MATH OBJECTS

The following words and phrases are called MATH objects:

step __	*all steps*	*part* __	*all parts*
form __	*all forms*	*formula* __	*all formulas*
all (everything)	*all values* (values of all stored letters)		

These objects may be deleted, typed, or stored. (See *Delete*, *Type*, *Store*.) The dashes may be filled with a specific number or with an expression that computes a number or formula identifier.

users: This is the number of MATH consoles in use, not just those computing (not available at all locations).

time: This is actual 24-hour clock time at the computer center.

Note that *users* and *time* appear only in *Type* commands.

```
Type users.
      users:  15
Type time.
       time:  11:33
```

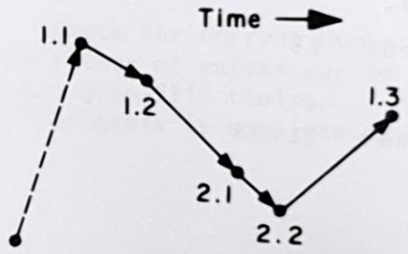
DO

This is the primary verb that initiates a computation. Upon completion of what the *Do* has ordered, control is returned to the point just after that *Do*. If *Do* is given as a direct command, then control is returned to you. If *Do* is stored, control returns to the step just after the *Do* step in the same part, and computation continues. *Do* can order either a single step or a part to be executed. Only the *Do* command can be modified by a *for* phrase. (See page 35.)

Do is interpreted only once. At that time, all values specified by expressions in the *for* phrase are calculated using current letter values and are saved. Hence, you cannot change the execution range of *Do* by giving new values to the letters in a *for* phrase range, as in FORTRAN. See *Done/Quit* (Page 27) for escape methods.

Any *if* clauses modifying *Do* commands are interpreted only when the *Do* is first met, so that the *if* condition does not apply during the subsequent repetitions over the set of values specified by the *for* phrase.

```
1.1 Type "step 1.1".
1.2 Do part 2.
1.3 Type "step 1.3".

2.1 Type "step 2.1".
2.2 Type "step 2.2".

Do part 1.

step 1.1
step 2.1
step 2.2
step 1.3

Do step 2.1.

step 2.1

Do part 2.

step 2.1
step 2.2
```



In this diagram MATH's attention is shown flowing from left-most dot (opposite Do part 1.), to the next dot (opposite step 1.1), to the next dot (opposite step 1.2), and so forth.

```
1.1 Type x.
1.2 Set x=10.
1.3 Type x*2.
1.4 Line.

Do part 1 for x=1(1)4.
          x =          1
        x*2 =        100

          x =          2
        x*2 =        100

          x =          3
        x*2 =        100

          x =          4
        x*2 =        100
```

Note that values of x specified by the for clause are retained and used in successive iterations even though x has a new value set at step 1.2.

```
1.1 Type x.

Do part 1 for x=1(1)3 if x≤2.
x = ???
Set x=2.
Do part 1 for x=1(1)3 if x≤2.
          x =          1
          x =          2
          x =          3

1.2 Set x=100.
Do part 1 for x=1(1)3.
          x =          1
          x =          2
          x =          3

Do part A for A=1,2,3.
A = ???
```

The if clause evaluated before Do execution.

The if clause evaluated only once-- before Do execution.

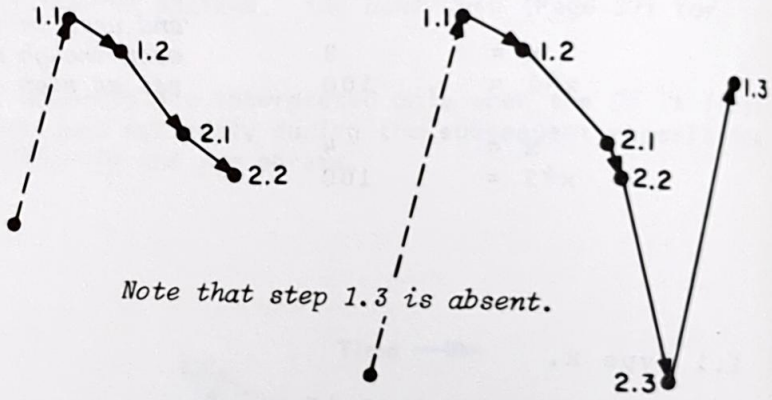
Iteration values saved by Do execution.

