III. GENERAL POINTS

RULES OF FORM

MATH(JOSS) commands are written as normal English imperative sentences: capitalization of first word, proper spelling and word spacing, final period. There can be only one command per line, which MATH "considers" only after you strike the carrier RETURN button. A command is executed completely or not at all. In typing mathematical expressions, spaces may be used freely, except within numbers and between the name of any function, formula, or array and the left parenthesis.

```
Type 2+10.
        2+10 =          12
type 2+2.
```
Initial capital omitted.

```
Eh?
Type 2+2
```
Final period missing.

```
Eh?
Type2+2.
```
No space.

```
Eh?
Tpye 2+2.
```
Misspelling.

```
Eh?
Type (2+2·3.
```
Unpaired parenthesis.

```
Eh?
Type 2.2.3.
```
Period instead of multiply.

```
Eh?
Type sin (4.7).
```
Space following function name.

```
Eh?
Type 2+l0.
```
"Ell" instead of "one."

```
Eh?
Type 2+10.
```
"Oh" instead of "zero."

```
Eh?
Type 2+1 0.
```
Space within number.

```
Eh?
```

PRECEDENCE RULES

MATH follows conventional rules in determining the order of operations: exponentiation first, followed by multiplication and division, and then by addition and subtraction. Parentheses, of course, can alter precedence rules, and in fact, in complicated cases, the best advice is to be liberal with parentheses and do a sample side calculation on MATH, using numbers, to see if you are getting what you want. Many good programs have foundered for lack of attention to precedence.

```
Type -2*2.
      -2*2 =        -4
Type 1+1/2.
      1+1/2 =        1.5
Type (1+1)/2.
    (1+1)/2 =        1
Type 1/2+1.
      1/2+1 =        1.5
Type 1/3*3.
      1/3*3 =         .999999999
Type 2*3*2.
      2*3*2 =       64
Type 2*(3*2).
    2*(3*2) =      512
Type 2*3*4+5.
    2*3*4+5 =       37
```

-2^2

$1+(1/2)$

$\frac{1+1}{2}$

$(1/2)+1$

$(1/3)\cdot 3$

$(2^3)^2$

2^{3^2}

$(2^3\cdot 4)+5$

EDITING

Line editing may be done prior to striking carrier return, even if the final period has been typed. You may backspace, overtype, and use # to blank out unwanted characters. Backspacing or forward spacing does not blank out characters. An asterisk at the beginning or end of a line will tell MATH upon carrier return to ignore the entire line. The MATH line may have 100 characters, but it is usually wise to keep lines short, say by abbreviations, to avoid extensive retyping of lines later found to be in error or to require modification.

```
Type 2.08 + 52.72.                    Backspace--overstrike corrections.
2.07 + 52.71 =        54.78

Type#2+2.                             # used to blank out.
          2+2 =          4

Type 2+2 *                            * at end causes line to be ignored.

Type 2+2.                             # indicates transmission error.
Type 2#+2.
Sorry. Say again:
```

VALUES

Decimal or logical values may be assigned to any of the 52 uppercase and lowercase letters that MATH uses for identifiers. Values may be organized into arrays by using indexed letters (page 19). Letters may also be assigned arbitrary expressions called *formulas* by MATH (see *Let*, page 33), but a letter may name only one value, array, or formula at a time, a new definition replacing the old.

MATH carries decimal values in the range $\pm 10^{-63}$ to $\pm 9.99999999 \cdot 10^{63}$ with nine digits of significance. Values less than 10^{-63} are replaced by zero.

```
Set x=3.
Type x•x, x*x.
        x•x =          9
        x*x =         27

Type volts.                          Single letter identifiers only.
Eh?

a(1)=4
a(2)=7                               Arrays may have 1 to 10 indices.
Type a.
       a(1) =          4
       a(2) =          7
Type 10*(-100).
   10*(-100) =         0
Type 10*100.
I have an overflow.

b(1)=true
b(2)=10≤x<100.
Type b.
       b(1) =       true             Logical values too.
       b(2) =       false
```

