MEMORANDUM
RM-5322-PR
MAY 1967

JOSS: PROBLEM SOLVING FOR ENGINEERS

E. P. Gimble

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-5322-PR
MAY 1967

JOSS: PROBLEM SOLVING FOR ENGINEERS

E. P. Gimble

This research is supported by the United States Air Force under Project RAND—Contract No. F44620-67-C-0045—monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

Published by The RAND Corporation

PREFACE

JOSS[†] is a computing system developed at RAND to provide the individual scientist or engineer with a personal computational service, immediately available whenever required, in his own working environment. The system consists of a central computer containing the JOSS software and a number of typewriter consoles connected to the computer via telephone lines. The central computer turns its attention rapidly from console to console, in such a way that each user seems to have exclusive use of the system.

From the engineer's point of view, JOSS is a problem-solving tool that he can use in solving numerical problems with a minimum investment on his part in learning its use. This memorandum introduces the basic principles of JOSS operation through examples and discussion, including a step-by-step demonstration of the features of the system that the reader may try for himself. These principles are presented in a sequence designed to enable the engineer to solve progressively more involved scientific problems. In addressing itself to an audience of users with access to the system, the memorandum assumes that the reader is familiar with the mechanics of the operation of the JOSS console. This background may be gained either from a short demonstration at a JOSS console or by a reading of pages 7-13 of C. L. Baker's JOSS: Introduction to a Helpful Assistant (The RAND Corporation, RM-5058-PR, July 1966).

[†]JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

This document, as originally published in August 1966 under the title Problem-Solving with JOSS, was written by E. P. Gimble, an electronics engineer with the Service Engineering Division, Sacramento Air Materiel Area, McClellan Air Force Base, California. In recognition of the value of this text to engineers, C. L. Baker has prepared the present version for publication in the RAND series of JOSS documentation, a part of The RAND Corporation's continuing program of research in computer sciences under U.S. Air Force Project RAND.

SUMMARY

JOSS, RAND's interactive, remote computing system, is designed to provide engineers with the ability to solve complex numerical problems. The JOSS language is easy to learn and easy to use, with relatively few rules governing correct use. To use JOSS in the solution of numerical problems, the engineer needs only to express the required answers in terms of formulas and to specify the values of the parameters involved. The formulas and values are entered into JOSS through a typewriter keyboard.

This memorandum begins with an overview of the JOSS system, including descriptions of the console, the language, and the basic principles of operational procedure. Examples of JOSS input and output are used to demonstrate step by step how the basic commands, both direct and indirect, are applied.

Direct commands are used to solve problems when only a few answers are required. On presentation of a direct command, JOSS immediately performs the requested action. If formulas or values are misused or missing, JOSS responds with a message indicating the trouble.

Indirect commands are used to solve problems when many answers are required, when many variables are to be set, or when a number of steps must be performed in a prescribed sequence. A command becomes indirect when it is prefixed by a step number, indicating its position within a sequence. The step number is, in effect, a request for JOSS to store the command for interpretation and execution when the step is reached in the course of a program.

The versatility of JOSS as an aid in solving complex

problems is demonstrated by a discussion of special techniques. Such techniques include choosing between several acceptable commands, minimizing function arguments, and directing JOSS to "dress up" output with headings and condensed printout. The interactive use of JOSS in correcting program errors is also discussed, including explanations of error messages that may arise.

A JOSS filing system provides long-term storage for programs and data. This system eliminates the need for retyping frequently used material and allows a program to be constructed gradually over a period of time.

The appendices to this memorandum consist of a list of legitimate JOSS commands, both direct and indirect, including rules of form and notes applicable to the commands; a description of the use of expressions and propositions; and a discussion of the available JOSS functions.

CONTENTS

Chapter 4
TECHNIQUES

Chapter 5
FILING

AUTHOR'S PREFACE

JOSS is a "computing aid interacting with the user by means of a simple language." The aim of this document is to acquaint the reader with the language and the manner of interaction to give him the immediate ability to solve problems on JOSS. Our method is to enumerate the various JOSS commands and to show, by examples and discussion, JOSS's response to each of them. With an understanding of these basic principles, the user should be able to solve progressively more involved scientific problems. If possible, he should actually use JOSS during the reading of this document and immediately apply the principles being discussed. He should try the examples, inserting various expressions, and also solve real problems with JOSS.

