DOCUMENT RESUME

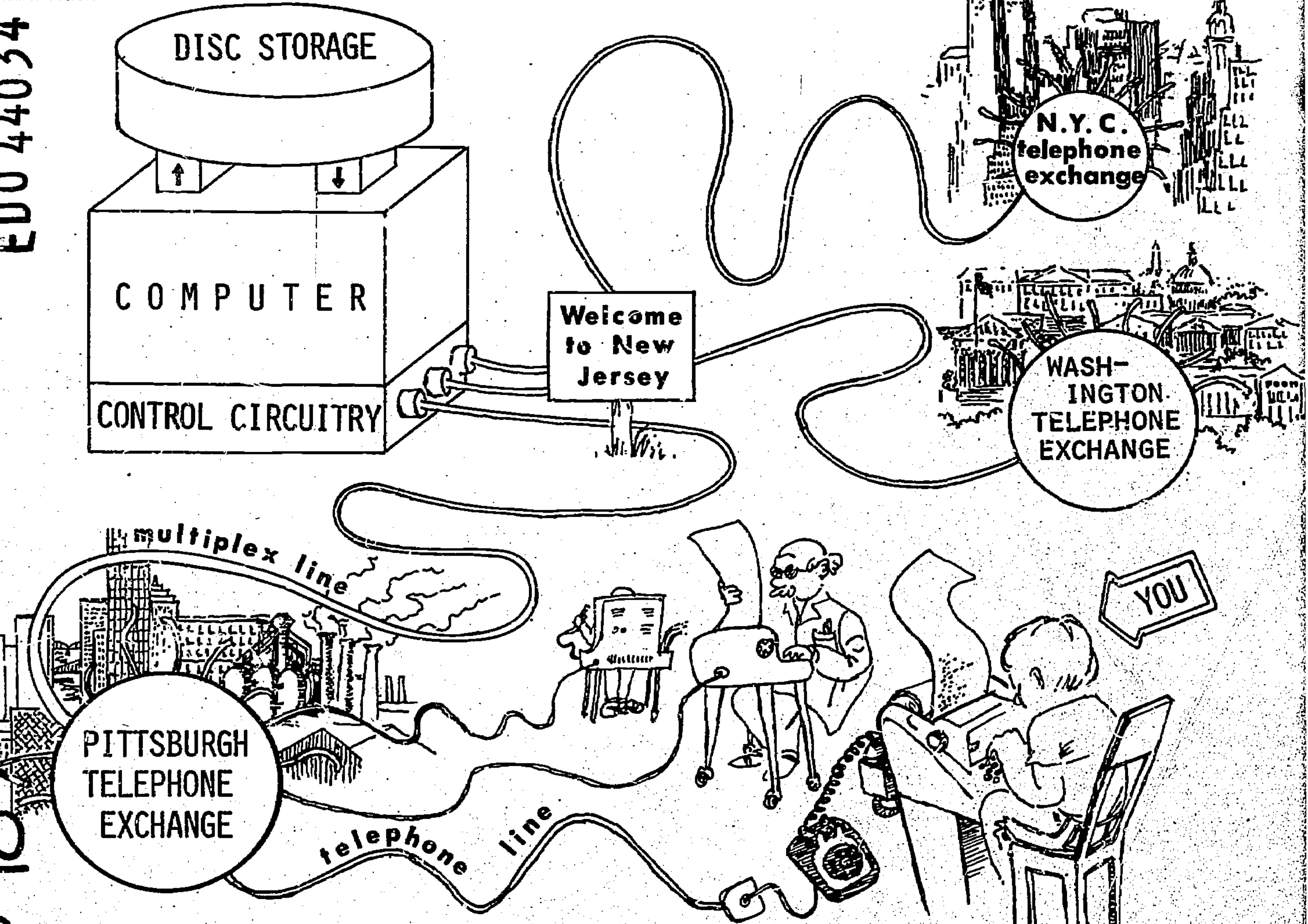
ED 044 034 EM 008 489

AUTHOR	Dwyer, Thomas A.; And Others
TITLE	A Primer for the NEWBASIC/CATALYST System.
INSTITUTION	Pittsburgh Univ., Pa.
PUB DATE	Oct 70
NOTE	54p.
EDRS PRICE	EDRS Price MF-$0.25 HC-$2.80
DESCRIPTORS	Computer Assisted Instruction, *Manuals, *Programing Languages, Programs, Time Sharing
IDENTIFIERS	BASIC, Beginners All Purpose Symbolic Instruction Code, CATALYST, Computer Augmented Teaching and Learning System, NBS Basic, *NEWBASIC/CATALYST System

ABSTRACT

Assuming no previous experience with computers, this primer is designed to help students, teachers, scientists, and other scholars to learn how to use the NEWBASIC/CATALYST system (NBS). The primer contains nine sections: (1) instructions for establishing contact with the computer (logging on); (2) examples and problems to lead the student through the use of the rudiments of NBS; (3) an introduction to advanced NBS and the use of files; (4) an introduction to Com-Share Executive commands, with application to CAI lesson management; (5) explanations and examples of the CATALYST features; (6) information about the use of functions in NBS; (7) suffixes, multiple statements, extended data types, matrix operations, debugging commands, and scientific applications; (8) format control, QED (text editor), business and administrative applications; and (9) a combined index and summary of NEWBASIC/CATALYST. (MF)

A Primer for the NEWBASIC/CATALYST System



October, 1970

Prepared at the University of Pittsburgh by T. Dwyer, C. Len, E. Zielinski, V. Salko, M. Critchfield and M. Staton.

INTRODUCTION

This primer, which assumes no previous experience with computers, is designed to help you learn how to use the NEWBASIC/ CATALYST* system (abbreviated NBS) for communicating with a high speed computer. As the picture on the cover suggests, many different persons will be using the same computer "simultaneously". This is possible through time-sharing, a scheme that gives each user a turn, but in such a rapid fashion that he seems to have exclusive use of the machine.

Users control the computer by typing in commands at the keyboard of a special device called a terminal. These commands are sent to the computer over telephone lines, with responses coming back to the user over the same wires.

The commands have to be in a "language" that the computer will understand. There are several such languages, but not all of them are suited to the manner in which human beings (who have very extensive vocabularies) prefer to state their problems. The structure of the language used in NBS is one of the best ever developed from this point of view. It is particularly useful for handling the complex needs of students, teachers, scientists, and other scholars who employ computers as an aid to learning as well as for problem solving of all kinds.

The primer contains nine sections:

*BASIC (Beginners All purpose Symbolic Instruction Code) was developed at Dartmouth College. NEWBASIC is an advanced version of BASIC developed by Com-Share, Inc. CATALYST (Computer Augmented Teaching And Learning sYSTem) was developed at the University of Pittsburgh.

ESTABLISHING CONTACT WITH THE COMPUTER (LOGGING ON)

You will be using a teletype, a typewriter-like device, to communicate with the computer. Follow the steps below each time you wish to use the computer:

1. Turn the knob on the lower right corner of the teletype to LINE.

2. Dial the number given you by your instructor on the telephone next to the teletype console.

3. When you hear a high-pitched tone on the line, insert the phone receiver into the coupler (the small rectangular box which has rubber receptacles for holding the telephone receiver), with the cord at the end marked "cord".

4. The following will be typed on the paper roll. (The computer types the parts we have underlined. You should respond with the non-underlined portions. After you respond, press the return button. This sends your response to the computer.):

COM-SHARE CENTER K 40
PLEASE LOG IN:K166B^CE^C;MS

At this point the user typed an account number followed by the password B^CE^C, a semi-colon, the user code MS, and a a carriage return. B^CE^C are made by holding down the CTRL key while striking B and E. They will not actually print.

```
READY, SYSTEM W04
 JUL 9 17:08
LAST LOGIN JUL 9 15:17
```

```
-NBS
VER. SEP 8 17:04
```

-NBS means you wish to use the NBS system.

```
>PRINT 3.5+2.0*4.0 (carriage return)
 11.5
>PRINT (3.*5.)/(9.*5.)
 0.333333333
```

This tells the computer to PRINT 3.5 + (2.0 X 4.0). Computers use * for multiplication, / for division. Parentheses are used to group terms.(see note)

```
>PRINT SQRT(256.)/7.
 2.285714236
```

SQRT means "square root of". 7. means 7.000000000

```
>PRINT I;I*I;I↑.25 FOR I=2 TO 5
 2      4     1.189207115
 3      9     1.316074013
 4      16    1.414213562
 5      25    1.495348781
```

This line tells the computer to print I, I squared, and the fourth root of I, for I = 2, 3, 4, and 5.

```
>EXIT
```

This says you want to leave NBS.

```
-LOGOUT
USAGE
CCU:   003
CLT:   0.08 HOURS
```

This says you want to be disconnected from the computer. If the usage message does not print out, ask for help.

(Hang up the telephone.)

The above is an example of a session at a terminal using the computer as a calculator in what is called direct mode. It is called "direct" because the computer does what you tell it immediately after you press the return button. You should go to a terminal and try the above before reading any further.

INDIRECT MODE. The rest of this Primer will concentrate on using what is called "indirect" or "stored program" mode, which is a much more powerful way of using NBS. You will notice that in indirect mode you place a number at the beginning of each line. This tells the computer to "store" your instructions, and only carry them out (in the order of your line numbers) when you say RUN.

NOTE: The order in which calculations are done is to first evaluate functions (like SQRT), then do exponentiation (↑ or **), then multiplication and division, then addition and subtraction, UNLESS parentheses indicate otherwise. Thus 3+4-5*6/7 is interpreted as (3+4)-(5*6)/7, when you also apply the fact that the computer scans from left to right. When in doubt, use parentheses to clarify your intentions.

AN EXAMPLE OF INDIRECT MODE NBS

```
CØM-SHARE CENTER K 40
PLEASE LØG IN:K166B^CE^C;MS
```

An account number, followed by the password B^CE^C; followed by the user code MS, followed by a carriage return.*

```
READY, SYSTEM W04
 JUL 9  17:08
LAST LØGIN  JUL  9 15:17

-NBS
VER. JUL  9  9:37
```

A system command to ask for the New BASIC System.

```
>10 LET A=5
>20 LET B=34567
>30 PRINT "THE SUM IS", A+B
>40 END
```

This is an NBS program. The other sections of this series concentrate on learning to write such programs. These statements will become clearer as you proceed.

```
>RUN
```

This statement causes the computer to execute the preceding NBS program. It may be repeated as many times as desired without retyping the entire program.

```
THE SUM IS    34572
```

Program output for the above program.

```
>EXIT
```

Takes you out of NBS to either leave the computer or begin a new program (this can be done by typing NBS again at this point).

```
-LOGOUT
```

Informs the computer that you are leaving. Do not leave the teletype without doing this or you will be overcharged.

```
USAGE
CCU:   001
CLT:   0.03  HØURS
```

Reports the amount of computer time you used.