DIRECT AND STORED COMMANDS

MATH commands are given in one of two ways. A direct command begins with a MATH verb and is executed immediately (directly) upon carrier return. If the command begins with a number - the step number - it will be stored for later execution. This is the stored command, also called the "indirect" command. A sequence of numbered commands forms a stored program in MATH, to be executed when desired by a direct command. Steps are stored in the sequence given by increasing step numbers (e.g., 1.05, 1.1, 1.19, 1.3), which need not be consecutive. MATH reorders your input according to these step numbers, regardless of the order of inputting. (Note that 1.19 precedes 1.3)

A sequence of steps with the same number to the left of the decimal is called a *part*, identified by that integral number (e.g., *part 1*). The first step of a part may be labeled with an integer, but then it will not be possible subsequently to insert a new step before it. It is good practice to begin with, say 1.1. A stored step will be deleted and replaced by a new input step with the same number. This automatic replacement is general in MATH, applying to letters and definitions.

```
1.1 Type 2+2.                    Stored (indirect) command.
Do step 1.1.                     Direct command.
          2+2 =          4

1.2 Type 3•4.
1.05 Type 5/6.

Type all.

1.05 Type 5/6.                   Steps ordered by step number.
1.1 Type 2+2.
1.2 Type 3•4.


2.1 Type 4*5.
1.2 Type 3•4•5•6.                Replaces previous step 1.2.

Type part 1.
1.05 Type 5/6.                   Steps are stored in order of step number.
1.1 Type 2+2.
1.2 Type 3•4•5•6.


Type part 2.
2.1 Type 4*5.
```

ORDER OF PROGRAM EXECUTION

The part is the major unit of program execution. When commanded to do a part, MATH follows the step sequence within that part, unless directed by a step to go to some place (a step or part) other than the next step. For the verbs that change this order of computation, see *Do*, *To*, *Done*, *Quit*. Otherwise MATH will not move automatically to a next *part*. A program can be started by executing a part with any number. However, most people start with part 1, viewing it as something like an executive routine controlling the whole program.

```
1.1 Type "a".
1.2 Type "b".
1.3 Type "c".

2.1 Type "d".
2.2 Type "e".
```

See page 37 for use of quotes.

```
Do part 1.
a
b
c
Do part 2.
d
e
```

Part 1 execution.

Part 2 execution.

```
1.25 To step 2.2.

Do part 1.
a
b
e
```

Step inserted in part 1.

Note that c and d do not appear.

VALUE RANGES

Range of value expressions are used in *for* phrases and in special functions (page 51):

x = a,b,c	means take values a,b, and then c for x.
x = a(b)c	means take values for x from a to c in steps of b.
x = a,b(c)d(e)f,g	means that x will be given successively the values a,b,b+c,b+2c,...,d,d+e,d+2e,...,f,g.

Note that there is no comma between b(c) and d, since d is the last value for the subrange b(c)d and the first value in the subrange d(e)f. If b+nc<d and b+(n+1)c≥d, the subrange will terminate with ...,b+nc,d. Then the values d+e,d+2e,...,f, and finally g will be used.

```
a=3
1.1 Type i.

Do part 1 for i=1(1)3(a)a*2.          Note that expressions may be used.
            i =          1
            i =          2
            i =          3
            i =          6
            i =          9

Do part 1 for i=0(1/a)1.
            i =          0
            i =           .333333333
            i =           .666666666
            i =           .999999999
            i =          1            End of range is always hit exactly.
```

ARRAYS

Values may be stored in places named by indexed letters. These arrays may carry up to ten indices whose values must be integers in the closed range -250 to +250. A letter may have only one dimension at a time; thus if you have already stored values for a(2,1) and a(3,4) and request that a(2,3,1) be given a value, a(2,1) and a(3,4) will be erased and only a(2,3,1) will remain. Similarly, a nonindexed letter "a" will also be erased. Letters so indexed are used to represent vectors and matrices of any dimension up to 10.