Machine limitations have not been fully described. These will be pointed out in the form of error messages from JOSS as the need arises.

The JOSS language is summarized in the appendices. The material for this section was drawn almost entirely from JOSS: One-page Summary, JOSS Functions: A Brief Description, and What's New in JOSS II, informal memos prepared by The RAND Corporation. The remainder of the material in the document is an expansion of the above memos together with information the author has learned directly from using JOSS.

August 1966

E. P. Gimble
McClellan Air Force Base
Sacramento, California

Chapter 1

WHAT IS JOSS?

1.1. THE JOSS† SYSTEM

JOSS is an on-line, time-sharing computer system developed by The RAND Corporation for direct use by engineers without the assistance of programmers. Terminals are connected through individual multiplexor lines to the computer. Inputs are accepted one typed line at a time. Multiplexor lines are rapidly sampled in sequence by the computer so that outputs are generally as fast as the IBM typewriter that gives the printout.

The console control box at each input/output console has an on/off switch that clears the immediate memory allotted to that console and automatically causes a request for operator identification. The only other panel switch is an interrupt button with which the user may stop JOSS to make changes in the midst of a program or computation. Also included are green and red lights that indicate, respectively, user control and JOSS control of the typewriter. Three white lights indicate "system on," "console on," and "typewriter on."

The typewriter has a standard IBM keyboard, altered by the addition of a few special mathematical symbols. Its operation is much like any electric model, but pressing a key causes the associated electronics to transmit a coded character to the computer. Striking the carrier return key

†JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

gives console control to the computer, thus presenting a complete line for interpretation.

There are three typewriter adjustments with which we should be familiar. The platen pressure lever located on the upper right should be positioned to the rear. The double space lever is next to it and should be positioned forward. The page adjust knob is on the left platen roller and should be positioned in. The page is adjusted so as to receive its typed heading on the top line by pulling out the knob, rolling to the correct position, and pressing in the knob.

1.2. THE LANGUAGE

Appendix A lists all the legitimate JOSS commands. Each command occupies a single line and is presented by striking the carrier return key.

On presentation of a direct command, JOSS immediately takes the requested action and then returns control to the user. The direct command is not stored.

A command becomes indirect when it is prefixed by a step number (mixed decimal number). The presentation of an indirect command is, in effect, a request for JOSS to store the command. JOSS will interpret and execute the indirect command when that step is reached in the course of a program. A program is a step or set of steps (indirect commands) to be performed in a specified order.

The 26 upper- and 26 lower-case English letters are used as identifiers. Each of the 52 identifiers may be defined as a variable, function, or proposition. An identifier must always be a single letter. A definition is changed (or corrected) by simply reusing the identifier.

We shall define an expression as one number or identifier or a combination of numbers and/or identifiers and/or JOSS functions (App. C) that is reducible to a number when JOSS is called on to use it. Thus, the term "expression" used alone will mean "mathematical expression reducible to a number." The usual mathematical symbols

$$| \ |, \ [\ ], \ (\ ), \ *, \ /, \ \cdot, \ +, \ -$$

are used. The symbols are conventional, except for the asterisk, which indicates exponentiation.

The order of precedence for operations is conventional:

1. | |, [ ], and ( ) from inside outward
2. *
3. · and / from left to right within each term
4. + and -

Attention to this order is necessary because we are restricted to a single line for the expression. The left-to-right rule, for example, means that

$$x/2\cdot y \quad \text{is} \quad \frac{xy}{2}, \quad \underline{\text{not}} \quad \frac{x}{2y},$$

$$x/2/y \quad \text{is} \quad \frac{x}{2y}, \quad \underline{\text{not}} \quad \frac{x}{2/y}.$$

The order of precedence rule, for example, means that

$a+b\cdot c+d$ is $a+(b\cdot c)+d$ not $(a+b)\cdot(c+d)$, nor $(a+b)\cdot c+d$,

$a\cdot b+c\cdot d$ is $(a\cdot b)+(c\cdot d)$ not $a\cdot(b+c)\cdot d$, nor $(a\cdot b+c)\cdot d$,

and $$x/y*2 \quad \text{is} \quad \frac{x}{y^2}, \quad \underline{\text{not}} \quad \left(\frac{x}{y}\right)^2,$$

$$x*2/y \quad \text{is} \quad \frac{x^2}{y}, \quad \underline{\text{not}} \quad x^{2/y}.$$

JOSS also handles Boolean expressions composed of mathematical statements using the relational operators

$$=,\ \neq,\ \leq,\ \geq,\ <,\ >,$$

and the negation

not,

and connected in turn by the logical operators

and, or.