*The user must press the carriage return key at the end of any line he enters. We will not show this in the rest of the Primer, but you must always press the RETURN key at the end of a line.

Before you leave the teletype, hang up the telephone and turn the lower right knob to off, unless another user is ready to begin.

The lines typed after the symbol > are NBS commands. The lines typed after the symbol - are called executive system commands. Before examining more of these commands, we should first learn to correct typing errors.

CORRECTING ERRORS

1. If you type A^C (called "control A" which means you hold down the "CTRL" key while pressing "A"), an up-arrow (↑) is printed. This means that the character to the left of the arrow has been erased in the computer.

   Example:

   ```
        >10 PRR↑INT A(4↑↑5
   ```
   is interpreted by the computer as:
   ```
         10 PRINT A5
   ```

2. If you type W^Ca back slash (\) is printed. This means all characters up to but not including the preceding comma or blank, or to the start of the line are erased.

   Example:
   ```
        >10 INTA\PRINT A(\B+A
   ```
   is taken as:
   ```
         10 PRINT B+A
   ```

3. If you type Q^C a back arrow (←) is printed. This means that the entire current typed line is erased and a carriage return and line feed are given.

   Example:

   ```
        >10 LET C=←
         15 LET C=10     [Note:  No > is given]
   ```
   means that only line 15 is accepted by the computer:
   ```
         15 LET C=10
   ```
4. Suppose you don't want line 15 in your program any more. Just type >15 followed by a carriage return.

5. If you type a line over, the last line typed replaces any previous line with the same number.

   Example:

   ```
       10 LET X=5*7/8
       20 PRINT X
       10 LET X=6*7/9
   ```

   really means (to the computer)

   ```
       10 LET X=6*7/9
       20 PRINT X
   ```

USE OF SECTION 2 OF THE PRIMER

Section 2 consists of example NBS programs and problems. It is suggested that you first read each example, and then try it on a terminal.

The problems require that you spend some time away from the terminal writing out the program that you think solves the problem. You should then try your program on a terminal. If you have trouble, your teacher can supply solutions to the problems. Don't move ahead until you understand each problem. Feel free to also try whatever variations on the problems your ingenuity might suggest.

All of the examples in this section will be variations on the problem of calculating population figures for future years, using the simplified assumption that each year's growth can be expressed as a percentage of the population at the beginning of the year.

Example 1(a): Population Growth

The 1970 Census for Pittsburgh shows a population of approximately 600,000. Assuming a growth of 5% each year, the following program calculates and prints the population for each year from 1971 to 1980.

10 LET P=600000	Assigns the initial population P*(commas are omitted!).
20 LET Y=1970	Assigns the initial year to Y.
25 LET Y=Y+1	Calculates the next year.
30 IF Y>1980 GØTØ 70	This is an IF statement. If the year Y has become greater than 1980, it tells the computer to continue at instruction 70; otherwise the computer executes the next instruction. (line 40)
40 LET P=P+.05*P	Increases the population by 5% of its latest value and assigns this new value to P.
50 PRINT Y;P	Year and population are printed. The semi-colon puts two to four spaces between them on the output sheet, and leaves room for a sign--see below.
60 GØTØ 25	Go back to instruction 25 to increment Y and continue from there.
70 PRINT "FINISHED"	The word FINISHED will be printed on the output.
80 END	The last statement of any NBS program should be an "END".

The output (the results printed by the computer) of this program is:

```
>RUN
 1971      630000
 1972      661500
 1973      694575
 1974      729303.75
 1975      765768.9375
 1976      804057.3844
 1977      844260.2536
 1978      886473.2662
 1979      930796.9296
 1980      977336.776
FINISHED
```

*P is a variable name. In NBS a variable name must be a single alphabetic letter or single alphabetic letter followed by a single number. (Examples A, B, Z, Z9, A1, I, N3).

Example 1(b): Another Technique

Another student discovered a second way to write the same problem. Instead of using an IF statement along with several LET statements to increment Y and check for the final year, he used a FOR-NEXT loop. This is his program below:

```
10 LET P=600000
20 LET Y=1970
30 FOR Y=Y+1 TØ 1980 STEP 1
```

This is a FOR statement. In this case, Y is first given the value of Y+1. Then the computer automatically checks to see if Y is greater than 1980. If it is, it skips to the instruction after the NEXT Y instruction, in this case, instruction 70. Otherwise it continues with 40.

```
40 LET P=P+.05*P
50 PRINT Y;P
60 NEXT Y
```

Each FOR statement in a program must have its own matching NEXT statement. When the computer sees this, it automatically goes back to the FOR statement and increments Y by the value of the number following the word STEP (in this case it "increments" or "increases" Y by 1).

```
70 PRINT "FINISHED"
80 END
```

Note: Line Numbers

1. Each instruction of the program is numbered. These numbers may be any integer five digits or less, indicating the order in which you wish the program to run. The first instruction above has a line number (LN) 00010, which may be written as 0010, 010, or 10. Thus line numbers go from 00000 to 99999.

2. The instructions and their proper line numbers may be typed into the computer in any order. The computer will then reorder them for you in ascending order.

Problem 1: Variation on Population Growth

Now adapt this program to print out the populations of Pittsburgh from 1981 to 1990, assuming that in this period the rate of population growth will be eight per cent or .08, and that the population in 1980 is 977337.

First write the program below:

```
10
20
30
__
__
__
__
```

Now run Problem 1 on the computer to see what your output looks like. When you are finished or if you are having serious problems, ask your instructor to see how someone else did it. This same procedure should be used for each program you are asked to write.

Note: Modifying Your Program

1. If you wish to modify a few lines of the program you just wrote, or add lines, just type in the new lines and then type >RUN.

2. If you wish to enter a new program you must first destroy your previous program. To do this type EXIT*. This takes you out of NBS. Then type NBS. This brings you back into NBS and at the same time destroys your previous program. Now you are ready to type in your new program.

* Note that the symbol > must be typed by the computer before the command EXIT works. If you can't get a ">", press the escape (ESC) key first. If the computer asks CONTINUE?, answer NO.

Example 2(a): New Information

It would be useful to have figures for the population increase for every five years between 1975 and 2000. The program below is adapted to meet this need. Although calculations are done for each year, printing is only done every five years. K is used as a counter to determine when it is time to print.

```
100 PRINT "YEAR","PØPULATIØN"
```

Headings for output, note that the comma places each element fifteen spaces from the beginning of the previous element (see output).

```
105 LET P=600000
110 LET Y=1970
```

```
115 LET Y=Y+5
```

The year is increased by five, since we will be printing every five years.

```
120 IF Y>2000 GØTØ 150
```

If Y is greater than the last year we desire printed, we are finished.

```
125 FØR K=1 TØ 5 STEP 1
130 LET P=P+.05*P
135 NEXT K
```

A FOR-NEXT loop uses K to count to five in steps of 1 to do calculations for each one of the five years. All the steps between the FOR statement and the NEXT statement are executed for each value of K (in this case, K = 1, 2, 3, 4, 5,). When K is greater than five, the computer skips to instruction 140.

```
140 PRINT Y,P
```

Prints under headings.

```
145 GØTØ 115
```

Goes back to calculate another group of five years.

```
150 PRINT "FINISHED"
155 END
```

The output from this program is:

```
>RUN
YEAR           PØPULATIØN
 1975           765768.9375
 1980           977336.776
 1985           1247356.908
 1990           1591978.623
 1995           2031812.964
 2000           2593165.425
FINISHED
```

Example 2(b):

Another student wrote the following program which produces the same output, but uses a FOR-NEXT loop to increment J and check for the final year.

```
100 PRINT "YEAR","PØPULATIØN"
105 LET P=600000
110 LET Y=1970

115 FØR J=Y+5 TØ 2000 STEP 5          Each J indicates the next year
                                      for which P is to be printed.
                                      In this case, J = 1975, 1980,
                                      1985, 1990, 1995, and 2000.

120 LET K=1                           K counts each individual year
                                      in a five year group; it begins
                                      with one.

130 IF K>5 GØTØ 150                   When the fifth year's calcula-
                                      tions are complete, it skips to
                                      print; otherwise it continues
135 LET P=P+.05*P                     calculating P.
```

140 LET K=K+1	K is increased for the next year.
145 GØTØ 130	Goes back to check K.
150 PRINT J,P	After calculations for five years are completed, the current year and population are printed.
155 NEXT J	Time to get a new J. The computer goes back to the FOR statement (instruction 115) to increment and test J.

```
160 PRINT "FINISHED"
165 END
```

Problem 2: Use of "Nested"* FOR Loops

Can you find a third approach using two FOR-NEXT loops in your program? Write your program out on one of the coding sheets (supplied by your teacher) before trying it out on a terminal.

*The word "Nested" indicates that one FOR-NEXT loop is placed within a second FOR-NEXT loop, etc., etc.

Example:

```
     >10 FØR I=1 TØ 3
     >20 FØR J=1 TØ 2
     >30 PRINT I, J         INNER LOOP        OUTER LOOP
     >40 NEXT J
     >50 NEXT I
     >60 END
     >RUN
```

Output:

```
     1         1
     1         2
     2         1
     2         2
     3         1
     3         2
```

The J-loop is nested within the I-loop.

Example 3(a): Reading in Data

It has now been determined that the rate of population increase changes for each five year period, rather than remaining at a constant .05. Taking this new fact into consideration, the following program was written to determine five-year population growth.

```
100 PRINT " YEAR",   (LF)
"POPULATION", "RATE" (CR)
```

(LF) means line feed; this is used instead of carriage return (CR) to continue a line. Notice that the space before YEAR positions the heading more accurately (see output).

```
105 LET Y=1970
110 LET P=600000

115 LET S=5
```

S is assigned the numer of years in each period; in this case we always have five year periods.

```
120 FØR J=Y+S TØ 2000 STEP S
```

S is used to determine the STEP value. Once the FOR statement begins the STEP value cannot be changed within the loop.

```
125 READ I
```

Each time this statement is executed, a new value is read from the data list in steps 160 and 161 and put into I. I becomes the new rate of increase.

```
130 FØR K=1 TØ S
```

Since the STEP value is 1, it can be omitted.

```
135 LET P=P+I*P
```

I is now used in the calculation.

```
140 NEXT K
145 PRINT J,P,I
150 NEXT J
```

The rate for each period is also printed.

```
155 PRINT "FINISHED"

160 DATA .05,.03,.08
161 DATA .06,.04,.05

165 END
```

Each time a READ statement is executed, the next number is read from this list. There can be more than one READ statement in a program.