Do command interpreted only once.

TO

This is a stored command only. It will move computation to the step or part indicated, but will not return control to the step following the *To* step. Computation continues as directed by the program subsequent to the point reached by the *To* command. *To* can send you freely backward and forward in the program.

```
1.1 Type "step 1.1".
1.2 To part 2.
1.3 Type "step 1.3".

2.1 Type "step 2.1".
2.2 Type "step 2.2".

Do part 1.
step 1.1
step 2.1
step 2.2

2.3 To step 1.3.
Do part 1.
step 1.1
step 2.1
step 2.2
step 1.3
```



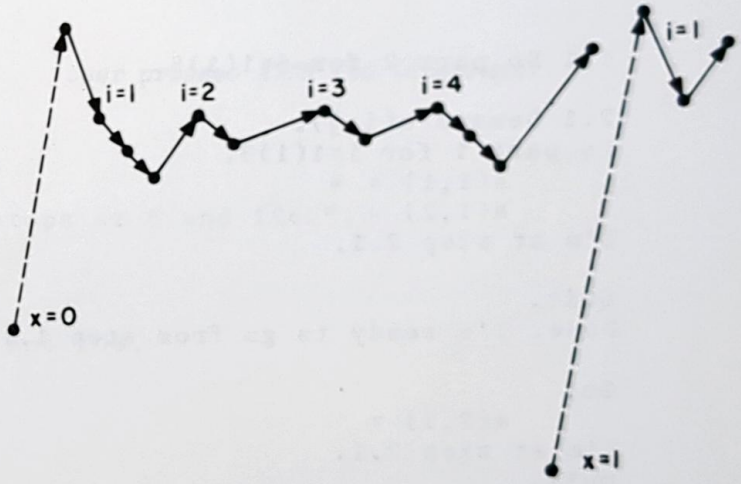
Note that step 1.3 is absent.

Compare with *Do* *example page 24.*

DONE/QUIT

The *Do* command will normally be executed for the successive values in the set given by the *for* phrase. If a *Done* statement is reached in this execution, nothing specified farther on in the program will be done for the currect value being processed. However, the remaining values will be processed. If a *Quit* statement is reached, neither the current value nor any of the remaining values will be further processed. *Done* and *Quit* may have an *if* clause. In this case, the above behavior with respect to the current value and subsequent values is subject to that condition. In MATH, each part is provided automatically with an implied *Done* command as its last step. That is, as each current value being processed reaches this step, nothing further is done with it. *Quit* but not *Done*, may be used directly.

```
1.1 Do part 2 for i=1(1)4.
1.2 Type i.

2.1 Quit if x=1.
2.2 Done if i=2 or i=3.
2.3 Type i in form 1.

Form 1:
*** ***
    _

Do part 1 for x=0.
***1***
***4***
            i =          4

Do part 1 for x=1.
            i =          1
```



```
1.1 Set i=i+1.
1.2 Quit if i=5.
1.3 Done if i>3.
1.4 Type i.

i=0
Do part 1, 10 times.
            i =          1
            i =          2
            i =          3

Type i.
            i =          5
```

Quit and Done apply also to "times" iterations.

```
1.1 Do part 2 for j=1(1)5.

2.1 Demand a(i,j).
Do part 1 for i=1(1)3.
      a(1,1) = 4
      a(1,2) =
I'm at step 2.1.

Quit.
Done. I'm ready to go from step 1.1.

Go.
      a(2,1) =
I'm at step 2.1.
Quit.
Done. I'm ready to go from step 1.1.

Quit.

Quit.
I have nothing to do.
```

RETURN pressed, which interrupts.