These will be discussed further in Sec. 2.7.

1.3. ERROR CORRECTION

JOSS helps the user to type his commands and expressions precisely. During the typing of a line, the user may backspace and strikeover, or he may use the strikeout symbol # to delete a character. To delete an entire line, he may begin or end the line with an asterisk.

If errors are detected in the line as finally presented, JOSS will respond with an error message. For improperly written direct commands, JOSS will give an immediate error message; the direct command should be reentered. For indirect commands, however, the only check that JOSS makes is for a valid step number. Thus, a decimal number followed by a space and any string of characters, for instance, will be stored as a step. Errors in the command itself, if any, are detected only when the command is interpreted; in this case, JOSS stops at the point of the error, gives a message beginning with "Error at step ...", and returns control to the user.

If an improper formula definition is indicated, the definition may be replaced or corrected by reusing the identifier. The step itself may be replaced or corrected by reusing the step number. JOSS resumes the program at the point of error when the Go command is given.

Chapter 2

IDENTIFIERS AND DIRECT COMMANDS

2.1. INTRODUCTION

Many engineering and other mathematical problems can be solved by (1) expressing the required answer in terms of formulas, (2) determining the values of the parameters involved, and (3) evaluating these formulas for these values. To solve such problems with the aid of JOSS, we do (1) and (2) by entering formulas and values through the typewriter keyboard; JOSS will evaluate and type the answer (3) when requested. Missing or misused formulas will be pointed out in lieu of an answer. Typing (1) and (2) is similar to setting up the problem with pencil and paper.

All methods of solving problems, other than calculating numerical expressions, normally require the definition of identifiers as variables, functions, and/or propositions. These identifiers are defined and used on JOSS according to standard mathematical usage and notation except for a few added English words.

In this chapter we will use only direct commands. We will solve problems by using the Type, Set, Let, and Delete commands directly. Type or Delete may be used at any time to examine or to delete part or all of our input.

Appendix A presents a list of the allowable forms of these four commands and includes rules of form and notes that are applicable to the commands. Appendix B describes the use of expressions and propositions, and App. C discusses the available JOSS functions. The reader should make reference to these sections, as he finds necessary,

for elaboration of the material presented in the main text.†

2.2. CALCULATION: TYPE COMMAND

When we enter "Type □.", JOSS repeats the expression signified by "□" and types its value: "□ = (number)". The expression is, in fact, a number.

EXAMPLES

```
Type 2.
           2 =         2
Type _,3*2,3.1416•3*2,_,sin(3.1416/6), 5•7•11/12/8•3.

         3*2 =         9
  3.1416•3*2 =        28.2744

sin(3.1416/6) =         .50000106
5•7•11/12/8•3 =       12.03125
```

Note that expressions separated by commas may be listed in one Type command, and that underscores may be used to indicate null items for formatting JOSS's output. (A void response would result from "Type all.", as JOSS has stored no information.)

2.3. VARIABLES: SET COMMAND

We define a variable by choosing an identifier (such

†The symbol □ is used throughout this memorandum to indicate that any mathematical expression reducible to a number may be substituted. Actual JOSS input and output will be distinguished by a two-color, elite-typeface, indented format. The reader with access to a JOSS console may verify JOSS's responses by entering these examples as he reads them.

as "x") and typing "x = □" or "Set x = □."

```
Set x = 5•10*3.
y=sqrt(x/3) + 5
Type all.

          x =      5000
          y =        45.8248291
```

The identifiers x and y are set to the proper numerical values and retain these values until redefined. This command has two distinct parts joined by an equals sign. The expression that appears to the right of the equals sign must be reducible to a number. The single letter to the left of the equals sign, although it may have been previously defined as a number (variable), is not a number in this command, but an identifier about to receive a new definition. For example, after defining x above, we may type

```
x = 2•x+1
```

The expression on the right of the equals sign is a number, while the x on the left is the identifier being newly defined.