The output from this program is:

```
>RUN
 YEAR           PØPULATIØN       RATE
 1975            765768.9375      0.05
 1980            887736.0761      0.03
 1985            1304375.541      0.08
 1990            1745548.712      0.06
 1995            2123726.907      0.04
 2000            2710473.495      0.05
FINISHED
```

Further Explanation of Data Statements:

When the computer first finds a READ statement (Example: 125 READ I), it looks for a data statement (Example: 160 DATA .05, .03, .08), and assigns the first item (.05) in the data list to variable given in the READ statement (in this example, I). The second time a READ statement is found, the second item (.03) on the data list is assigned to the variable given in the READ statement, etc. In the example above, there are six items in the data list, so the programmer should be certain that the READ statement is not executed more than six times.

There can be as many DATA statements as you wish, as long as the number of data items matches the number of times the program executes the READ statement.

In the above example we could use:

```
160 DATA  .05, .03, .08, .06, .04, .05
```

or

```
160 DATA  .05, .03, .08
161 DATA  .06, .04, .05
```

or

```
160 DATA  .05, .03
161 DATA  .08, .06
162 DATA  .04, .05
```

etc.

Items on a DATA list are thus "used up" each time a READ takes place. A special command called RESTORE can be used to reactive the data lists so that they can be used again.

Problem 3: Variable Time Periods

It is also desirable to see the population growth, when the rate of increase changes for variable time periods. For example, for the first five years, the rate is .05, for the next three years the rate is .08, etc. Below is output obtained from such a program. See if you can write a program to produce the same report.

Hints:

1. The number of years in each period might be also read from the DATA statement.
2. Once a FOR-NEXT loop is begun in NBS, the step value cannot be changed. Since we do not have a constant STEP value (the time periods change) it might be better to use an IF statement to control the testing of J. The incrementing for each time period is then controlled by a LET statement.

Output:

```
>RUN
 YEAR          PØPULATIØN      RATE
 1975           765768.9375     0.05
 1978           964648.3198     0.08
 1984           1368372.078     0.06
 1985           1423106.961     0.04
 1987           1568975.425     0.05
 1990           1922062.361     0.07
 1995           2020106.859     0.01
 1998           2272345.481     0.04
 2000           2601608.342     0.07
FINISHED
```

Write your program out on a coding sheet or blank paper before running it.

<u>Example 4(a)</u>: Area Per Person

Rather than write a separate program whenever it was desired to change the initial year and last year of the study, it was decided to permit the user to enter the years and the initial population each time the program runs. In addition, the number of square feet per person is also calculated each time a year is printed. For this, you must estimate the area of the city of Pittsburgh. This is the program which resulted.

100 INPUT Y	The computer presents a question mark, after which the user types in the value for Y (the year of the study).
105 INPUT P	Produces a second question mark, after which a value for P is typed. P is the population for the initial year.
110 INPUT E	Produces a third question mark for inputting the last year desired.
115 PRINT " YEAR", "PØPULATIØN","SQ.FT./PERSØN"	Notice the new heading.
120 LET S=5 125 LET I=.05	Using a constant rate of .05 again.
130 LET W=5	The width of the city in miles.
135 LET L=11	The length of the city in miles.
140 LET W=W*5280 145 LET L=L*5280	Changes miles to feet.
150 LET F=W*L	Determines square feet in the city, assuming it has an area equivalent to a rectangle 5 miles wide and 11 miles long.

```
155 FØR J=Y+S TØ E STEP S
```

Each value in this statement is indicated by a variable expression.

```
160 FØR K=1 TØ S
165 LET P=P+I*P
170 NEXT K

180 PRINT J,P,F/P
```

Square feet per person is also calculated and printed.

```
185 NEXT J
190 PRINT "FINISHED"
195 END
```

The output from this program is:(the user typed in the three words after the question marks)

```
>RUN
?1970
?600000
?2000
 YEAR           PØPULATIØN      SQ.FT./PERSØN
 1975            765768.9375     2002.316789
 1980            977336.776      1568.867598
 1985            1247356.908     1229.248815
 1990            1591978.623     963.1486113
 1995            2031812.964     754.6521392
 2000            2593165.425     591.2896976
FINISHED
```

Problem 4: Beyond Pittsburgh

It would be advantageous to be able to use the same program to report the population growth for any city, state, or country desired. However this makes it necessary to allow the user to input at the beginning of the program the length and width of the area, the rate of increase, and the time period for which printing is desired as well as the initial year, initial population and final year. Write such a program on a coding sheet or the rear of this page. In addition, calculate the number of persons per square foot for each year printed.

Example 5(a): A User Control

With so many input statements it is difficult for a user to remember the order in which the values are inputted. Comments before each INPUT statement, identifying the value to be inputted are therefore useful.

It is also convenient to be able to branch back to the beginning of the program to print a report for a new city without having to type a new RUN statement. In fact, it is possible to ask the user his preference. The following program incorporates these ideas.

```
100 PRINT "TYPE THE CITY'S INITIAL PØPULATIØN"
105 INPUT P
110 PRINT "TYPE THE RATE ØF INCREASE USING DECIMAL NØTATIØN"
115 INPUT I
120 PRINT "CØNSIDER THE CITY AREA AS A RECTANGLE"
121 PRINT "TYPE ITS APPRØXIMATE WIDTH IN MILES"
125 INPUT W
130 PRINT "TYPE ITS APPRØXIMATE LENGTH IN MILES"
135 INPUT L
140 PRINT "TYPE THE YEAR"
145 INPUT Y
150 PRINT "TYPE THE FINAL YEAR FØR WHICH YØU WANT THE"
151 PRINT "PØPULATIØN CALCULATED"
155 INPUT E
160 PRINT "TYPE THE TIME PERIOD FØR WHICH YØU WANT THE"
161 PRINT "CALCULATED PØPULATIØN PRINTED"
165 INPUT S
200 PRINT "YEAR","PØPULATIØN","SQFT/PERSØN","PERSØN/SQFT"
210 LET W=W*5280
212 LET L=L*5280
215 LET F=W*L
220 FØR J=Y+S TØ E STEP S
225 FØR K=1 TØ S
230 LET P=P+I*P
235 NEXT K
240 PRINT J,P,F/P,P/F
245 NEXT J
250 PRINT "IF YØU WISH TØ DØ FURTHER CALCULATIØN TYPE YES."
```

```
255 INPUT A$
```

Variables followed by a dollar sign ($) contain character strings (a character is any number, letter or special symbol). These variables cannot be used in calculations, although they may be compared to each other or to a string of characters enclosed in quotes.

```
260 IF A$="YES" GØTØ 100
265 PRINT "FINISHED"
270 END
```

In a string comparison, the two strings must be exactly the same, for th statement to be true.

Output:

```
>RUN
TYPE THE CITY'S INITIAL PØPULATIØN
?600000
TYPE THE RATE ØF INCREASE USING DECIMAL NØTATIØN
?.05
CØNSIDER THE CITY AREA AS A RECTANGLE
TYPE ITS APPRØXIMATE WIDTH IN MILES
?5
TYPE ITS APPRØXIMATE LENGTH IN MILES
?11
TYPE THE YEAR
?1970
TYPE THE FINAL YEAR FØR WHICH YØU WANT THE
PØPULATIØN CALCULATED
?2000
TYPE THE TIME PERIØD FØR WHICH YØU WANT THE
CALCULATED PØPULATIØN PRINTED
?5
 YEAR          PØPULATIØN     SQFT/PERSON    PERSON/SQFT
 1975           765768.9375    2002.316789    4.994214729E-04
 1980           977336.776     1568.867598    6.374024178E-04
 1985           1247356.908    1229.248815    8.135049537E-04
 1990           1591978.623    963.1486113    1.038261373E-03
 1995           2031812.964    754.6521392    1.325113848E-03
 2000           2593165.425    591.2896976    1.691218372E-03
IF YOU WISH TO DO FURTHER CALCULATION TYPE YES.
?YES
```

NOTE: 4.994214729E-04 is "Scientific Notation" for

$$4.994214729 * 10^{-4} = 4.994214729 * .0001$$
$$= .0004994214729$$

```
TYPE THE CITY'S INITIAL PØPULATIØN
?600000
TYPE THE RATE ØF INCREASE USING DECIMAL NØTATIØN
?.05
CØNSIDER THE CITY AREA AS A RECTANGLE
TYPE ITS APPRØXIMATE WIDTH IN MILES
?5
TYPE ITS APPRØXIMATE LENGTH IN MILES
?11
TYPE THE YEAR
?1970
TYPE THE FINAL YEAR FØR WHICH YØU WANT THE
PØPULATIØN CALCULATED
?1975
TYPE THE TIME PERIØD FØR WHICH YØU WANT THE
CALCULATED PØPULATIØN PRINTED
?1
 YEAR           PØPULATIØN      SQFT/PERSØN     PERSØN/SQFT
 1971             630000          2433.828571     4.108752817E-04
 1972             661500          2317.931973     4.314190458E-04
 1973             694575          2207.55426      4.529899981E-04
 1974             729303.75       2102.432628     4.75639498E-04
 1975             765768.9375     2002.316789     4.994214729E-04
IF YØU WISH TØ DØ FURTHER CALCULATIØN TYPE YES.
?NØ
FINISHED
```

Problem 5: A User Controlled Package

The user may not always desire the information on available land area for the population. Write a program which makes the calculation and printing of this information optional, according to the user's specification at the beginning of the program. Use a separate coding sheet to write your program, if you wish.

