$,
 0 if $X = 0$, or -1 if $X < 0$.
SIN(X) Trigonometric sine of X
 (X is an angle in radians).

These functions can be called in any expression and their parameter, X , can itself be any expression:

```
100 A = SGN (N + 25) * INT (ABS (SIN (Q)))
```

There are more functions:

SQR(X) Square root of X .
COS(X) The trigonometric cosine of X (X is an angle in radians).
TAN(X) The trigonometric tangent of X (X is an angle in radians).
PIX(X) π times X .

Write a program to print the value of each of these functions (labeled as such) for numbers entered from the terminal. Run the program. Can you find any values which cause problems in the functions?

There are many other special purpose functions predefined in BASIC/3000, some of which will be brought out (as necessary) in this manual. We have already seen one of them, LIN, which skips lines on output printing. These functions are not called in expressions, only in certain limited contexts, such as a PRINT statement.

```

>10 REM DEMONSTRATE FUNCTIONS
>15 PRINT " X   INT(X) ABS(X) SGN(X) SIN(X)"
>20 FOR X= -5 TO 5
>30 PRINT X;INT(X);ABS(X);SGN(X);SIN(X)
>40 NEXT X
>RUN

```

X	INT(X)	ABS(X)	SGN(X)	SIN(X)
-5	-5	5	-1	.958924
-4	-4	4	-1	.756803
-3	-3	3	-1	-.14112
-2	-2	2	-1	-.909297
-1	-1	1	-1	-.841471
0	0	0	0	0
1	1	1	1	.841471
2	2	2	1	.909297
3	3	3	1	.14112
4	4	4	1	-.756803
5	5	5	1	-.958924

```

>

```

SUBROUTINES

A subroutine is just a block of statements like any other in the program except that it contains a RETURN statement. The programmer can GOSUB to any statement number in the program. For example:

```
100    GOSUB    1000
200    A = A + 1
      :
      :
      :
1000    REM SUBROUTINE
      :
      :
      :
1500    RETURN
```

A GOSUB statement is similar to a GOTO, except that it allows the program to return to the original location.

Also, statements within the subroutine can GOSUB to other subroutines before RETURNing. This is called a "nested" subroutine; and is allowed in BASIC/3000 up to 10 deep. That is, 10 GOSUBs can occur without any RETURNS. The RETURN statement always goes back to the point of the most recent GOSUB.

This program plays a simple guessing game. You must guess a random number between 1 and an upper limit. A subroutine at line 3000 picks the random numbers. The subroutine is called in line 500. Notice that the guessing becomes harder each time you guess correctly and easier each time you guess incorrectly.

The method used to pick the numbers demonstrates a use of INT and introduces a new function — RND — which returns a random number greater than or equal to 0 but less than 1. To generate the same sequence of random numbers each time a program is run (if you want to compare results) the first call made to RND in the program must have a negative parameter. (Use the same number each time the program is run.) Subsequent calls in the program should be made with a parameter greater than or equal to zero. If the parameter is greater than or equal to zero, the random numbers will start at a different point each time the program is executed.

The program can be terminated by typing *Control Y* (Y^C) or reaching an upper limit of 10.

Can you modify this program to increase the upper limit (H) for correct answers faster than you decrement it for wrong answers? This would make it easier to win the game.

```

100  REM NUMBER GUESS GAME
200  H=3
300  IF H=10 THEN 2500
400  PRINT "I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN "
450  PRINT "1 AND ";H;" INCLUSIVE."
500  GOSUB 3000
600  PRINT "WHAT IS THE NUMBER";
700  INPUT A
800  IF A=N THEN 2000
900  PRINT "WRONG.  THE NUMBER WAS";N;"."
1000 IF H=2 THEN 1200
1100 H=H-1
1200 GOTO 300
2000 PRINT "RIGHT."
2100 H=H+1
2200 GOTO 300
2500 PRINT "YOU HAVE BROKEN THE BANK.  YOU WIN."
2600 STOP
3000 REM RANDOM NUMBER
3100 LET N=INT(H*RND(0))+1)
3200 RETURN
3300 END

```

```

I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 3      INCLUSIVE.
WHAT IS THE NUMBER?2
WRONG.  THE NUMBER WAS 1      .
I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 2      INCLUSIVE.
WHAT IS THE NUMBER?2
WRONG.  THE NUMBER WAS 1      .
I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 2      INCLUSIVE.
WHAT IS THE NUMBER?2
RIGHT.
I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 3      INCLUSIVE.
WHAT IS THE NUMBER?3
RIGHT.