We may find the value of a variable by typing

```
Type x.
          x =    10001
```

which is the same command as used in Sec. 2.2.

A lengthy expression may be shortened by using variables to represent its parts.

EXAMPLE

Evaluate the expression

$$\frac{\left(6 + \frac{11}{32}\right)^2 + \frac{42 - \sqrt{11}}{17}}{\left(6 + \frac{11}{32}\right)}$$

by assigning identifiers to the values of parts of the expression.

Solution:

```
a = 6+11/32
b=[42-sqrt(11)]/17
Type (a*2+b)/a.
   (a*2+b)/a =          6.70244836
```

We may define frequently used constants, such as pi and a conversion factor for changing from degrees to radians, by

```
p=arg(-1,0)
k=p/180
```

where arg is the JOSS abbreviation for the argument function of a rectangular coordinate point.† Any of these defined

†The argument function is used here merely as a convenient way of getting a numerical value for the constant π; see App. C for a detailed description of this function.

variables, or constants, may now be used to form expressions. For example:

```
x=2
Type p•x*2, 45•k.
       p•x*2 =        12.5663706
        45•k =          .785398163
```

The status of our input may be examined by typing

```
Type all.
```

```
          a =         6.34375
          b =         2.27549266
          k =          .0174532925
          p =         3.14159265
          x =         2
          y =        45.8248291
```

2.4. VARIABLES DEFINED BY FORMULAS: LET COMMAND

We can tell JOSS how to calculate x, rather than fixing the value of x as in the previous section, by typing "Let x = □." Before demonstrating this, however, we should type "Delete all." to instruct JOSS to erase all of our previous inputs.

```
Delete all.
Let d = c/arg(-1,0).
Type all.
```

```
          d:  c/arg(-1,0)
```

The use of Let indicates that the identifier d is not set to the numerical value of the expression on the right,

but is defined by the formula. This formula for diameter is indeed an expression reducible to a number when all of its elements are defined, but its value will be calculated only when d is called for; as when we enter

```
c = 10.5
Type d, arg(-1,0)•d*2/4.
           d =          3.34225381
arg(-1,0)•d*2/4 =       8.77341623
```

Although JOSS has not stored a value for d, it calculates a value each time d is called for, provided the ☐ is reducible to a number at that time.

If ☐ contains a Y that is undefined, for example,

```
Let x = Y*2•arg(-1,0).
```

we get

```
Type x.
Error in formula x:  Y = ???
```

when we attempt to use x. JOSS helps us keep our formulas straight. We may check our formula with

```
Type formula x.
          x:  Y*2•arg(-1,0)
```

JOSS's response presents the formula as we defined it.

Formulas may be broken into parts in much the same way used for the example in Sec. 2.3.

EXAMPLE

Solve each of the following equations for x:

$$10x^2 + 45x + 29 = 0,$$

$$1.7x^2 + .08x + .95 = 0.$$

Solution:

We write the quadratic formula

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

in parts by typing

```
Delete all.
Let A = -b/2/a.
Let B = sqrt(b*2-4•a•c)/2/a.
```

then set a, b, c and solve; thus

```
a=10
b=45
c=29
Type A+B, A-B.
        A+B =        -.77945588
        A-B =       -3.72054412
```

We may reset a, b, c and solve the second equation:

```
a=1.7
b=.08
c=.95
Type A+B, A-B.
Error in formula B:  I have a negative argument for sqrt.
```

This means that the answer will be A ± Ci, where i is the square root of -1 and defining C as

```
Let C = sqrt(-b*2+4•a•c)/2/a.
Type A, C.
          A =         -.0235294118
          C =          .747174612
```

Let us check the present contents of our immediate memory:

```
Type all.


          A:  -b/2/a
          B:  sqrt(b*2-4•a•c)/2/a
          C:  sqrt(-b*2+4•a•c)/2/a

          a =         1.7
          b =          .08
          c =          .95
```

The fact that a variable may be defined by a formula containing variables, which are defined in turn by other formulas, simplifies the analysis of physical problems. Unknown quantities may be designated by letter identifiers; the user may enter these identifiers and formulas in any convenient order.

EXAMPLE

Find the tensile stress in a wire with radius of r supporting a triangular piece of material with sides a, b, and c, thickness t, and density d (in inches and pounds).