SECTION 3 - ADVANCED NBS FEATURES; USING FILES

MORE ON THE FOR-NEXT STATEMENT

In this section we illustrate some alternate forms of the FOR-NEXT statements that are available in NBS

FOR...TO...STEP

`10 FOR I=1 TO 5 STEP 1`	I is initialized to 1. Each time this statement is returned to by the NEXT statement, I is incremented by the STEP value of 1. Therefore I takes on the values 1, 2, 3, 4, and 5. When I is greater than 5, execution skips to the instruction following the NEXT statement.
`20 PRINT I`	Prints the value of I.
`30 NEXT I`	Goes back to statement 10.
`40 END`	

```
RUN
1
2
3
4
5
```

`10 FOR I=1 TO 10 STEP 3`	Similar to previous example except I now is incremented by 3.
`20 PRINT I`	
`30 NEXT I`	
`40 END`	

```
RUN
 1
 4
 7
10
```

FOR...STEP...UNTIL

`10 FOR I=1 STEP 5 UNTIL I>20`
`20 PRINT I`
`30 NEXT I`
`40 END`
	I is initialized to 1 then incremented by 5. The instructions within the loop are executed until I attains a value greater than 20. At this time the loop is skipped and the instruction after the NEXT statement is executed.

```
RUN
 1
 6
11
16
```

The statements between FOR and NEXT are executed as long as the expression (I>20) is false. When it becomes true, the loop is skipped.

FOR...STEP...WHILE

```
10 FOR I=1 STEP 3 WHILE I<10
   initial    value    Test
      I        of
    value   increment
20 PRINT I
30 NEXT I
40 END
RUN
1
4
7
```

I is given the initial value of 1 then incremented by 3. If I is less than 10, the statements between the FOR and NEXT statement are executed. If I is greater than or equal to 10, execution skips to the statement following the NEXT instruction.

EXAMPLES OF USING DELETE (DEL) IN NBS

```
-NBS
10 INPUT A
20 INPUT B
30 PRINT "A=";A
40 PRINT "B=";B
50 PRINT "A+B";A+B
60 END
RUN
?33
?66
A=      33
B=      66
A+B     99
50 PRINT "A+B";A+B;"A*B";A*B
DEL 20
LISTNH
10 INPUT A
30 PRINT "A=";A
40 PRINT "B=";B
50 PRINT "A+B";A+B;"A*B";A*B
60 END
DEL 40-50
5 PRINT "START"
LISTNH
5 PRINT "START"
10 INPUT A
30 PRINT "A=";A
60 END
RUN
START
?7
A=      7
```

50 PRINT "A+B";A+B;"A*B";A*B — This replaces the old line 50.

DEL 20 — To delete line 20.

DEL 40-50 — To delete lines 40 through 50.

5 PRINT "START" — Step 5 is new.

'FOR' WITHOUT 'NEXT'

FOR is usually used with NEXT as in the following example:

```
>10 FOR I = 1 TO 10 STEP 2
>20 PRINT I*I
>30 NEXT I
```

In NBS the above program can be written with one line as follows:

```
>10 PRINT I*I FOR I = 1 TO 10 STEP 2
```

We call FOR a SUFFIX when used this way. Here is another example that uses FOR as a suffix in lines 10, 20, and 30. It also illustrates the use of "+" between strings (line 20) which is called string concatenation.

```
>5 DIM A$(5)
>7 PR."WHAT ARE FIVE GØØD ADJECTIVE MØDIFIERS FØR 'BABY'?"
>10 INPUT A$(I) FØR I=1 TØ 5
>20 LET A$(I)= A$(I)+" BABY" FØR I=1 TØ 5
>25 PR."LET'S SEE HØW THEY LØØK:"
>30 PRINT A$(I) FØR I=1 TØ 5
>40 END
>RUN
WHAT ARE FIVE GØØD ADJECTIVE MØDIFIERS FØR 'BABY'?
?LØVABLE
?CUTE
?PLAYFUL
?BØUNCING
?LAUGHING
LET'S SEE HØW THEY LØØK:
LØVABLE BABY
CUTE BABY
PLAYFUL BABY
BØUNCING BABY
LAUGHING BABY
>
```

NOTE: Step 5 above is called a DIMENSIONing statement. It is a way of saying that there will be five variables called A$(1), A$(2), A$(3), A$(4), and A$(5) available to this program. A$ is a string array (see next page for further explanation).

USE OF SUBSCRIPTED VARIABLES, DIMENSIONING ARRAYS

Mathematicians use letters like X, Y, Z, A, B, C for variable names. In NBS we do the same. We can also use variable names like X1, Y1, Z2, A3, B1, C9, i.e., a single letter followed by a single digit.

A third possibility allowed in NBS is to use names like X(1), X(2), X(3), A(23), A(24), A(25), A(26), etc. These are called subscripted variables, and a group of such variables all using the same letter is called an array. Thus, for example, we might speak of the "X array" which contains the three variables, X(1), X(2), and X(3). The 1, 2, and 3 are called subscripts of X. The subscript can be any integer, or any expression which can be evaluated by the computer. Examples: X(1), X(500), X(K), X(K+J)

Before using an array, you must warn the computer as to how many elements you will want in each array so that it will reserve space for all of them. You do this with the DIMENSION (DIM) statement.

EXAMPLE OF THE USE OF ARRAYS

Here is a program that calculates the net cost of four appliances for 15%, 20%, and 25% discounts. The original cost of each appliance is supplied by the user, and stored in the array A. The three discounted costs are stored in arrays E, F, and G, and then printed out from these arrays.

```
10 DIM A(4),E(4),F(4),G(4)
20 FOR K=1 TO 4
21 PRINT
25 PRINT "TYPE COST OF APPLIANCE":K;
30 INPUT A(K)
35 LET E(K)=A(K)*.85
40 LET F(K)=A(K)*.80
45 LET G(K)=A(K)*.75
50 NEXT K
54 PRINT
55 PR. " "," 15% DISCOUNT"," 20% DISCOUNT"," 25% DISCOUNT"
60 FOR K=1 TO 4
65 PRINT "APPLIANCE":K,E(K),F(K),G(K)
70 NEXT K
75 END
```

Statement 10 reserves space for A(1),A(2),A(3),A(4),E(1), E(2),E(3),E(4), etc.

The subscript K changes as determined by the FOR-NEXT loop.

The PRINT statements in lines 21 and 54 cause a blank line to be printed for purposes of spacing. The ";" at the end of line 25 causes the "?" generated by line 30 to print after "K" of line 25.

```
>RUN
TYPE COST OF APPLIANCE 1?56.89
TYPE COST OF APPLIANCE 2?123.99
TYPE COST OF APPLIANCE 3?8.95
TYPE COST OF APPLIANCE 4?456.00

                 15% DISCOUNT   20% DISCOUNT   25% DISCOUNT
APPLIANCE 1      48.3565        45.512         42.6675
APPLIANCE 2      105.3915       99.192         92.9925
APPLIANCE 3      7.6075         7.16           6.7125
APPLIANCE 4      387.6          364.8          342
```

USE OF GO SUB

The following is an example of the use of GOSUB-RETURN statements:

```
100 LET X=3
110 GOSUB 400
120 PRINT U,V,W
200 LET X=5
210 GOSUB 400
220 LET Z=U+2*V+3*W
230 PRINT Z
240 END
400 LET U=X*X
410 LET V=X*X*X
420 LET W=X*X*X*X+X*X*X+X*X+X    ←Subroutine (any size)
430 RETURN
```

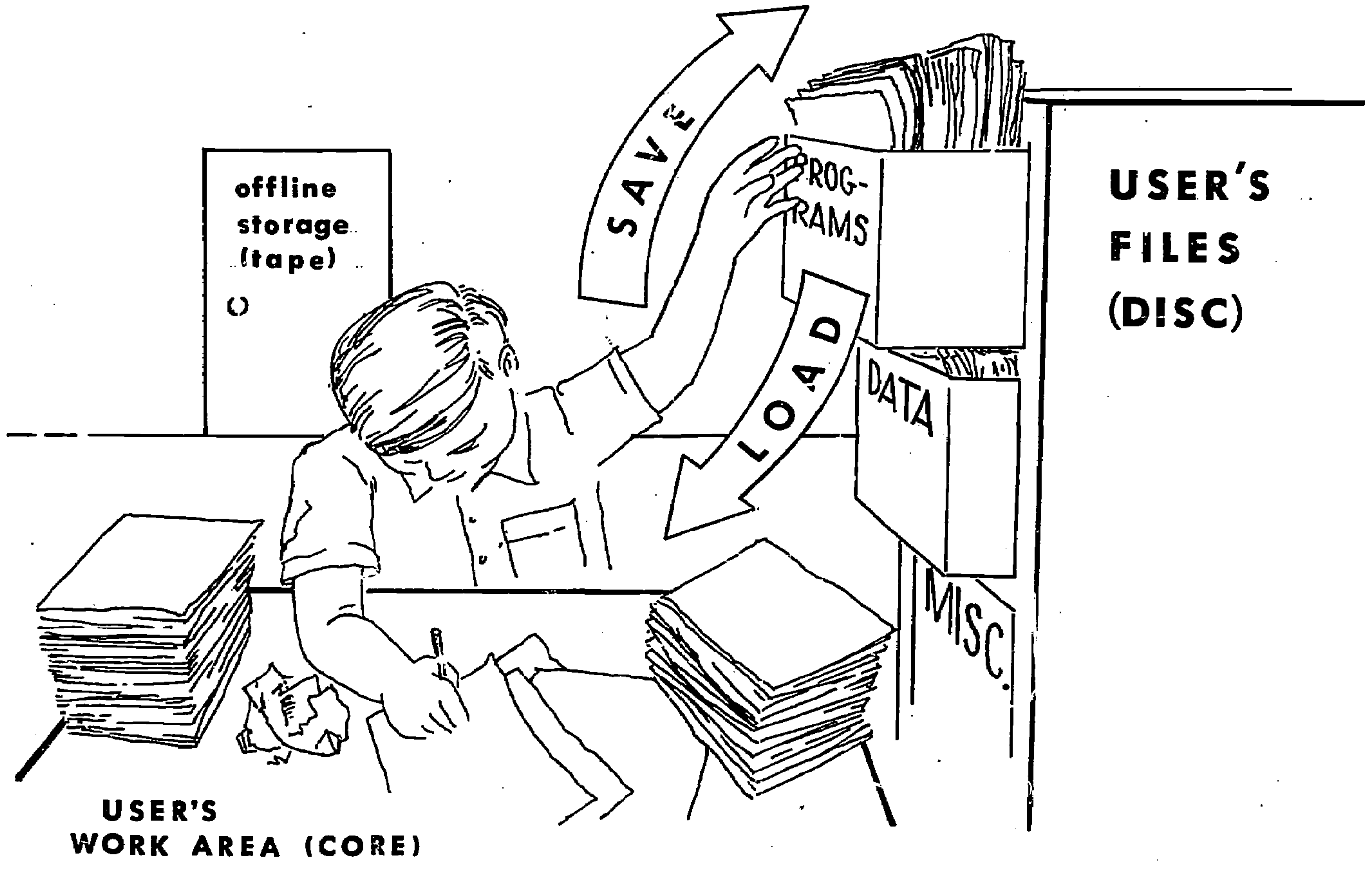
Normal return of GOSUB is to next sequential statement. In NBS you can change the normal return by using:
210 GOSUB 400 GOTO XXX
where XXX is any line number in your program.