```

USER-DEFINED FUNCTIONS

We mentioned previously that the programmer can design his own functions in BASIC/3000. He does this with the DEF statement. Each DEF statement defines the purpose of one function and is not executed until the function is called (i.e., it is like a DIM statement in this way). User-defined functions are identified by the letters FN and any alphabetic character (i.e., FNA, FNB, . . . , FNZ). The statement is of this form:

1000	DEF	FNB	(A)	=	LOG (A) / LOG (10)
<i>Statement Number</i>		<i>Name</i>	<i>Parameter</i>		<i>Expression</i>

This example returns the common logarithm (base 10) of any expression substituted for A in a function call:

```
100      A5 = FNB (53.2 * Q)
```

Functions can receive more than one parameter. For example, a function can return the average of 5 numbers:

```
1100     DEF  FNA (A,B,C,D,E) = (A+B+C+D+E)/5
```

Note that A, B, C, D, and E above are dummy variables; that is, they hold the place of actual values which will be specified when ever the function is called. When the function is run the parameter names are not used; the values specified in the parameter positions are used:

```
110      A6 = FNA (6,7,12,32,0)
          (A = 6, D = 7, C = 12, D = 32, E = 0)
```

Here is the number guess program, rewritten using a function to pick the number instead of a GOSUB. Run it to verify that it works the way the previous program ran. Can you modify it to generate the same sequence of numbers each time it is run. If you can, then you will be able to "break the bank."

```

100  REM NUMBER GUESS GAME
200  H=3
300  IF H=10 THEN 2500
400  PRINT "I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN "
450  PRINT "1 AND ";H;" INCLUSIVE."
500  N=FNA(H)
600  PRINT "WHAT IS THE NUMBER";
700  INPUT A
800  IF A=N THEN 2000
900  PRINT "WRONG.  THE NUMBER WAS";N;"."
1000 IF H=2 THEN 1200
1100 H=H-1
1200 GOTO 300
2000 PRINT "RIGHT."
2100 H=H+1
2200 GOTO 300
2500 PRINT "YOU HAVE BROKEN THE BANK.  YOU WIN."
2600 STOP
3000 REM RANDOM NUMBER
3100 DEF FNA(J)=INT(J*RND(0))+1)
3200 END

```

RUN

```

I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 3      INCLUSIVE.
WHAT IS THE NUMBER?2
RIGHT.
I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 4      INCLUSIVE.
WHAT IS THE NUMBER?3
WRONG.  THE NUMBER WAS 1      .
I HAVE THOUGHT OF A RANDOM NUMBER BETWEEN
1 AND 3      INCLUSIVE.
WHAT IS THE NUMBER?3
WRONG.  THE NUMBER WAS 1      .

```

LESSON IV

Summary

Predefined functions

INT(<i>X</i>)	Largest integer less than or equal to <i>X</i> .
ABS(<i>X</i>)	Absolute value of <i>X</i> .
SGN(<i>X</i>)	Sign of <i>X</i> (1 if <i>X</i> > 0, 0 if <i>X</i> = 0, -1 if <i>X</i> < 0).
SIN(<i>X</i>)	Sine of <i>X</i> (radians).
COS(<i>X</i>)	Cosine of <i>X</i> (radians).
TAN(<i>X</i>)	Tangent of <i>X</i> (radians).
SQR(<i>X</i>)	Square root of <i>X</i> .
PIX(<i>X</i>)	$\pi * X$.
RND(<i>X</i>)	Random number (repeats only if first call is <0 and subsequent calls are ≥ 0).

Statements

<i>Statement number</i>	GOSUB <i>statement number</i>
<i>Statement number</i>	RETURN
<i>Statement number</i>	DEF FNa (<i>parameter list</i>) = <i>expression</i> <i>a</i> is any alphabetic character.