Solution:

Stress, x, is weight divided by wire cross-sectional area:

```
Delete all.
Let x = W/A.
```

Wire cross-section:

```
Let A = p•r*2.
p = arg(-1,0).
```

Weight is thickness times face area times density:

```
Let W = t•f•d.
```

Area of triangular face of weight:

```
Let f = [s•(s-a)•(s-b)•(s-c)]*.5.
Let s = (a+b+c)/2.
```

All that remains to be done is to set values for r, t, a, b, c, d, and type

```
Type x,W,A.
Error in formula W:  t = ???
```

Thus, JOSS will remind us in the event parameters have not been defined.

```
r=.1
t=1
a=15
b=20
c=25
d=490/12*3
Type x,W,A.
          x =      1353.92226
          W =        42.5347223
          A =          .0314159265
```

To solve the above example with a change in any of the parameters, say, a and b, it is only necessary to type

```
a=18
b=18
Type x,W,A.
          x =     1461.31468
          W =       45.9085546
          A =         .0314159265
```

because all other identifiers retain their definitions.

```
Type all.
```

```
A:  p•r*2
W:  t•f•d
f:  [s•(s-a)•(s-b)•(s-c)]*.5
s:  (a+b+c)/2
x:  W/A

a =       18
b =       18
c =       25
d =         .283564815
p =        3.14159265
r =         .1
t =        1
```

2.5. FUNCTIONS: LET COMMAND

We may assign the identifier x to be a (user-defined) function of two arguments by typing "Let x(y,z) = □."

```
Delete all.
Let x(y,z) = y*2 + z/2.
```

The letters y and z in this example are dummy arguments; up to ten of these may be used in defining a function. Thus,

$$x(a,b) \quad \text{is} \quad a^2 + \frac{b}{2},$$

$$x(2,4) \quad \text{is} \quad 2^2 + \frac{4}{2},$$

and so forth, and x has become a function. We may apply the newly defined function by furnishing, in this case, two arguments:

```
Type x(3,7).
     x(3,7) =        12.5
```

A user-defined function, together with arguments, is an expression (reducible to a number). The identifier used alone is defined as a function and has no numerical value. When we try

```
Type x.
     x(y,z):  y*2 + z/2
```

JOSS does not respond as it did to a "Type □." command because x, alone, is not an expression. We are saying "Type the definition of x." JOSS gives us the definition of the function x exactly as we typed it initially. Note that the same type of response was obtained in Sec. 2.4 by

```
Type formula x.
     x(y,z):  y*2 + z/2
```

which is also a valid command here.

The value of the function x(y,z) is not defined at this point because y and z have not been defined.

```
Type x(y,z).
y = ???
```

The use of an identifier as a dummy argument does not define it nor affect its previous definition in any way.

Some useful user-defined functions are log to base 10:

```
Let L(x) = log(x)/log(10).
```

and tangent:

```
Let T(a) = sin(a)/cos(a).
```

Now let us use the newly defined functions. First we will define a radian/degree constant:

```
k=arg(-1,0)/180
```

It is clear from the defining expressions (to the right of the equals signs) that numbers or expressions are to be used in place of the dummies x and a. So we type

```
Type L(100), T(45•k).
     L(100) =        2
    T(45•k) =        1
```

In the derivative we have a function of a function, which is defined (approximately) by

$$\frac{d}{dx} f(x) = \frac{f(x + \delta) - f(x)}{\delta}$$

or, for JOSS:

```
Let D(f,x) = [f(x+.0001) - f(x)]/.0001.
```

In using this function, the dummy f must be replaced by a function, and the dummy x replaced by a value for the independent variable. We can evaluate (approximately)

$$\frac{d}{dx} \tan x\Big|_0 \quad \text{and} \quad \frac{d}{dx} \sin x\Big|_{30^\circ}$$

and compare the latter with cos 30° by the command

```
Type D(T,0), D(sin,30•k), cos(30•k).
      D(T,0) =        1
 D(sin,30•k) =         .866
   cos(30•k) =         .866025404
```