When statement 400 is entered by the GOSUB 400 in line 110, the computations in lines 400, 410, and 420 are performed, after which the computer goes back to statement 120. When the subroutine is entered from statement 210, the computer goes back to statement 220.

GOSUB can be used in every situation where GOTO can be used, including IF and ON statements. For example:

```
                                 GOES TO   100   110   90   20
20 ON X GOSUB 100,110,90,20      When X=     1     2    3    4
```

Note the use of END in the above program. Although an END statement is not needed to terminate a sequential type program, it must be used in programs using the GOSUB-RETURN statements. An END statement should separate the main program and the subprograms to make sure that the subprograms are not executed after the main program has been processed.



Using Files in NBS

There will be times when you wish to save programs on files, loading them back into your user work area at another time. The picture above suggests a good way of thinking of this process. Although computers use electronic counterparts of the work area and the file cabinet shown in the picture, the analogy is quite accurate.

We will first show you how to carry out the saving and loading process for NEWBASIC programs, and then illustrate how files can also be used to save data for use by NBS programs, or data generated by NBS programs.

SAVING AND LOADING NBS PROGRAMS ON DISC

The commands SAVE and LOAD together with an appropriate file name do what the picture on the previous page suggests. There are a number of other commands that also help in this process. This section will explain these commands, and give examples of their use.

Previously you used two of the commands in the command language of NBS. These were:

>RUN	To execute your program.
and	
>EXIT	To leave NBS.

A few other commands available to programmers in NBS are:

>LISTNH	To type in numeric order the current numbered statement of the user's program. (NH means no heading)
>LISTNH 110	To type out the line(s) associated with the line number(s). 110 can be any legal line number or range of line numbers in the program. e.g.(LISTNH 100-110)
>SAVE /PROG/	To store a copy of all numbered statements in the user's core area on a disc file under the name given between the two slashes by the user.
>LOAD /PROG/	To copy the program stored on the file /PROG/ into the user's area. (PROG is any legal file name of a previously stored file. File names up to 9 characters are allowed).
>APPEND /PROG2/	To add the contents of the file specified between the 2 slashes to the current user's core area.

Examples of using the NBS commands explained on the previous page:

<u>LISTNH</u>

```
-NBS
>10 PRINT A,B,C
>5 INPUT A,B,C
>15 END
>RUN
?10,5,25
 10             5               25
>LISTNH
 5 INPUT A,B,C
 10 PRINT A,B,C
 15 END

>LISTNH 10-15
10 PRINT A,B,C
15 END
```

Notice that the lines are in numeric order.

<u>SAVE</u>

```
>SAVE /EXAMP/
>NEW FILE? YES
>EXIT
```

This saves the program in your user area on a disc file named /EXAMP/.

<u>LOAD</u>

```
-NBS
>LISTNH
NO PROGRAM PRESENT
>LOAD /EXAMP/
>RUN
?1,2,3
1               2               3
```

The user area is "CLEAN" when you first enter NBS.

<u>NOTE</u>: >RUN /EXAMP/ has the same effect as these two commands.

SAVE

```
-NBS
>15 LET D=A+B+C
>20 PRINT D
>25 END
>SAVE /EXAMP2/
>NEW FILE? YES
```

APPEND

```
-NBS
>LOAD /EXAMP/
```

Loads the first program.

```
>LISTNH
5 INPUT A,B,C
10 PRINT A,B,C
15 END
>APPEND /EXAMP2/
```

Adds second program to first one.

```
>LISTNH
5 INPUT A,B,C
10 PRINT A,B,C
15 LET D=A+B+C
20 PRINT D
25 END
```

Notice that line 15 is line 15 of the APPEND(ed) program. It over writes the first loaded line 15.

```
>SAVE /EXAMP/
>OLD FILE? YES
```

Answering "YES" destroys the file and puts the contents of the user area (Steps 5, 10, 15, 20, 25) on that file.

```
>LISTNH 20
20 PRINT D
>EXIT
-NBS
>RUN
NO PROGRAM PRESENT
```

```
>LOAD /EXAMP/
>RUN
?3,4,5
3             4              5
12
>EXIT
-LOGOUT
```

PAPER TAPE FILES

Since the amount of disc storage is limited, you can also use tape storage for your NBS programs as follows:

Preparing NBS Programs on Paper Tape "Off-Line"

1. Turn the knob on the lower right corner of the teletype to LOCAL. DON'T use the telephone to call the computer.
2. On the paper tape punch push down the button marked "ON".
3. Push down the "HERE IS" key until about two inches of tape are punched. This is a leader for your tape.
4. Type the first line number and NBS statement on the teletype of your program.
5. After your line is typed, hit the "RETURN" key, the "LINE FEED" key, and the "RUB OUT" key in that order.
6. If you wish to type another line, repeat steps 4 and 5.
7. Push down the "HERE IS" key until about two additional inches of tape are punched. This is a trailer for your tape.
8. Push the "OFF" button on the paper tape punch. Turn the knob on the lower right corner of the teletype to "OFF".
9. Tear off your punched paper tape.

Loading a Paper Tape "On-Line"* in NBS

1. Push the switch on the paper tape reader to the position marked "FREE".
2. Place the paper tape into the paper tape reader and lock it into position. LOGON, and ask for -NBS.
3. On the teletype, after the > type LOAD TPT, then hit "RETURN".
4. Push the switch on the paper tape reader to the position marked "START".
5. After the tape is read in (it will list while it is reading and error messages will be given) type a control D.
6. Now you should have a > and are ready to proceed with your program; for example you might type RUN, or add new statements to the program you just read in.

* On-Line means that your terminal is connected to the computer by phone lines.

Off-Line means that your terminal is not connected to the computer.

Saving New NBS Programs on Paper Tape While "On-Line"

1. Type your program into the computer as usual.
2. After your program is in the computer, to save it on tape:
   a. After the > type SAVE TPT, then hit the "RETURN" key
   b. Push the "ON" button on the paper tape punch
3. The program is listed as it is punched. When the punching is done, push down the "OFF" button on the paper tape punch.
4. Type EXIT and LOGOFF if you are finished.

Saving Old NBS Programs on Paper Tape "On-Line"

Suppose you have a program stored on the disc file /SILLY/, and you wish to remove this file from disc and save it on paper tape. Do the following:

```
-NBS
>LOAD /SILLY/
>SAVE TPT                (CR)  (Now turn on punch).
>EXIT
-DEL /SILLY/             (This deletes /SILLY/ from
                          disc.
-LOGOUT
```

Listing Paper Tapes "Off-Line"

1. Turn the knob on the lower right corner of the teletype to LOCAL. DON'T use the telephone to call the computer.
2. Push the switch on the paper tape reader to the position marked FREE. Place the tape into the reader, and lock it in position.
3. Push the paper tape switch up to START.

STORING DATA ON FILES

In addition to using files to store NBS programs, they can also be used to store data which is generated by an NBS program (writing on files), or data which is used by NBS programs (reading from files).

In NBS to access a file for reading input or writing output, the file must be opened. The same file cannot be read from and written onto at the same time. When the file is no longer needed the CLOSE command is issued. Before a file can be opened (OPEN) and used again in the same program, it must be closed.

Example for Writing onto a File (generating data)

```
>110 OPEN /STORY/ FOR OUTPUT AS 3
```

OPEN is the statement that allows you to access a file, in this case the file named /STORY/. OUTPUT states that you want to write onto the file.

The number 3*is associated with the file /STORY/. Whenever 3 is used in a print statement it refers to file /STORY/. (See line 130)

When a file is opened for output, it will destroy anything previously written on the specified file making it a clean file.

```
>120 INPUT A$
```

Accepts alphanumeric data from the terminal.

```
>130 PRINT ON 3:A$
```

Writes the contents of A$ onto the file reference by 3.

```
>140 IF A$= "##" GØTØ 160
```

Ends program if A$= "##".

* The user chooses any integer from 2 to 9. (0 and 1 refer to the teletype).

```
>150 GØTØ 120
```

Goes back and gets more input from terminal.

```
>160 CLOSE 3
```

Closes file 3. This statement must accompany any OPEN statement. A file opened for output must be closed after the last bit of information is written on it.

```
>170 END
```

Terminates program. The only output of this program is on the file /STORY/.

```
>RUN

?ONCE UPON A
?TIME A LONG
?TIME AGO
?##
>
```

In this example, the "story" inputted by the user is the "data" written on the file.

<u>Example of Reading from a File (Using Data)</u>

```
>110 OPEN /STORY/ FOR INPUT AS 4
```

OPEN again allows you to access the file /STORY/, in this case for input. <u>INPUT</u> means that you want to read information from the file. In this example we are associating the number 4 with /STORY/.

```
>120 INPUT FROM 4:B$
```

Reads one line of information from /STORY/ and stores it in the alphanumeric variable B$.

```
>130 IF B$= "##" GØTØ 160
```

Ends program if B$= "##".

```
>140 PRINT B$                            Prints on terminal the con-
                                         tents of B$.

>150 GØTØ 120                            Goes back and reads the next
                                         line from the file.

>160 CLOSE 4                             Closes file 4 (File /STORY/).

>170 END                                 Terminates program.

>RUN

 ONCE UPON A
 TIME A LONG
 TIME AGO
```

The following programs illustrate a business application of data files. Briefly, the first program creates a master file consisting of an employee number (E), hourly rate (R), and number of hours worked (H). The second program reads the master file, then calculates and prints each employee's weekly salary (A) including time and a half for overtime, and total weekly payroll (S).