LESSON V

Strings and Files

All of the programming techniques covered so far in this manual have been essentially quantitative and numerical. In this section, we move on to nonquantitative capabilities of BASIC/3000 – features which allow you to manipulate strings of characters and large files of data, as is so often required in business and educational applications.

STRING VARIABLES (A\$)
STRING ARRAYS – DIM
FILES



STRING VARIABLES

A string variable is identified by an alphabetic character (or a character and a digit) and a \$ (A\$, B\$, . . . , Z\$) and can contain a string constant of up to 255 characters in length. The maximum size string which a string variable can hold is specified by a DIM statement:

```
100 DIM A$(72)
      maximum string size
```

String variables are assigned values through LET, READ, or INPUT statements. Their values can be printed using PRINT and compared in expressions using =, >, <, <=, >=, and <>. (the result is true (1) or false (0)).

When strings are compared in a numeric expression, the resulting value is either 1 (true) or 0 (false). For example

```
200   IF A$ = "END OF INPUT" THEN 300 ELSE 400
300   N = A$ <> "END OF DATR"
```

Here string variables are used to hold a person's name and address, and a date (N\$, A\$, D\$). These variables are filled from the terminal, then used to fill in spaces in a form letter (PRINT). Note that the strings are dimensioned (200, 300). Run the program entering your own name and address and some date.

>LIST

```
100 REM FORM LETTER PROGRAM
200 DIM N$(60)
300 DIM A$(60),D$(45)
400 INPUT "ENTER DEMOLITION DATE.",D$
500 INPUT "NAME?"N$
550 IF N$="STOP" THEN STOP
600 INPUT "ADDRESS?",A$
700 PRINT LIN(2),"DEAR ";N$
800 PRINT
900 PRINT "THIS LETTER IS SENT TO NOTIFY YOU THAT YOUR RESIDENCE,"
1000 PRINT A$;", IS"
1100 PRINT "SCHEDULED FOR DEMOLITION ON ";D$;"."
1200 PRINT LIN(1);"SINCERELY YOURS,";LIN(2);"R.L. GRAND"
1300 PRINT "DIRECTOR OF BEAUTIFICATION";LIN(3)
1400 GOTO 500
```

>

>RUN

```
ENTER DEMOLITION DATE.MARCH 3, 1973
NAME?ROBERT T. POORBOY
ADDRESS?121 DIRTY ST., SAN CRISTOBAL, NOWHERE
```

DEAR ROBERT T. POORBOY

THIS LETTER IS SENT TO NOTIFY YOU THAT YOUR RESIDENCE,
121 DIRTY ST., SAN CRISTOBAL, NOWHERE, IS
SCHEDULED FOR DEMOLITION ON MARCH 3, 1973.

SINCERELY YOURS,

R.L. GRAND
DIRECTOR OF BEAUTIFICATION

NAME?STOP

>

Here string variables are used to drill the person at the terminal in spelling by comparing the person's answer against a variable containing the correct answer. I\$ is the heading, N is the number of questions, Q\$ is the question, A\$ is the correct answer, B\$ is the person's answer, R is the number of questions answered correctly. Note that several string variables are dimensioned together in 175, that strings are READ from DATA statements as needed in 200, 500, and that B\$ is compared to A\$ in 900. Run this program and see if it works correctly. How would you rewrite the program to allow for some question words to be correctly spelled as is? (HINT: It is not necessary to change any of the basic logic of the program.)

```

>LIST
 100 REM COMPUTER DRILL IN SPELLING
 150 R=0
 175 DIM A$(60),B$(60),Q$(60),I$(60)
 200 READ I$,N
 300 PRINT I$,LIN(1)
 400 FOR I=1 TO N
 500   READ Q$,A$
 600   W=0
 700   PRINT Q$;
 800   INPUT B$
 900   IF B$=A$ THEN 2000
1000   W=W+1
1100   IF W=2 THEN 1500
1200   PRINT "TRY AGAIN";
1300   GOTO 800
1500   PRINT "THE CORRECT ANSWER IS: ";A$,LIN(1),Q$;
1600   INPUT B$
2000   R=R+2-W
2100   PRINT
2200 NEXT I
3000 R=R/(2*N)
3100 PRINT "YOUR SCORE IS ";INT(R*100+.5);"%"
4000 REM HEADING FOR DRILL
4100 DATA "GIVE THE CORRECT SPELLING FOR EACH WORD."
4200 REM NUMBER OF WORDS
4300 DATA 5
4400 REM INCORRECT AND CORRECT SPELLINGS
4500 DATA "ADVERTIZE","ADVERTISE","BEGINING","BEGINNING"
4600 DATA "RECIEVE","RECEIVE","EXCEDE","EXCEED"
4700 DATA "ITINERY","ITINERARY"
>RUN
GIVE THE CORRECT SPELLING FOR EACH WORD.

ADVERTIZE?ADVERTISE

BEGINING?BEGGINING
TRY AGAIN?BEGINNING

RECIEVE?RECEIVE

EXCEDE?EXCEED

ITINERY?ITINNERY
TRY AGAIN?ITINERRY
THE CORRECT ANSWER IS: ITINERARY
ITINERY?ITINERARY

YOUR SCORE IS  70  %

>