Next execution point after 1.1 (the next value for j).

RETURN pressed.

No more action for the Do in 1.1.

No more action for the direct Do, so control returns to the user.

There is now nothing to quit.

STOP/CANCEL/END

Stop is programmed interrupt (stored only) that returns control to you. This permits some operator action. A subsequent *Go* command will resume computation at the step following the *Stop* step.

Cancel may only be used directly. The machine's knowledge of current position in the computation is erased, without destroying either values computed up to that point or the program itself. A *Do* command is needed to restart the computation. A new *Do* command alone would achieve the same erasure and would start a new computation. But if you are space limited (See page 39), *Cancel* will give you back maneuver space and let you restart after corrections and insertion of new values. It is a rarely used verb.

End is used to terminate a MATH session. It may be given directly or stored. DOS/MATH termination differs from CTS/MATH termination.

```
1.1 Set f(i)=i*2.
Do part 1 for i=-100(1)100.
I'm at step 1.1.                    User presses ATTN (to interrupt).
Cancel.
Go.
I have nothing to do.

1.1 Type "Please set margin stops at 6 and 106.".
1.2 Stop.
1.3 Type "Thank you.".
Do part 1.
Please set margin stops at 6 and 106.
Stopped by step 1.2.
Go.
Thank you.

End.
Ended at 15:37.                     Signed off DOS/MATH.

1.1 End.
Do part 1.

SELECTION: end                      Off MATH but not off CTS.
OFF AT 15:42                        Signed off CTS.
```

GO

This command continues computation in progress after a programmed or manual interruption, as if the interruption had not occurred. Interruptions may occur because of (1) an error message, (2) pressing ATTN button, or (3) a programmed *Stop* verb. During the interruption, program or values may be modified as desired. If no computation was in progress (although you may have thought so) *Go* will produce the *I have nothing to do.* message. *Go* can only be used after MATH says he is somewhere.

```
1.1 Type x.
1.2 Stop.
1.3 Type x*3.

Do part 1.
Error at step 1.1:  x = ???
x=2
Go.
          x =          2
Stopped by step 1.2.

1.25 To step 1.25.                    A meaningless loop.
Go.
I'm at step 1.25.                     User presses ATTN (to interrupt).

Delete step 1.25.
Go.
        x*3 =          8
Go.
I have nothing to do.
```

DELETE

The verb *Delete* followed by one or more MATH objects, separated by commas, may be used directly or indirectly, and will cause those objects to disappear. Any letter or mixed lists of objects and letters can be erased similarly. *Delete* may also be used to erase an item stored in the file. (See page 46.)

```
Type all.

1.1 Delete part 1.
1.2 Set r=fp(291•r).
1.3 Type "I'm step 1.3".

2.1 Set g=t(100•ip(r)).

Form 1:
____.__      _____      .........

        t(k):   s•f(k)/r(k)

           s =          3.7901

      a(1,1) =          4

Delete step 1.2, part 2, form 1,s,a, formula t.
Type all.

1.1 Delete part 1.
1.3 Type "I'm step 1.3".
Do part 1.
Type all.
```

Steps organized into parts.

Form.

Formula.

Values.

May be used indirectly.

Blank lines follow to indicate nothing stored.

SET

This is a replacement operation that assigns a new value -- given on the right of the equals sign -- to the identifier named on the left (a letter or an indexed letter). *Set* can be direct or indirect. In the direct mode, the word *Set* and the terminal period may be omitted. *Set* computes the right-hand expression just once and stores the value at a given place, named on the left-hand side. For indexed letters, values corresponding to each integral index are correspondingly stored in places named by the left-hand side. Note that indices can play the role of subscripts. (See page 33 for the important distinction between *Let* and *Set*.) Note that, say, the order *Type A.* will produce all values of *A* if *A* is an indexed letter.

```
Set x=x+1.
x = ???
x=3
x=x+1
Type x.
           x =          4
Set a(-1,2)=3.
a(1,-2)=1
Type a.
     a(-1,2) =          3
     a(1,-2) =          1
Set a(1,2,3,-5)=4.
Type a.
 a(1,2,3,-5) =          4

1.1 x=5
1.2 Type x.
Do part 1.
Error at step 1.1:  Eh?
```

} *The word Set and the period are optional in direct commands.*

} *Array of new dimension replaces old.*

The word Set must be used in stored commands.