Program #1

```
     110 OPEN /MASTER/ FOR OUTPUT AS 4   (opens the file)
     111 INPUT E,R,H                     (allows the operator to
                                         type in data from the
                                         teletype)
     112 PRINT ON 4:E,R,H                (writes the data on the
                                         master file)
     113 IF E>0 THEN 111                 (allows steps 111 and
                                         112 to be repeated as
                                         often as needed)
     114 CLOSE 4                         (closes the file)
     115 END                             (ends the program)
```

Program #2

```
     209 LET S=0                         (initializes S)
     210 OPEN /MASTER/ FOR INPUT AS 4    (opens the master file
                                         created by the first
                                         program)
     211 INPUT FROM 4:E,R,H              (instructs the system
                                         to read the data from the
                                         master file)
```

```
212 IF E=0 THEN 220
213 IF H>40 THEN 218
214 LET P=H*R
215 PRINT E,P
216 LET S=S+P
217 GO TO 211                         (allows the program to
                                      go back, read the next
                                      set of data, and repeat
                                      the required computations)
218 LET P=40*R+(H-40)*R*1.5
219 GO TO 215
220 PRINT "TOTAL PAYROLL",S
221 CLOSE 4                           (closes the file)
222 END                               (ends the program)
```

Once a file has been created it may be the input to many programs. For example, the following program reads the same master file and calculates total company overtime.

Program #3

```
309 LET S=0                           (initializes S)
310 OPEN /MASTER/ FOR INPUT AS 4      (opens the master file)
311 INPUT FROM 4:E,R,H                (reads the data from the
                                      master file)
312 IF E=0 THEN 316
313 IF H<=40 THEN 311
314 LET S=S+(H-40)
315 GO TO 311                         (allows the program to
                                      go back, read the next
                                      set of data, and repeat
                                      the required computations)
316 PRINT "COMPANY OVERTIME",S
317 CLOSE 4                           (closes the file)
318 END                               (ends the program)
```

SEC. 4 - THE EXECUTIVE SYSTEM

The Com-Share system is a general purpose time-sharing system that utilizes an SDS 940, various peripheral equipment, and a network of remote terminals (teletypewriters) linked to the computer by Bell System DATA-PHONE sets. Users of the system may create and execute programs from any location by simply dialing the computer on the teletypewriter's telephone. An advantage Com-Share's users enjoy is the speed and convenience of a conversational system. This interactive characteristic of the Com-Share system allows each user complete control over his activity.

Although a large number of users may be accessing the Com-Share system at the same time, the sophisticated software that controls the system assures the complete separation of each activity. Thus, each user may conceptually imagine that he is the only user on the system.

To further enhance the convenience of programming, the Com-Share system maintains control of all information (files) created by its users. Prior to creating a file the user gives it a name. Subsequently, he may at any time request the file by that name; the system will retrieve it from peripheral storage without the user needing to know where it is.

The system assumes that any file created by a user belongs exclusively to him and will deny any other user's request to access it. Should a user wish to make his file available to either those individuals sharing his account number or all the users on the system, he may request the system to change the status of the file.

One of the significant advantages of the Com-Share system is the large number of language processors (subsystems) available to its users. They allow the system to be used for nearly any purpose. For example, it is possible to use the system as a desk calculator merely by typing in the expression to be evaluated. On the other hand, one can write highly complex scientific programs or even perform system simulation simply by selecting the appropriate subsystem. The subsystems currently available are:

BASIC - A simple, easily learned and used compiler language.
NBS/CATALYST - An advanced BASIC plus special CAI features.
CAL - A powerful conversational compiler for numerical computation (like PIL).
COMPACT - A numerical control language for machine tools.
DDT - An on-line debugging system.
XTRAN - Extended FORTRAN IV.
PDPS - A simulator for PDP-8, PDP-5, or PDP-8S computers.
COSS - A general simulation language.
QED - A sophisticated text editing system.
CODED-CAP - A network analysis language.
SNOBOL - A string manipulation language.
TAP - A machine language macro assembler.

Each of these subsystems is fully described in its own manual.

The Com-Share system also maintains a large number of permanent library routines. Users with particular problems to solve may often discover that one of the library routines can be used, affording great savings in time and effort. These routines cover such areas as mathematics, management sciences, engineering and general sciences, manufacturing, business, and utility. The reader is referred to the Library Reference Manual for a complete list of these routines. Users are also encouraged to submit their programs of general interest for possible inclusion in the library.

In order to use the system, one needs to know the telephone number of the system and be able to supply proof that he is a Com-Share customer. This proof, or identification, consists of a single string of up to 15 characters. Some of these characters are assigned by Com-Share when the user agrees to purchase time on the system, and the rest are chosen by the user. A user may elect to have non-printing characters in his identification to provide password protection.

After the user dials the appropriate telephone number and connection to the computer is made, the system will request him to type in his identification. This process is known as "logging in". If his identification is legitimate, the user will be given access to the system.

Once the user gains access to the system there is a set of commands available that allows him to inform the system about the task he expects it to perform. All of these commands are first evaluated by one of the system modules called the Executive. It subsequently directs the appropriate system module or subsystem to complete the requested task. The user, therefore, need not be concerned with the complex interface between the Executive and the rest of the system--he only needs to know the commands that comprise his interface with the Executive.

Having the Executive in complete control of the system is of great advantage to the user. Files created in one subsystem can be utilized by other subsystems because the Executive is responsible for storing and retrieving files.

EXECUTIVE COMMANDS THAT MAY BE ENTERED AFTER THE DASH (-)

The following is a brief example of some of the operations that can be performed with the Executive on Com-Share's time-sharing system:

```
PLEASE LOG IN:128COM;LL
READY,SYSTEM T04
COM-SHARE AA
APR 30     9:17
LAST LOGIN:   APR 23        12:46
```

```
UP TIL 7:30 P.M.
-FILES                    (The user requests a listing of the files
                          in his file directory)
5 FILES
/BMAC/   /PR/   /BIN/   /SORT/   /BOB/
-COPY  /SORT/
TO TELETYPE
 40 PRINT"HOW MANY NUMBERS TO BE SORTED?"
 50 INPUT N
 55 PRINT"TYPE THE NUMBERS "
 60 FOR I=1 TO N
 70 INPUT A(I)
 80 NEXT I
100 FOR I=1 TO N-1
110 FOR J=I+1 TO N
120 IF A(I)>A(J) THEN 150
130 NEXT J
140 NEXT I
145 GO TO 185
150 LET T=A(I)
160 LET A(I)=A(J)
170 LET A(J)=T
180 GO TO 130
185 PRINT"SORTED NUMBERS"
190 FOR I=1 TO N
200 PRINT A(I)
210 NEXT I
220 END

-RENAME  /SORT/  AS  /BASICSORT/       (He changes the name
                                       of the file.)
-DELETE  /PR/             (He removes one of the files from his
                          directory and again requests a listing
OK?  Y                    of his files.)
-FILES
4 FILES
/BMAC/  /BOB/  /BASICSORT/  /BIN/
-NBS                 (He requests the use of the NBS subsystem, loads
                     his program into it, and then instructs NBS to
                     run the program.)
>LOAD
FROM  /BASICSORT/
>RUN
HOW MANY NUMBERS TO BE SORTED?
? 5                  (The question marks indicate that the NBS sub-
                     system is requesting input)
TYPE THE NUMBERS
? 4
? 7
? 2
? 8
? 9
                          SORTED NUMBERS
                          2
                          4
                          7
                          8
                          9
                          > EXIT        (Exit from the NBS Subsystem)
                          -
```

On the next 3 pages we will show some additional executive commands that are particularly useful for CAI work.

Session 1

METHOD BY WHICH TEACHERS MAY CREATE A PROGRAM FOR THEIR STUDENTS' USE

```
COMSHARE CENTER K124
PLEASE LOG IN:K166;CL                          Teacher "CL" logs on.
READY,SYSTEM W04
 SEP 16 14:45
LAST LOG IN SEP 16 10:59
-NBS                                           Enters NBS.
VER.AUG 25 17:18

>10 LET Y=2                                    Creates a drill or
>20 LET X=1                                    tutorial program.
>25 IF PASS > 8 GOTO 90
>30 LET Y=Y+1
>40 LET X=X+1
>50 PRINT "WHAT IS ":X:" + ":Y
>60 INPUT A
>70 IF ABS(A-(X+Y))<.001 CALL REIN,GOTO 25
>80 PRINT "NO THE ANSWER IS ":X+Y
>85 GOTO 25
>90 PRINT "GOOD-BY FOR NOW"
>100 END
>SAVE /MATH/                                   Saves program on a file.
NEW FILE?YES
>EXIT                                          Leaves NBS.
-DEFINE /MATH/ AS PUB                          Uses the define command
                                               to make his file public
                                               (i.e., other users may
                                               access it).

-WHATS /MATH/                                  Checks the status of
                                               his file.

/MATH/            2  SYM  SEP 16 14:55  PUB RDO
                  ↑   ↑              ↑     ↑
               Size Type Date Created Status
                ↑    ↑               ↑
       (2 blocks=  (Symbolic lang.,  (Public
        2000 char.) i.e., NBS)        Read Only)

       Note:  Approximately 15 lines of NBS fit in 1 block.

-LOGOUT
```

Session 2

A STUDENT INTERACTING WITH THE TEACHER'S PROGRAM

```
COMSHARE CENTER K124
PLEASE LOG IN:K166;FW
READY,SYSTEM W04
 SEP 17 15:04
LAST LOG IN SEP 16 14:56
-NBS
VER.AUG 25 17:18
>LOAD 166CL /MATH/
```

Student "FW" logs on.

Goes into NBS.

To load the public program /MATH/ the student types the account number and the user identification code of the creator, followed by a blank and the program name enclosed in slashes.

K is not typed -(This is the name of the computer we are using.)

```
>RUN

WHAT IS  2 +  3
?5
GOOD
WHAT IS  3 +  4
?7
THAT'S RIGHT
WHAT IS  4 +  5
?8
NO THE ANSWER IS  9
WHAT IS  5 +  6
?10
NO THE ANSWER IS  11
WHAT IS  6 +  7
?13
CORRECT
WHAT IS  7 +  8
?9
NO THE ANSWER IS  15
WHAT IS  8 +  9
?7
NO THE ANSWER IS  17
WHAT IS  9 +  10
?91
NO THE ANSWER IS  19
GOOD-BY FOR NOW

>EXIT

-LOGOUT
```

Student runs the program and interacts with it. In this example, a simple arithmetic drill is shown.

The PASS statement in line 25 causes the program to terminate after 8 responses. See page 5-3 for further information.