The integral function may be defined by Simpson's rule, which tells us that the area under a curve from a to b is

$$\frac{\Delta x}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 \ldots + y_n),$$

where $\Delta x = \frac{b - a}{n}$,

$y_0 = f(a)$,

$y_n = f(b)$,

and the general ordinate is

$$y_i = f\left[a + \frac{i}{n} (b-a)\right].$$

We define this function in JOSS by typing

```
Let I(f,a,b) = (b-a)/n/3•[f(a) + f(b) + S(f,a,b)].
Let S(f,a,b) = sum[i=1(1)n-1: [3-(-1)*i]•f(a+i•[b-a]/n)].
n=100
```

where f is the function, and a and b are the lower and upper limits. An even number must be used for n. One hundred gets good accuracy (around .001 percent, depending on the function). Larger n improves accuracy but takes more computation time. Other schemes for numeric integration (for example, two-point Gaussian) may be constructed in an analogous fashion.

EXAMPLE

Using the Simpson's rule integral function defined above, we may numerically integrate JOSS functions or those that we define:

$$\int_0^{10} x^3\,dx \quad \text{and} \quad \int_0^{90^\circ} \sin x\,dx.$$

For example:

```
Let a(x) = x*3.
k=arg(-1,0)/180
Type I(a,0,10), I(sin,0,90•k).
   I(a,0,10) =     2500
I(sin,0,90•k) =       1
```

2.6. INDEXED VARIABLES: SET COMMAND

A variable may be indexed and defined by setting it equal to a number in the same manner as were simple variables in Sec. 2.3. Variables defined by formulas (using Let) may not be indexed.

Each indexed x, for example, is a distinct variable and may be assigned values and used in the same way as other variables. We may type

```
Delete all.
x(1) = 9
x(2) = 13
```

and then use these x's as numbers:

```
Type 2•x(1), x(2)*2.
     2•x(1) =       18
     x(2)*2 =      169
```

The nonindexed variable x cannot coexist with the indexed variable x. Accordingly, in this case the command

```
Type x.
       x(1) =        9
       x(2) =       13
```

results in <u>all</u> the defined values of x being typed, just as a single value was typed for a variable in Sec. 2.3.

In Sec. 2.3 we indicated that an identifier may be used in only one way at a time and that redefinition deletes any previous definition. In the example above, x is being used as an indexed variable with a dimensionality of 1. If we now type

```
x(1,1) = 25
```

x(1) and x(2) are deleted:

```
Type x.
     x(1,1) =       25
```

An identifier may be indexed with from one to ten subscripts with integer values in the range -250 through +250.

When we need to set a large number of variables, the use of indirect commands (Sec. 3.3) simplifies the procedure.

Indexed variables are frequently used with the JOSS functions sum, prod, max, min, and first. (See App. C.)

EXAMPLE

Assume that for a straight member, the weight distribution has been defined by n weights, w(i), and their moment arms, x(i), about some reference point.

```
w(1) = 5
x(1) = -3
w(2) = 7
x(2) = 4
w(3) = 6
x(3) = 2
w(4) = 8
x(4) = -5
n=4
```

Find the total moment M, total weight W, and the location C (with respect to the reference point) of the center of gravity.

Solution:

```
M = sum(i=1(1)n: w(i)•x(i)).
W = sum(i=1(1)n: w(i)).
Type M,W,M/W.
          M =      -15
          W =       26
        M/W =        -.576923077
```

The iterative phrase reads "[sum] over i, from 1, in steps of 1, through n." The i (or other letter) used as the iteration variable in the arguments of sum, prod, max,

min, and first functions is a dummy; that is, the use and definition of identifier i outside the bracketed argument are not affected.

2.7. PROPOSITIONS: LET COMMAND

Propositions are Boolean expressions† composed of arithmetic statements using the relational operators

$$=,\ \neq,\ \leq,\ \geq,\ <,\ >.$$

If there is more than one statement, they are connected in turn by the logical operators

and, or.

Each proposition has a logical value of true or false. To define an identifier as a proposition, we type

```
Delete all.
Let x=2<3 and not 3≥4 or (5=3 and 2=2).
```

with the defining proposition following the first equals sign. To see how JOSS handles propositions, we type

```
Type x.
          x =      true
Type formula x.
          x:  2<3 and not 3≥4 or (5=3 and 2=2)
```

†In this text, the term expression used alone (or the symbol □) means a mathematical expression that is reducible to a number, and Boolean expressions (propositions) are excluded.