LET

This command is used to define functions, called "formulas" in MATH, of at most 10 variables (called "parameters" in error messages). *Let* defines a rule for computation that is invoked each time the function is referenced. Unlike *Set*, the value computed is not stored but is used transiently. The "parameters" are dummies--they do not affect the values of stored letters with the same names. (The command *Set f(x)=3.* requires x to be an integer, $|x| \leq 250$, since we are naming a storage place, by the letter and the index.)

Let is normally used directly since the defined function is usually employed in the program as a whole. It is used indirectly when in the course of the program the function is to be redefined (thus erasing the previous definition).

Let can be used to abbreviate an expression (e.g., *Let s=sqrt(x*2+y*2).*). In this case when s is needed, current stored values of x and y will be used. If there is no stored x, MATH will say, *Error in formula s: x=???.* When you type *x=1*, this becomes a stored value. On the other hand, *Let s(x,y)= sqrt(x*2+y*2).* permits calling via *s(1,2).* As explained above, x,y are now dummies, and stored values are not affected.

```
Let f(a,b,c,d,e,f,g,h,i,j,k,l)=a+b+c.
Please limit number of parameters to 10.

Set S(x,y)=sqrt(x*2+y*2).
x = ???

Let S(x,y)=sqrt(x*2+y*2).

Type S(1,2).
      S(1,2) =          2.23606797

Type S(3+45,4.5/17).
S(3+45,4.5/17) =        48.0144153

Let D(f,x)=[f(x+d)-f(x)]/d.
d=.00001


Type D(sin,0), D(cos,0).
    D(sin,0) =          1.
    D(cos,0) =          0
```

Any expression may be used for the argument.

Derivative of function f at x. Note that parameters may name formulas, functions, or values.

Derivative of sine and cosine at zero.

```
Let s = sqrt(x*2+y*2).
Type formula s.
          s:  sqrt(x*2+y*2)
```

Note colon.

```
Type s.
Error in formula s:  x = ???
x=3
y=7
Type s.
          s =          7.61577311
Let S(x,y) = sqrt(x*2+y*2).
Type S.
       S(x,y):  sqrt(x*2+y*2)
Type S(.1, 10.05).
S(.1, 10.05) =          10.0504975
Type x,y.
           x =           3
           y =           7
Let I(x) = 1≤x<2.
Type I(3), I(1.05).
        I(3) =       false
     I(1.05) =       true
```

Equivalent to Type formula S.

Note that values of x, y are unaffected by Type S.

```
Let B(i) = [ i=0:L; B(i-1)<P:0; B(i-1)•(1+r)-P ]
L=1000
P=100
r=.005

1.1 Type I,B(I) in form 1.
Form 1:

___  _____.__
Do part 1 for I=0(1)11.
  0     1000.00
  1      905.00
  2      809.53
  3      713.57
  4      617.14
  5      520.23
  6      422.83
  7      324.94
  8      226.57
  9      127.70
 10       28.34
 11         .00
```

See page 56 for Conditional Functions.

Formula B calculates, in a recursive way, the balance of loan L which has payment P and interest rate r.

FOR/TIMES

The word *for* gives the set of values (See page 18) for which a *Do* order is to be executed. It can be used only with the verb *Do*. The set of values in the *for* phrase is stored, as if a *Set* command had been used for each value successively.