Session 3

METHOD BY WHICH TEACHER MAKES CHANGES IN HIS "PUBLIC" LESSON

```
COMSHARE CENTER K124
PLEASE LOG IN:K166;CL                           Teacher "CL" logs on.
READY,SYSTEM W04
 SEP 18 14:50
LAST LOG IN SEP 16 14:45
-DEFINE /MATH/ AS PRI                           The file must be re-
                                                defined as "private" in
                                                order to change it.

-NBS                                            Enters NBS.
VER. AUG 25 17:18
>LOAD /MATH/
>50 PRINT "WHAT IS ":X:" * ":Y                  Changes lines 50, 70,
>70 IF ABS(A-(X*Y))<.001 CALL REIN,GOTO 30      and 80 so that the drill
>80 PR."NO THE ANSWER IS ":X*Y                  is now on multiplication.
>SAVE /MATH/
OLD FILE?YES                                    Answering "yes" causes
>EXIT                                           old file (on disc) to
                                                be erased, and the new
                                                version (in core) to be
                                                stored under the name
                                                /MATH/.

-DEFINE /MATH/ AS PUB                           Defines /MATH/ as
                                                public again.
```

Session 4

INTERACTION OF A STUDENT WHO USES /MATH/ <u>AFTER</u>

SESSION 3 IS FINISHED

```
COMSHARE CENTER K124
PLEASE LOG IN:K166;FW              Student "FW" logs on.
 SEP 18 15:30
LAST LOGIN SEP 17 15:04
-NBS                               Goes into NBS.
VER. AUG 24 17:18
>LOAD 166CL /MATH/                 This time when he loads the public
>RUN                               /MATH/ program, he gets the new
WHAT IS  2 *  3                    version, a drill on multiplication.
?6
CORRECT
WHAT IS  3 *  4
?12
RIGHT
WHAT IS  4 *  5
?22
NO THE ANSWER IS 20
WHAT IS  5 *  6
?                                  He did not wish to continue so he
←←ESC:         70                  hit the escape key.
```

SECTION 5 - USE OF THE CATALYST FEATURES IN NBS

CATALYST was originally a separate language, developed at the University of Pittsburgh for Computer Assisted Teaching and Learning. The key features of CATALYST have been re-written to be included in NBS. In other words, programmers can mix CATALYST statements right in with NBS statements. In addition to the new statement types allowed, there is a CATALYST feature called @NBS, which allows users to write and run small NBS programs while interacting with someone else's NBS program. We will first describe the new CATALYST statements, then show some sample programs using these features together with interactions that show how to use @NBS.

IS, ICO, REIN, and IBEF (as used in an IF statement)

$\boxed{\text{IS(R\$,A\$,W)}}$ R$ holds the student's response.
A$ holds a string supplied by the programmer.
W is either 0 or 1.

IS(R$,A$,0) has the value TRUE when R$ and A$ are exactly the same, value FALSE otherwise.

IS(R$,A$,1) has the value TRUE when R$ and A$ are exactly the same after special characters* are removed from in front of and after R$. It has value FALSE otherwise.

Several IS calls can be combined by using the Boolean connectives NOT, AND, OR, XOR, and BUT.

EXAMPLE:

```
>10 PR."WHAT ADJECTIVE REFERS TO SEA-GOING VESSELS?"
>20 INPUT R$
>30 IF IS(R$,"MARINE",1) OR IS(R$,"OCEANIC",0) GOTO 60
>40 PR."NO, TRY AGAIN."
>50 GOTO 20
>60 PR."CORRECT"
>70 END

>RUN

WHAT ADJECTIVE REFERS TO SEA-GOING VESSELS?
?SUBMARINE                                  Not accepted because
NO, TRY AGAIN.                              of the B in front of MARIN

?...OCEANIC!                                Not accepted because
NO, TRY AGAIN.                              W = 0.

?...MARINE!                                 Accepted because
CORRECT                                     W = 1.
```

*Anything not a number or letter. Thus, for example, #@NO! will be treated as NO, but KNOW or NO8 will not be.

ICO(R$,A$,W)

is similar to IS, except that the student can type in a long string (R$) which is scanned to see if it contains A$.

ICO(R$,A$,0) has the value TRUE when the string A$ is found anywhere in R$.

ICO(R$,A$,1) has the value TRUE when the string A$ is found anywhere in R$, provided there is not a letter or number immediately before or after the occurrence of A$ in R$. In this case we speak of looking for the 'word' A$ rather than the string A$.

CALL REIN, and
CALL RRIN

supply "mild" or "enthusiastic" reinforcement messages.

EXAMPLE:

```
 5 PR."ADJECTIVES: MARINE, WET"
 10 PR."WHAT ADJECTIVE REFERS TO THE SEA?"
 20 INPUT R$
 30 IF ICO(R$,"MARINE",1) BUT NOT ICO(R$,"WET",1) CALL REIN, GOTO 50
 40 PR."NO, TRY AGAIN" GOTO 10
 50 END
 RUN
ADJECTIVES: MARINE, WET
WHAT ADJECTIVE REFERS TO THE SEA?
?MARIN                                  Not accepted since R$ does not
NO, TRY AGAIN                           contain A$.

?SUBMARINE                              Not accepted since W = 1.
NO, TRY AGAIN
?WET OR MARINE                          Not accepted since WET is
NO, TRY AGAIN                           present in R$.

?THE ANSWER IS MARINE!                  Accepted.
CORRECT
```

IBEF(R$,A$,W,B$,W)

IBEF has the value TRUE if A$ is contained in R$ and B$ is contained in R$, but with A$ coming before B$.

The first W is made 1 or 0 depending on whether you do or do not wish A$ to be judged to be a whole 'word'.
The second W us used in the same way for B$.

e.g. IBEF(R$,"X1",1,"X(3)",0)

```
?(X1) PRECEDES X(3)                     ACCEPTED
?X12 PRECEDES X(3)                      REJECTED
?SEX1 PRECEDES X(3)                     REJECTED
?X1 PRECEDES SEX(3)                     ACCEPTED
```

EXAMPLE:

```
10 PR."WRITE A DECLARATIVE SENTENCE USING ONLY THE WORDS:
   DESKS,MEN,CARS,DEEPLY,THINK,THINGS,BUILD"
20 LET M$="MEN"
30 LET B$="BUILD"
40 LET T$="THINK"
50 INPUT R$
60 IF IBEF(R$,M$,1,T$,1) OR IBEF(R$,M$,1,B$,1) CALL RRIN,
   GOTO 80
70 PR."ARE YOU SURE THAT'S A SENTENCE?"
80 IF PASS >2 GOTO 100
90 PR."TRY ANOTHER" GOTO 50
100 END

>RUN

WRITE A DECLARATIVE SENTENCE USING ONLY THE WORDS:
DESKS,MEN,CARS,DEEPLY,THINK,THINGS,BUILD
?CARS THINK DEEPLY
ARE YOU SURE THAT'S A SENTENCE?
TRY ANOTHER
?MEN BUILD THINGS
VERY GOOD
.
.
.
```

```
100 IF PASS > N GOTO 800
110 . . . . . . . .
```

You can read this statement to mean that the program will PASS through to the next statement (in this case 110) N times. The (N+1)st time the program will branch to line 800.

EXAMPLE:

```
20 LET Y=NUM(10)-1
30 LET X=SIN(Y)
40 PR."WHAT IS THE SIN OF":Y:"RADIANS";
50 INPUT R
60 IF ABS(R-X)<.001 CALL REIN, GOTO 100
70 PR."NO, THE ANSWER IS":X
80 IF PASS >3 GOTO 200
90 GOTO 20
100 IF PASS >5 GOTO 220
110 GOTO 20
200 PR."YOU HAD 4 WRONG.  ASK YOUR TEACHER FOR HELP."
210 STOP
220 PR."YOU HAD 6 CORRECT.  NOT BAD."
230 END
```

USE OF @NBS

@NBS is a command that can be given only during the execution of an NBS program in response to a question mark (?) that is asking for input. @NBS suspends your main program and takes you into a "scratch pad" version of NBS. In this version any NBS statement or command can be used. All variables are considered new, i.e. There is no transfer of variable values between the main NBS program and the @NBS program.

A Sample Program Execution Showing @NBS Feature

```
>RUN

TYPE THE LENGTH OF A RECTANGLE

?23

TYPE THE WIDTH OF A RECTANGLE

?4

TYPE THE AREA OF YOUR RECTANGLE

?@NBS                              In response to (?) asking
                                   for input type @NBS.

VER. AUG. 24 18:13                 Now you are in the subsystem
                                   of NBS.

>PRINT 23*4                        The computer gives you >.
                                   You may do any short NBS
                                   program.

92

>EXIT                              Takes you back to your main
                                   program--all variables are
                                   at their previous values--
                                   the values and variables in
                                   the scratch pad program are
                                   lost.
```

```
?92                                        NBS now returns with the
                                           question mark for the
                                           previous input demand.

THAT'S RIGHT.  WANT TO DO ANOTHER? NO

>
```

LISTING OF THE ABOVE PROGRAM

```
 110 PRINT "TYPE THE LENGTH OF A RECTANGLE"
 120 INPUT L
 130 PRINT "TYPE THE WIDTH OF A RECTANGLE"
 140 INPUT W
 150 PRINT "TYPE THE AREA OF YOUR RECTANGLE"
 160 INPUT A
 165 IF A=L*W GØTØ 180
 170 PRINT "THAT'S WRONG TRY AGAIN"
 175 GØTØ 160
 180 PRINT "THAT'S RIGHT. WANT TO DO ANOTHER";
 190 INPUT A$
 200 IF A$="YES" GØTØ 110
 210 END
```

USE OF IEQIV

By removing line 200 in the above program, and inserting:

```
195 LET Y$=",YES,SURE,OK,O.K.,CERTAINLY,OF COURSE,YUP"
200 IF IEQIV (A$, Y$, 0) GØTØ 110
```

the previous program will branch to line 110 for a user who responds to the line 190 input request with

```
           ?YES
        or ?SURE
        or ?OK
        or ?O.K.
        or ?CERTAINLY
        or ?OF COURSE
        or ?YUP
```

Example of using @NBS to Retrieve Information

Let's suppose a teacher creates a file called /RIPLEY/ with the following data on it:

```
4 ITEMS:
THE WASHINGTON MONUMENT
THE HONG KONG HILTON
THE TEMPLE OF KARNAK
THE HANGING GARDENS
```

(See pages 3-13 & 3-14 if you don't remember how to do this.) If the teacher (TD) wants other people to use the file, he must also define it as PUBLIC (see page 4-4).