Expressions may replace the numerical values in the foregoing illustration. Also, the arithmetic statements may be replaced by identifiers defined as propositions, such as

```
Let p = x or y and z.
Type p.
Error in formula p:  y = ???
```

We have defined x as a true proposition. Let us define y and z with

```
Let y = true.
Let z = false.
```

JOSS reads the word "true" as a true proposition and "false" as a false proposition. Now the command

```
Type p.
          p =      true
```

can be executed.

The order of precedence of operations within a proposition is as follows:

1. evaluation of expressions
2. ( ) from inside outward
3. relational operations
4. not
5. and
6. or

A series of relational operations is an <u>and</u> chain. For example,

$$a < x = y > b$$

is read "a < x and x = y and y > b."

The truth value function, tv( ), takes a proposition as its argument. The truth value for a true proposition is 1; for a false proposition, it is 0.

```
Type tv(x), tv(y), tv(z), tv(x or y and z).
       tv(x) =        1
       tv(y) =        1
       tv(z) =        0
tv(x or y and z) =    1
```

Thus, the truth value function may be used as an expression since it is reducible to a number.

Aside from strictly Boolean applications, propositions find use in JOSS as conditions for the conditional command (see if phrase, Sec. 3.6) and for conditional expressions (Sec. 2.8).

Propositions may be written as functions, with their arguments consisting of expressions or propositions. For example, we may use the propositions defined above:

```
Let P(a,b) = a or b.
Type P(y,z).
      P(y,z) =      true
```

or use expressions as arguments:

```
Let p(a,b) = a>b.
Type p(3,2).
      p(3,2) =      true
Type all.


      P(a,b):  a or b
      p(a,b):  a>b
           x:  2<3 and not 3≥4 or (5=3 and 2=2)
           y:  true
           z:  false
```

2.8. CONDITIONAL EXPRESSIONS

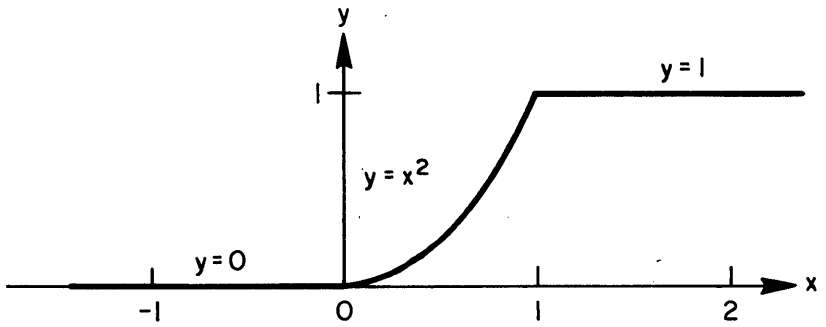
A conditional expression is indicated by a set of parentheses (or brackets) containing several expressions separated by semicolons, with each expression preceded by a proposition and colon:

(prop: ▭ ; prop: ▭ ; ▭)

The value of the conditional expression is the value of the expression ▭ that follows the first true proposition, reading from left to right. The conditional expression, therefore, reduces to a single expression for any given condition. The last proposition may be omitted if the last expression is to hold for all cases not covered by the other propositions.

EXAMPLES

1. Define the function shown below:




```
Let f(x) = (x<0: 0; 0≤x<1: x*2; 1).
Type f(-3), f(.25), f(.5), f(.75), f(3).
      f(-3) =        0
     f(.25) =         .0625
      f(.5) =         .25
     f(.75) =         .5625
       f(3) =        1
```

2. Define the factorial function.

Solution: By definition, if n is a positive integer, n! is 1 · 2 · 3 ... (n - 1) · n; if n is 0, n! is 1.

```
Let f(n) = (n=0: 1; fp(n)=0: prod(i=1(1)n: i)).
Type f(0), f(5), f(20).
        f(0) =         1
        f(5) =       120
       f(20) =         2.432902•10*18


Type f(3.5).
Error in formula f:  Eh?
```

The second proposition guards against inadvertent application of the function to a noninteger.

We may also write a conditional proposition by using propositions in place of the expressions. The conditional proposition reduces to a single proposition for any given condition.