An additional modifier for the *Do* command only is *times*:

Do step __, n times.
Do part __, n times.

Note the comma, which makes it possible to avoid an unpleasant conjunction of numbers, and the integer n, which, as always in MATH, may arise from a computation.

```
1.1 Type i.
Do part 1 for i=2(3)10(5)20,100,.003.
            i =          2
            i =          5
            i =          8
            i =         10
            i =         15
            i =         20
            i =        100
            i =           .003
Do part 1 for i=0(1/3)1.
            i =          0
            i =           .333333333
            i =           .666666666
            i =           .999999999
            i =          1
Do part 1, 3 times.
            i =          1
            i =          1
            i =          1
Do part 1, .56 times.
Number-of-times must be integer and ≥ 0.
```

Expressions for numbers could have been used.

Note that range end point is always hit exactly.

Last stored value of i is 1.

IF

This word may be used to qualify any command. It specifies the conditions under which the imperative will be executed. These conditions may contain logical and algebraic expressions. In examining a command with an *if* clause, MATH first evaluates the condition in that clause. When the condition is not met, the command is not processed in any way. Thus the command could be in error and the error not detected until the condition is finally met. (See also examples under *Do*, page 24.)

```
x=1
y=4
Type x if x=1 and y<34.
           x =          1
Type x if x=3 and not (x+y>34) or y≠17.
           x =          1
Set y=35 if x<40.
Type y.
           y =         35
1.1 Type 1≤x<3<y≤a.                    Compound condition.
Do step 1.1 for a=15,100.
   1≤x<3<y≤a =       false
   1≤x<3<y≤a =       true
Do step 1.1 if y>10*6.                 Condition fails, so command is
                                       not interpreted
Type garbage if y=0.                   Condition fails, so command is
                                       not interpreted.
```

TYPE

Type may be direct or stored. Any MATH object (Page 23) can follow *Type*, as may any letter or expression, or any list of objects, letters, and expressions separated by commas. The command causes appropriate typeout on successive lines.

If double quotes are placed around any expression or text, it will be typed out verbatim without quotes. (Single quotes are not part of MATH punctuation.) The underscore will cause a corresponding blank line in the typeout.

The MATH word *in* appears only in the command *Type__in form__*. (See page 42).

```
x=3
Let y=x*2.
1.1 Type x,y in form 3.
Form 3:
x=__    y=__      appear in form 3.
Type x, step 1.1, form 3, formula y.
              x =          3
1.1 Type x,y in form 3.
x=__    y=__      appear in form 3.
                y:   x*2

Type "MATH".
MATH
Type ""MATH"MATH"".
"MATH"MATH"

Type x,_,y.
              x =          3

              y =          9
```

TIMER

This measures real time in minutes and hundredths of minutes since the instant of console activation or since the last (unique) command *Reset timer.*. The word *timer* may be used in computational expressions (hence minutes and hundredths). (See pages 61-63 for some uses of *timer*.)

```
Type timer.
       timer =        13.1

Reset timer.

Type timer.
       timer =          .12

Set x=timer.
Type timer-x.
     timer-x =          .18

Set timer =5.                          Value of timer may not be set.
Eh?
```

SIZE

An individual MATH user has at his disposal a maximum of about 1500 cells (storage areas), even if he is the sole user. Some locations may restrict you to fewer cells. A cell will hold one numerical value (but some overhead cells are required for array rows and columns) or nine characters of a program step, form, or formula. The response to the command *Type size* is the number of cells currently in use. Since *size* may be used in computational expressions, the user may arrange his program to alert him to an imminent out-of-space error message. During the execution of a *Do* command, cells are also required to record the progress of that execution. Similarly, cells are needed for the evaluation of each formula. (Note that chaining (pages 47 and 65) may be used for large problems.)

```
Type size.
        size =          0
1.1 Set a(i,j)=i+j.

2.1 Do part 1 for j=-250(1)250.

Do part 2 for i=-250(1)250.
I'm at step 1.1. I ran out of space.

Type size.
        size =       1352

Cancel.
Type size.
        size =       1336

Delete a.
Type size.
        size =         11
```

Cancel retrieves 16 cells that were used for the execution records of Do.