He then writes the following program:

LISTING:

```
>10 PR."THIS IS A TRIVIAL EXAMPLE ØF A TUTØRIAL WHERE
YØU MAY USE @NBS TØ RETRIEVE DATA FRØM THE FILE /RIPLEY/.
DØ YØU HAVE YØUR FILE INSTRUCTIØN SHEET WITH YØU?"
>20 LET Y$=",YES,YUP,SURE,ØF CØURSE,AFFIRMATIVE,"
>30 INPUT R$
>40 IF IEQIV(R$,Y$,0) GØTØ 70
>50 PR."PLEASE LØGØUT AND ØBTAIN THE FILE INSTRUCTIØN
SHEET.  PRACTICE USING IT ØN A TERMINAL BEFØRE TRYING
THIS LESSØN."
>60 STØP
>70 PR."HERE IS YØUR FIRST QUESTIØN.....
    NAME AN ØBELISK FØUND IN AFRICA"
>80 INPUT R$
>90 IF ICØ(R$,"KARNAK",1) CALL RRIN,GØTØ 120
>100 PR."SØRRY - YØUR ANSWER ISN'T ØNE WE ANTICIPATED.
WE HAD THE 'TEMPLE ØF KARNAK' IN MIND"
>120 PR."LET'S TRY ANØTHER QUESTIØN
         ......ETC........."
>130 END
```

<u>AN INTERACTION:</u>

```
    >RUN /TRIVIAL/      (OR:  RUN 166TD /TRIVIAL/ )
    THIS IS A TRIVIAL EXAMPLE ØF A TUTØRIAL WHERE
    YØU MAY USE @NBS TØ RETRIEVE DATA FRØM THE FILE /RIPLEY/.
    DØ YØU HAVE YØUR FILE INSTRUCTIØN SHEET WITH YØU?
    ?YUP
    HERE IS YØUR FIRST QUESTIØN.....
         NAME AN ØBELISK FØUND IN AFRICA
    ?@NBS
    VER. AUG 12 17:20
    >ØPEN /RIPLEY/ FØR INPUT 2  (OR:  OPEN 166TD /RIPLEY/ FOR INPUT 2)
    >INPUT FRØM 2, A$(I) FØR I=1 TØ 5
    >PRINT A$(I) FØR I=1 TØ 5
    4 ITEMS:
    THE WASHINGTØN MØNUMENT
    THE HØNG KØNG HILTØN
    THE TEMPLE ØF KARNAK
    THE HANGING GARDENS
    >CLØSE 2
    >EXIT
    RESPØND TØ LAST INPUT REQUEST
    ?THE TEMPLE ØF KARNAK IS THE ANSWER
    VERY GØØD INDEED!
    LET'S TRY ANØTHER QUESTIØN
         ......ETC...........
```

<u>Example 2</u>

LISTING:

```
    >100 PR."SLIDE RULE DRILL:  ESTIMATING CUBE RØØTS"
    >110 REM WE ASSUME USE ØF A RANDØM GENERATØR AND A
    >120 REM LINEAR TRF TØ SUPPLY A VALUE FØR X IN LINE 130
    >130 LET X=37595.4
    >140 REM WE ASSUME THAT A SUBRØUTINE WØULD BE CALLED IN LINE
    >150 REM 160 FØR CALCULATING ANSWERS TØ MØRE GENERAL PRØBLEMS
    >160 LET C=X↑.333333
    >170 PR."PLEASE ESTIMATE THE CUBE RØØT ØF":X
    >180 INPUT R
    >190 IF R>C-C*.05 AND R<C+C*.05 CALL REIN, GØTØ 300
    >200 IF R>=C+C*.05 GØTØ 240
    >210 IF R<=C-C*.05 GØTØ 260
    >220 PR."DØ NØT UNDERSTAND - PLEASE REPEAT"
    >230 GØTØ 170
    >240 PR."NØ -- HINT:  YØUR ESTIMATE IS TØØ LARGE"
    >250 GØTØ 170
    >260 PR."NØ -- HINT:  YØUR ESTIMATE IS TØØ SMALL"
    >270 GØTØ 170
    >300 PR."LET'S TRY ANØTHER - IF YØU WISH TØ"
    >310 PR."STØP AT ANY TIME PRESS THE 'ESC' KEY"
    >320 GØTØ 130
    >330 END
```

AN INTERACTION:

```
>RUN /SLIDES/
SLIDE RULE DRILL:  ESTIMATING CUBE RØØTS
PLEASE ESTIMATE THE CUBE RØØT ØF 37595.4
?10
NØ -- HINT:  YØUR ESTIMATE IS TØØ SMALL
PLEASE ESTIMATE THE CUBE RØØT ØF 37595.4
?60
NØ -- HINT:  YØUR ESTIMATE IS TØØ LARGE
PLEASE ESTIMATE THE CUBE RØØT ØF 37595.4
?@NBS
VER. AUG 12 17:20
>5 INPUT A,B
>10 FØR I=A TØ B
>15 PRINT I;I*I*I
>20 NEXT I
>25 END
>RUN
?35 40
 35     42875
 36     46656
 37     50653
 38     54872
 39     59319
 40     64000

>RUN
?30 35
 30     27000
 31     29791
 32     32768
 33     35937
 34     39304
 35     42875

>PRINT 33.4*33.4*33.4
 37259.704
>PRINT 33.5*33.5*33.5
 37595.375
>EXIT
PLEASE RESPØND TØ LAST INPUT REQUEST
?33.5
CØRRECT
LET'S TRY ANØTHER - IF YØU WISH TØ
STØP AT ANY TIME PRESS THE 'ESC'KEY
PLEASE ESTIMATE THE CUBE RØØT ØF......
?←←ESC:         180
```

LIBRARY FUNCTIONS IN NBS (SEPT. 1970)

ABS(X)	Absolute value of X
INT(X)	Integer part of X
MOD(X,Y)	X modulus Y
MAX(X,...,Z)	Maximum of arguments
MIN(X,...,Z)	Minimum of arguments
SGN(X)	Sign of X
DIF(X,Y)	Positive difference ABS(X-Y)
IMAG(C)	Imaginary part of C
REAL(C)	Real part of C *not implemented
CMPLX(X,Y)	Complex number X,Y
CONJG(C)	Conjugate of C
EXP(X)	e to the X power
LOG(X)	Natural log of X
LGT,LOG10(X)	Log,base 10, of X
SIN(X)	Sine of X
COS(X)	Cosine of X
TANH(X)	Hyperbolic tangent of X
SQB,SQRT(X)	Square root of X
ATAN(X),ATAN(X,Y)	Arctangent of X, or of X/Y
ARCSIN(X)	Arcsine of X
ARCCOS(X)	Arccosine of X
CINH(X)	Hyperbolic sine of X
COSH(X)	Hyperbolic cosine of X
FIX(X)	Integer mode form of X
FLOAT(I)	Floating point mode form of I
SNGL(D)	Single precision mode form of D
NUM(X)	Random numbers from 1 to X
LSH(I,J)	Binary left shift I for J positions
RSH(I,J)	Binary right shift I for J positions
WAIT(X)	Halt execution for X seconds
POS(I)	Print head position of file I
TAN(X)	Tangent of X

STRING FUNCTIONS

(S = String Argument, N = Numeric Argument)

INDEX(S_1,S_2)	Position of S_2 within S_1; e.g., INDEX("ABC","C") = 3.
LEFT(S,N)	Substring of S; N characters long starting from left.
LENGTH(S)	Number of characters in S.
RIGHT(S,N)	Substring of S; N characters long, starting at right.
SPACE(N)	String N spaces long.
STR(N)	String of the characters comprising N. STR(4) = "4".
SUBSTR(S_1N_2)	Substring of S from N_1th character to the end of S.
SUBSTR(S,N_1,N_2)	Substring of S: N_2 characters long, starting at N_1th character.
VAL(S)	Numeric value of S, where S must be a numeric string; e.g. VAL("+8") = 8.

Example of using the XTRAN function FACTRL (see the XTRAN library manual for details on over 100 such functions which are also available in NBS).

```
10 INTEGER N,I,K
15 LET N=8
20 LET K=N-3
21 LET X=FACTRL(N,F,B,I)/FACTRL(K,F,B,I)
22 PRINT N;K;X;FACTRL(N,F,B,I);FACTRL(K,F,B,I)
25 END
>RUN
 8     5      336    40320     120
```

The Random Generators RRAND And NUM

The function RRAND(X) produces a random number between 0 and X, where X is any number containing a decimal point.

EXAMPLE:

PRINT RRAND(50.0) causes a different number to be printed each time it is executed, of the form:

```
27.3469
 0.8356
18.2634
 5.7082
49.0236
```

INT(RRAND(50.0)) would produce the integer parts of these numbers:

```
27
 0
18
 5
49
```

NUM(50) would produce numbers like 41, 1, 28, 50, 13, etc., i.e. 1<=NUM(50)<=50.

Example of Use of RRAND in a Drill Program (created by user XX)

```
>10 PR. "ADDITIØN DRILL    (LF)
>WHAT IS YØUR NAME PLEASE";
>20 INPUT N$
>40 LET X=INT(RRAND(100.0))
>50 LET Y=INT(RRAND(100.0))
>60 IF PASS>5 THEN 110
>70 PR. "WHAT IS":X:"+":Y;
>80 INPUT R
>90 IF ABS(1-(X+Y)/R)<.03 CALL REIN, GØTØ 40
>100 PR. "NØ, THE ANSWER IS":X+Y GØTØ 40
>110 PR. "THAT'S ALL FØR NØW ":N$
>115 PR. "DØN'T FØRGET TØ EXIT AND LØGØUT"
>120 END          >EXIT
>SAVE             -DEF /DRILL/ AS PUB
>ØN: /DRILL/      -LOGOUT
```

Alternate Coding

```
40 LET X=NUM(100)
50 LET Y=NUM(100)
```

```
> RUN  /DRILL/
SUBPRØGRAMS REQUIRED
FILE:  166CL /RB/

ADDITIØN DRILL
WHAT IS YØUR NAME PLEASE? HØRATIØ
WHAT IS 65+ 94?23
NØ, THE ANSWER IS 159
WHAT IS 89+ 47?136
Ø.K.
WHAT IS 35+ 33?68
THAT'S RIGHT
WHAT IS 42+ 31?34
NØ, THE ANSWER IS
73
WHAT IS 79+ 91?170
CØRRECT
THAT'S ALL FØR NØW HØRATIØ
DØN'T FØRGET TØ EXIT AND LØGØUT
```

A user other than XX would type:
    >RUN 166XX /DRILL/
where XX is creator's ID.

Answering 166CL /RB/ is necessary because of use of RRAND in main program. (This is a temporary situation because RRAND is a newly developed "experimental" function.)

NUM doesn't request any SUBPROGRAMS.

NUM always returns an integer, so you don't have to say INT(NUM(50)).