```

STRING ARRAYS

Sets of string arrays can be created as arrays (matrices), just as numerical matrices can. All elements of the array must have the same maximum size determined in a DIM statement.

```
100 DIM C$(20, 72)
```

name number of element
elements size

Maximum element size is 255; maximum bound size is limited only by the size of the computer and the program. A subscripted string variable is accessed in the same manner as a subscripted numerical variable — with an expression specifying the element desired.

This program inputs a string array from the terminal, sorts it into alphabetic order and prints it out. Maximum number of strings to sort is 20 and maximum size of each string is 10 characters. Note that when more than one string is entered through the terminal in one INPUT statement, the individual strings must be enclosed in quotes and separated by commas.

String arrays can be handled as a whole by the MAT PRINT, MAT READ, and MAT INPUT statements. In the MAT PRINT statement, when an array is followed by a comma (or nothing), the strings are printed spread apart as in normal PRINT. When a semi-colon is used the strings are printed packed together. In both cases, if there is not enough room for the entire string on the current line the string is printed starting on the next line. Rewrite the sorter program using MAT INPUT and MAT PRINT and run it.

```

>LIST
100 REM STRING SORTER
200 DIM S$(20,10),T$(10)
300 INPUT "HOW MANY STRINGS TO SORT? ",N
400 INPUT (FOR I=1 TO N,S$(I))
500 FOR I=1 TO N
600   FOR J=I+1 TO N
700     IF S$(I)<=S$(J) THEN 1100
800     T$=S$(I)
900     S$(I)=S$(J)
1000    S$(J)=T$
1100   NEXT J
1200 NEXT I
1500 PRINT LIN(1);"SORTED STRINGS"
1600 PRINT LIN(1),(FOR I=1 TO N,S$(I))
>RUN
HOW MANY STRINGS TO SORT? 5
?MARTHA
??JOHN
??ROBERT
??TIMOTHY
??SIMON

```

SORTED STRINGS

JOHN	MARTHA	ROBERT	SIMON	TIMOTHY
------	--------	--------	-------	---------

>

FILES

What is a file? It is a container for large amounts of data that will not fit in a program because the arrays would be too large. Files can hold both strings and numerical data. In the type of files to be used here — serial files — data is written into the file in serial (or sequential order). To read an item in the file we move back to the beginning of the file and skip over items until we reach the desired item.

Before a file can be used in a program it must be "created" using the **CREATE** command:

CREATE *file name, file length*

file name is a unique name, consisting of up to 8 alphabetic or numeric characters, the first of which must be alphabetic.

file length is the number of records in the file (each record is 106 words long—a number requires 2 words and a string about half as many words as there are characters in the string).

This command asks the computer to allocate a certain amount of space for your file and gives that space a name. Create a file named DRLL1 with 100 records to use in the next program. If there is already a file with that name, make up your own name.