DEMAND

This is a stored command only. The order allows the program to call for values to be typed in by the user, thus minimizing the amount of typing to be done, systematizing inputs, and avoiding failures to input needed data. MATH types out the requested letter, which may be indexed, and an equals sign. The user types the desired value or expression to the right. If the user terminates his input line with an asterisk, the line is ignored and the *Demand* request is made again. If the user's response is improperly formed, MATH returns an appropriate error message and makes the request again. If the user hits RETURN only, MATH considers it to be an interrupt and returns control to the user with an appropriate message.

If the *Demand* command is followed by the word *as* and a string of text in quotes, MATH will type the quoted text followed by an equals sign to identify the requested value.

```
1.1 Demand a(i).

Do part 1 for i=1(1)5.
        a(1) = 10
        a(2) = 3.5+24
        a(3) = sin(4.789)
        a(4) = a(2)+3*a(3)
        a(5) = 28.75*
        a(5) = atemp
Eh?
        a(5) =
I'm at step 1.1.
a(1)=100


Go.
        a(5) = 15

1.1 Demand E as "Voltage maximum".

Do part 1.
Voltage maximum = 135.5
```

Any expression may be given.

Input ignored and request repeated.

Carrier return only is an interrupt.

SPARSE

This term is used only in the command *Let A be sparse.*, where *A* is the name of an array with up to 10 indices. When *A* is declared sparse and values have been given to some of the array's elements, MATH considers all other elements to be zero. If a new *A* is entered with a different number of indices, the old *A* is erased and the new *A* must be redeclared sparse. Inputting elements is a *Set* operation, and hence only these elements are stored. The command corresponds to *Let* in that nonstored elements are "computed" to be zero. Hence *sparse* saves storage space and inputting effort.

```
a(2,3,4)=7
Type a(2,3,5).
a(2,3,5) = ???
Let a be sparse.
Type a(2,3,5).
    a(2,3,5) =          0
Type a.
    a(2,3,4) =          7
a is sparse

a(2,3)=23
Type a.
       a(2,3) =         23
Type a(1,2).
a(1,2) = ???
Let a be sparse.
Type a(1,2).
       a(1,2) =          0
```

Because the command requested all values of a.

New array replaces old.

FORM

A *form* describes how you want your output to appear. In the body of the program, the order is: *Type* "list" *in form n.*. The "list" is usually a set of letters or expressions, separated by commas, whose values will be typed out following the form's layout. The form number n is an integer ($1 \le n < 10^9$) but may be an expression whose value will be computed.

Forms are typed directly. The first line is simply *Form n:*. The second line is a set of fields and/or internal information, spaced as desired.

A field is a string of underscores, perhaps with a decimal point, or a string of periods. Values will fill these fields in the order specified, rounded as necessary. All other characters in the form are typed out exactly as they were typed in (including asterisk).

The number of underscores to the right of a decimal point indicates the desired roundoff. The number of underscores to the left must be adequate to hold the expected integer part of computed values, including a minus sign. It is usually simplest to be generous. A field of periods will be filled by the value in scientific notation. The minimum field of 7 periods will give just two significant digits. The first period is reserved for the sign of the value, and the third from the end is reserved for the sign of the exponent.

Forms without fields are useful to output headings, messages, and the like. In this case, the second line of the form definition is the desired text and the command is simply *Type form n.*.

```
Form 1:
_____    ___._    .......

x=12.356
Type x,x,x in form 1.
  12      12.4      1.2 01

Type x,x,x,x in form 1.
I have too many values for the form.

Type x,x in form 1.
  12      12.4

x=1234.5678
Type x,x,x in form 1.
I can't express value in your form.

Form 4:
.........................
Type -2/3 in form 4.
-6.666666700000000000-01
```

Note rounding into form.