Indexing also plays the role of subscripting, to give greater freedom in naming than that provided by the 52 uppercase and lowercase letters of the alphabet. These uses may be mixed so that the first index stands for a subscript, and those following stand for the array; e.g., $a_i(j,k)$ becomes a(i,j,k) in MATH. Unlike FORTRAN, MATH does not reserve storage space for arrays according to "dimension" statements, but stores only specified values. (See also page 41.)

```
Set a(251)=3.
Index value must be integer and |index|≤250.

Set a(1,2,3,4,5,6,7,8,9,8,7)=5.
Please limit number of indices to 10.

Set a(2,1)=5.
Set a(3,4)=10.

Type a(3,4).
     a(3,4) =          10
Type a.
     a(2,1) =           5          Note that entire array is typed.
     a(3,4) =          10

Set a(2,3,1)=7.
Type a.
                                    Old array replaced because of
   a(2,3,1) =           7          dimension change.
a=3
Type a.

            a =         3          Array replaced by scalar.
```

SUMMARY OF RANGES

The examples below show, by MATH error messages, the limitations on step, part, form numbers, values, and indices. These ranges may be exceeded inadvertently, since values may be generated by the program.

```
1.234567876543 Set x=3.
Please limit step labels to 9 significant digits.

Type 1234.56789876.
Please limit numbers to 9 significant digits.

Set a(2345)=-17.
Index value must be integer and |index|≤250.

Do part 10*63.
Part number must be integer and 1≤part<10*9.

Do step 10*63.
Step number must satisfy 1≤step<10*9.

Type form 5.3.
Form number must be integer and 1≤form<10*9.

Type 10*100.
I have an overflow.
```

Range is $\pm 10^{-63}$ *to* $\pm 9.99999999 \cdot 10^{63}$.

FILES

Your long-term files are held on IBM 2314 or 2311 magnetic disks. Each IBM 2314 disk pack has a capacity of 16,000 "spaces". A space is roughly equivalent to 150 MATH cells (See page 39). The MATH disks will hold any number of private files, each with an almost unlimited number of items. However, the total number of available disks and the maximum number of spaces you are permitted to use are parameters set by computer center authorities. The number of "spaces" an item occupies is indicated in the SPACE column of your file summary (See page 45). If required, your "space" limit can be raised.

BUGS AND FAILURES

Bugs, those hard-to-find errors that prevent a program from working properly, fall into two major categories. First, the overall flow logic of the program itself may be in error. It is always good practice and discipline to go through the program manually using a few initial values from ranges. In writing this out, the program steps are reordered according to the way MATH will execute them, including iterations. You should also do sections of a program on MATH itself. Second, very subtle errors caused by roundoffs, accumulation of errors, very large values, or values close to zero may arise. You should always know enough about your problem, through previous hand calculation or other means, to detect gross output errors. But for subtle ones, the best advice is, THINK and then consult a numerical analysis man or a Mathmeister.

Failures of hardware or softwear can and do occur. If you log back onto MATH, but do not get the *I'm getting your work from your previous session.* message, the failure has lead to the loss of your program.

ERROR MESSAGES

With one major exception MATH error messages are on the whole self-explanatory and directly to the point. The exception is *Eh?*. It would be difficult to flag all such errors individually.

Eh? Check List:

1. Not typed properly. (See page 12.)
2. Words used not in MATH vocabulary.
3. Mathematical formatting errors (most frequently a missing multiplication dot or right-hand parenthesis).
4. A space between function, formula, or array name and left parenthesis.
5. Use of "ell" instead of one, "oh" instead of zero, "period" instead of dot.
6. Attaching a *for* phrase to verb other than *Do*.
7. Typewriter error in sending message to MATH. Retype.

Some individual error messages of note are the following:

Sorry. Say again.: Retype.
I can't express value in your form. Usually underscores to left of decimal point are insufficient in number. (An underscore is needed for a minus sign if the value is negative.)
I have too many values for the form. Check for correct number of fields. Note in the examples, however, that the number of values can be less than the number of fields and that underscores can also fill fields.
I have nothing to do. Calculation in progress is complete or canceled. A new *Do* is needed.