Here is a program designed to fill a file with all the information necessary to run our spelling drill program. The file—DRILL 1— must be specified to the program in a **FILES** statement (line 200). Line 500 is the first file print; the #1 says to use the first file listed in the **FILES** statement and the item to be written is the string I\$. Since this is the first file print statement in the program, I\$ is written as the first item in the file. Subsequent file prints (800, 1400) fill the file. If we want to list the contents of the file to double check, we must first reset to the beginning of the file (line 2300). Line 2300 (**READ #1,1**) resets the file pointer to the first record of file #1, but does not read anything since no variables are listed. Note that we can read or print several items at a time in one statement (1400, 2400, 2700).

```

>CREATE DRLL1,100
>
>LIST
  100 REM DRILL FILL PROGRAM
  150 DIM I$(72),Q$(72),A$(72)
  200 FILES DRLL1
  300 PRINT "ENTER COURSE HEADING"
  400 INPUT I$
  500 PRINT #1;I$
  600 PRINT "ENTER NUMBER OF QUESTION-ANSWER PAIRS: ";
  700 INPUT N
  710 IF N<=50 THEN 800
  720 PRINT "NUMBER TOO LARGE"
  730 GOTO 700
  800 PRINT #1;N
  900 FOR I=1 TO N
1000  PRINT "Q #";I;
1100  INPUT Q$
1200  PRINT "A #";I;
1300  INPUT A$
1400  PRINT #1;Q$,A$
1500 NEXT I
2000 PRINT "LISTING?YES OR NO";
2100 INPUT Q$
2200 IF Q$="NO" THEN 5000
2250 REM NEXT STATEMENT RESETS FILE TO START
2300 READ #1,1
2400 READ #1;I$,N
2500 PRINT I$,LIN(1),N,"PAIRS"
2600 FOR I=1 TO N
2700  READ #1;Q$,A$
2800  PRINT "Q #";I;Q$
2900  PRINT "A #";I;A$
3000 NEXT I
5000 END
>

```

The general form of file PRINT and READ is

Statement number PRINT #*n*; *variable list*

Statement number READ #*n*; *variable list*

n is the position of the file in the FILES statement (up to 16 files can be "opened" at any one time by a program).

Given what you have so far you should be able to rewrite our drill program to read its test data from the file DRILL1 instead of from DATA statements? Run the program to be certain that the results are identical. Putting the test data on a file allows the drill program to ask many more questions than when all questions must fit in the program.

When you attempt to READ beyond the data written in a file (i.e., beyond the logical end-of-file) or you attempt to PRINT beyond the physical end-of-file, your program is stopped. To avoid stopping your program when you don't know exactly how much data is in a file, use the ON END statement:

Statement number ON END #*n* THEN *statement number*

This tells the program to go to the statement specified when it reaches the end of file #*n*.

When you attempt to read a number and the next data item in a file is a string, your program is stopped. To avoid this you can look ahead at the next data item using the TYP function to determine what type of item it is

TYP(<i>X</i>)	<i>X</i> is the file number;
	returns 1 for number
	2 for string
	3 for end of file

Using these two capabilities (ON END and TYP) you can write a program to list the contents of any file, one data item per line. Try the program on our file DRLL1.

```

>SCR
>100 REM EXPANDED DRILL PROGRAM
>200 FILES DRLL1
>300 R=0
>400 DIMA$(72),B$(72),Q$(72),I$(72)
>500 READ #1;I$
>600 PRINT I$,LIN(2)
>700 READ #1;N
>800 FOR I=1 TO N
>900 READ #1;Q$,A$
>1000 W=0
>1100 PRINT Q$;
>1200 INPUT B$
>1300 FI\IF B$=A$ THEN 3000
>1400 W=W+1
>1500 IF W=2 THEN 3\2500
>1600 PRINT "TRY AGAIN"
>1700 GOTO 1200
>2500 PRINT "THE CORRECT ANSWER IS: "A$,LIN(1);Q$
>2600 INPUT B$
>3000 R=R+2-W
>3100 PRINT
>3200 NEXT I
>3500 R=R/(2*N)
>3600 PRINT "YOUR SCORE IS ";INT(R*100+.5);"%"
>5000 END
>

```

```

>SCR
>100 FILES DATUM
>200 DIM A$(255)
>300 ON END #1 THEN 1000
>400 IF TYP(1)=1 THEN 600
>500 IF TYP(1)=2 THEN 700
>600 READ #1;A
>650 PRINT A
>675 GOTO 400
>700 READ #1;A$
>750 PRINT A$
>800 GOTO 400
>1000 PRINT "FILE LIST COMPLETED"
>

```

FILE COMMANDS

A number of commands are available for file management (in addition to CREATE):

DUMP	<i>file name</i>	Lists the file.
NAME	<i>program name</i>	Names the current working program so that it can be saved as a program file.
SAVE		Saves the current working program as a program file so that it can be retained. The file is created and given the name of the program.
GET	<i>program name</i>	Makes the program file named into the current working program.
PURGE	<i>name</i>	Eliminates the program or data file named.