Fewer values than fields OK.

Note rounding to 9 significant digits.

```
Form 1:
___._ amps.   ___._ volts.
Form 2:
___._ amps.   ..... volts (voltage ≥ 1000).
1.1 Type I, I•R in form (I•R<1000:1; 2 ).
R=91

Do step 1.1 for I = 8.735(1.05)12.9.
  8.7 amps.   794.9 volts.
  9.8 amps.   890.4 volts.
 10.8 amps.   986.0 volts.
 11.9 amps.    1.1503 volts (voltage ≥ 1000).
 12.9 amps.    1.2 03 volts (voltage ≥ 1000).
```

Form number is computed (see page 56).

Form 1 used.

Form 2 used.

```
1.1 Type form 2.
1.2 Line.
1.3 Do part 2 for x=10(10)60.
2.1 Type sin(x), log(x), exp(x) in form 1.
Form 1:
___._ ___._ .....
Form 2:
SIN(X)    LOG(X)    EXP(X)

Do part 1.
SIN(X)    LOG(X)    EXP(X)

  -.54      2.30    2.20 04
   .91      3.00    4.85 08
  -.99      3.40    1.07 13
   .75      3.69    2.35 17
  -.26      3.91    5.18 21
  -.30      4.09    1.14 26
```

See page 44 for Line.

Form with literal information only.

Line. advances paper one line, leaving a blank. (See also page 37.) *Page*. advances paper to start a new page. *Head*. begins a new page on the next line. These commands can be used to improve output format. The format can be programmed by appending *if* clauses to *Line* and *Page*. The symbol *$* has as value the current line number on the page (1 to 54--the heading line is for administrative use only). The symbol *$* is often used in *if* clauses for format control purposes.

```
 Head.
9:38     4-16-68   JRD     (7)
```

```
 Page.
```

```
9:39     4-16-68   JRD     (8)
```

```
 Type $.
 Line.          $ =          2
 Type $.
                $ =          6
1.1 Line if fp($/3)=0.
1.2 Type $.
Do part 1, 4 times.
                $ =         10
                $ =         11
                                    Line command executed.
                $ =         13
                $ =         14
```

SUMMARY

The contents of your file may be reviewed by the command *Type file summary..* In this list, the heading SPACE refers to the number of disk spaces (See page 21.) required to store the item. DATE refers to the time that the item was stored. Your private file number is included in the final line of the summary. The contents of the general library may be listed by the command *Type library summary..*

```
Type library summary.

NAME                    DATE        SPACE
adding-machine          2-10-68         1
curve-plotter           4-16-68         5
gauss-integrate         3-15-68         1
loan-amortizer          3-25-68         1
math-lesson(1)          3-19-68         3
  ...                     ...         ...
math-lesson(4)          3-19-68         3
math-quiz               4-01-68         2
matrix-inverter         3-15-68         1
polynomial-fit          4-02-68         4
submarine-game          2-23-68         6
TOTAL SPACE IN LIBRARY:                33

Type file summary.

NAME                    DATE        SPACE
cosine-test             4-08-68         1
data-f                  3-12-68         1
data-g                  3-12-68         1
example(20)             3-16-68         1
gauss-integrate         3-14-68         1
integration             3-11-68         1
nim-game                2-10-68         2
program(2)              3-12-68         3
root-finder             3-14-68         1
TOTAL SPACE IN FILE 148:               12
```

STORE/DELETE/GET

All file commands may be direct or stored.

The commands to enter material in your file are *Store* "list" *as* "name" or *Store* "list" *replacing* "name". The "list" consists of MATH objects (See page 23.) and/or letters identifying values, arrays, or formulas. You choose a "name" of from four to fifteen letters and/or hyphens. The "name" may include an integer-valued index in the range [-250,250]. In the case of *Store ... replacing ...*, MATH assigns the specified name to the newly stored item and then deletes the old item of that name.