LESSON V

Summary

STRING VARIABLES: A\$, B\$, C\$, ..., Z\$

String variable relation string variable

relation is <, >, =, <=, >=, <>, #

FUNCTIONS

TYP(*n*) *n* = file number

Result = 1 (number)
2 (string)
3 (end-of-file)

COMMANDS

CREATE *file name, file length*
DUMP *file name*
NAME *program name*
SAVE
GET *program name*
PURGE *name*

STATEMENTS

<i>Statement number</i>	DIM	<i>string variable (size), string array (bound, size)</i>
<i>Statement number</i>	INPUT	<i>string variable</i>
<i>Statement number</i>	READ	<i>string variable</i>
<i>Statement number</i>	PRINT	<i>string variable</i>
<i>Statement number</i>	LET	<i>string variable = string or string variable</i>
<i>Statement number</i>	MAT PRINT	<i>string array</i>
<i>Statement number</i>	MAT READ	<i>string array</i>
<i>Statement number</i>	MAT INPUT	<i>string array</i>
<i>Statement number</i>	FILES	<i>list of file names</i>
<i>Statement number</i>	PRINT	<i>#n; list of variables</i>
<i>Statement number</i>	READ	<i>#n; list of variables</i>
<i>Statement number</i>	ON END	<i>#n, THEN statement number</i>



SUMMATION

You have used BASIC/3000 to calculate mathematical formulas, make decisions, repeat operations, manipulate strings, and handle large volumes of information. But, what you have done is only a small portion of what BASIC/3000 can do.

- Would you like to deal with other types of numbers, such as integers, more precise real numbers (11.7 digits of accuracy) or complex numbers? BASIC/3000 has all of these.
- Would you find it useful to perform operations such as multiplication, addition, subtraction, inversion, and transposition upon entire matrices? BASIC/3000 allows these operations for matrices of all numeric data types.
- Would you like to print out reports in a format other than that provided by the PRINT statement? The PRINT USING statement in BASIC/3000 allows exact specification of every character output on the terminal.
- What if you cannot fit all of the tasks required of a function onto one line or in an expression? Multi-line functions are available in BASIC/3000.
- Would you sometimes like to access parts of a string, rather than the whole string? Or add strings together to form new strings? These capabilities are part of BASIC/3000.
- What more could you do if you could access any part of a file without reading or skipping from the beginning? Direct access to any part of a file is provided in BASIC/3000.
- When a program is too large to fit in the computer in one piece, wouldn't you like to be able to break it into manageable segments that could call each other? The CHAIN, COM, and INVOKE statements in BASIC/3000 allow just this ability.

Why not get a copy of the full BASIC/3000 manual (*HP 3000 BASIC Interpreter, 03000-90008*) and start to work?



APPENDIX

How to Use BASIC/3000 Without a Terminal

BASIC/3000 can be accessed even though you do not have a terminal. You punch all your commands and statements on punched cards in exactly the same format as described for terminals in the book. You do *not* put a greater than (>) at the beginning of each line.

Instead of the :HELLO command, you use the :JOB command; this is the form,

:JOB user-name/password . account-name/password

See the man in charge of the computer for your *user-name* and *account-name* and *passwords*.

Instead of the :BYE command, you use :EOJ (end of job).

Here is a typical deck of cards to access BASIC/3000:

<i>Begin Job</i>	:JOB BOBG/ZIP95 . STUDENT/QQQ768
<i>Begin BASIC</i>	:BASIC
	100 INPUT A, B, C, D, E
<i>BASIC/3000 program</i>	200 S = A + B + C + D + E
	300 PRINT "SUM= "; S
	400 PRINT "AVERAGE= "; S/5
<i>BASIC/3000 command</i>	RUN
<i>input data</i>	5, 100, 93, 10001, 777
<i>End BASIC</i>	EXIT
<i>End Job</i>	:EOJ

All output is printed on the high-speed line printer device.