To erase an item from your file the command is *Delete* "name". (See also page 31.)

You can get a copy of an item from your file, from the library or from another file. Since it is a copy, you can manipulate or modify the item as you wish. The commands are *Get* "name"., *Get* "name" *from library*., and *Get* "name" *from file* "number". (See page 45 for "number".) After *Get*, you can use *Type* commands to examine the item.

```
1.1 Page.
1.2 Type form 2.
1.3 Do part 2 for i=1(1)10.
2.3 Type b,f(b),log(f(b)) in form 1.
Store all as example(16).                    Copies all steps, forms, formulas,
Done.                                        and values into the file.
Delete all.
Get example(16).                             Copies from file to computer.
Done.
Delete step 1.1.
1.2 Do part 10 if $≠1.                       Program is modified.
Store all replacing example(16).             Replace old version
Done.                                        with updated one.
Type all.

1.2 Do part 10 if $≠1.
1.3 Do part 2 for i=1(1)10.

2.3 Type b,f(b),log(f(b)) in form 1.


Delete example(16).                          Erases program from file.
Done.

Delete all.
Get matrix-inverter from library.            Copies from library to computer.
Done.

Do part 1.                                   Library programs start with part 1.
```

CHAINING

It is also possible to "chain" subprograms or data held in separate items and needed for a large problem. You will not run out of space or mix part numbers provided sufficient care is taken in programming the chain's housekeeping.
(See page 65.)

FILE WARNING

Getting file material is equivalent to typing in the filed material from a console during the session. Hence step numbers, values, etc., from the item replace similarly numbered or named ones previously typed in. It is good practice to *Delete all.* before recalling an item, Otherwise, an appalling confussion of steps can result. Manipulations of files and items can be done indirectly by program, and this is most useful.

File access time may vary from a few seconds to minutes, depending essentially on the number of users queuing for file access. An impatient interrupt will send you back to the end of the queue.

```
Type all.

1.05 Set r=fp(r•291).

2.45 Do part 17 for x=.003, .07(.03).15.

        b(1) =          3
        b(3) =          7

Get example(20).
Done.
Type all.

1.05 Set r=fp(r•291).
1.2 Do part 10 if $≠1.
1.3 Do part 2 for i=1(1)10.

2.2 Set b=-a.
2.3 Type b,f(b),log(f(b)),I(f(b)) in form 1.
2.45 Do part 17 for x=.003, .07(.03).15.

        b(1) =          3
        b(3) =          7
```

Recalling from the files results in a mixture of that on hand and that coming from the files.

PARENTHETIC COMPUTATION

Sometimes in the course of a calculation (say, at an interrupt point or an error message) it is convenient to perform a second calculation (perhaps to identify an error) without disturbing the one in progress. This may be accomplished by a parenthetic *Do*--a *Do* command enclosed in parentheses. The process may be carried out indefinitely (subject to storage limits). Parenthetic *Cancel.* cancels the current subexecution, and the primary execution may then be resumed. Note that new values so computed for letters will remain after the parenthetic computation.

```
1.1 Demand a(i,j).
2.1 Do part 1 for i=1(1)5.
Do part 2 for j=1(1)5.
      a(1,1) = 1
      a(2,1) = 2
      a(3,1) = 3
      a(4,1) = 4
      a(5,1) =                          Carrier return interrupts Demand.
I'm at step 1.1.
(Do part 2 for j=1.)
      a(1,1) = 2                        Parenthetic Do execution used to
      a(2,1) = 3                        reenter some values.
      a(3,1) = 4
      a(4,1) =
I'm at step 1.1.
(Cancel.)                               Parenthetic execution is canceled.
Done. I'm ready to go at step 1.1.
Type i.
           i =          4
i=5
Go.                                     Correct value of i set and suspended
      a(5,1) = 5                        execution resumed.
      a(1,2) = 4+2
      a(2,2) = 3+2
      a(3,2) =
```