EXAMPLE

Define f(x) to be true if x is even, false otherwise.

```
Let f(x) = [fp(x/2)=0: true; false].
Type f(1), f(2), f(5).
        f(1) =      false
        f(2) =      true
        f(5) =      false
```

2.9. SETTING VARIABLES TO BOOLEAN VALUES

In Secs. 2.3 and 2.6 we have emphasized that the Set command assigns a numerical value that the identifier retains until it is redefined. We will now find that with

a similar command we may assign Boolean values (Sec. 2.7) that the identifiers retain until redefined.

```
Set p(1) = true.
p(2) = a>b
a = ???
a=sin(3.3)
b=exp(-2)
p(2) = a>b
Type p.
        p(1) =      true
        p(2) =      false
```

Chapter 3

INDIRECT COMMANDS

3.1. INTRODUCTION

By the methods described in Chap. 2, we are able to solve a complex problem where only a few answers are required. We use indirect commands when many answers are to be typed, when many variables are to be set, or when a process is to be done repeatedly or in a particular sequence. Section 3.5 describes programs that set variables and organize tabular printout of data, using formulas to define the problems. Section 3.6 shows how calculations may be defined and performed by program steps.

3.2. STEPS AND PARTS; DO AND TO COMMANDS

We have defined a program as a step or set of steps (that is, indirect commands) to be performed in a specified order.

A step is defined by prefixing a command with a mixed decimal number, making it an indirect command. A step may be replaced or corrected by simply reusing the same step number.

A part is the collection of all steps whose step labels have the same integer part. This integer is the part number.

The Do command initiates a program by telling JOSS to do one part or one step. In doing a part, JOSS executes each step in turn by order of step number, regardless of the order in which the steps were typed. This means that a step may be inserted between two other steps at any time by using a step number whose value lies between that of the

other two steps. Execution of the last step in the part named completes the execution of the Do part □. command. To illustrate:

```
Delete all.
1.1 Type "The quick".
1.2 Type "jumps over".
2.1 Type "the lazy dog".
1.15 Type "brown fox".
Type all.

1.1 Type "The quick".
1.15 Type "brown fox".
1.2 Type "jumps over".

2.1 Type "the lazy dog".
→
```

Note how step 1.15 was inserted in the proper place.

```
Do part 1.
The quick
brown fox
jumps over
```

When JOSS finished part 1, the direct command was satisfied.

The part being done may contain Do commands that reach out to another step or entire part:

```
Delete all.
2.1 Type "brown".
2.2 Type "fox".
2.3 Type "jumps".

1.1 Type "The quick".
1.2 Do part 2.
1.3 Type "over the lazy dog".
Do part 1.
The quick
brown
fox
jumps
over the lazy dog
```

The To command has a different effect. Let us replace step 1.2 with

```
1.2 To part 2.
Do part 1.
The quick
brown
fox
jumps
```

Notice that To causes the part being executed to be discontinued at that point. A transfer is made to another part, and when the new part is finished, the direct command is satisfied. We may wish to transfer to some step other than the first in the new part:

```
1.2 To step 2.2.
Do part 1.
The quick
fox
jumps
```

The direct command is satisfied when the remainder of the new part is executed.

3.3. DO ITERATIONS; FOR PHRASE; DEMAND COMMAND

Do iterations are formed when the Do command contains the □ times phrase or the for phrase.

```
Delete all.
1.1 Type x*2.
Do part 1 for x = 1(1)4,6.
         x*2 =        1
         x*2 =        4
         x*2 =        9
         x*2 =       16
         x*2 =       36
Type x.
           x =        6
```

The for phrase, "for x from 1, in steps of 1 to 4, then 6," has caused JOSS to "Do part 1" for each value of x listed. In the process, x was actually set to each value listed, as evidenced by the response to Type x. Thus, a real identifier, not a dummy, is used in the for phrase iteration.

In Chap. 2 we used formulas to define our problem and obtained answers with direct commands for various settings of the parameters. We are now prepared to obtain as large a table of values as we wish by the use of Do iterations.

EXAMPLE

Define A, B, and C in terms of x by means of formulas.

```
Let A = x*2 + 5.2•x - 3.5.
Let B = x*2 - 6.1•x + 4.3.
Let C = x*2 + 3.3•x + 5.7.
```

Suppose we wish to evaluate these expressions for a number of values of the independent variable x. We may type

```
1.1 Type _,x,A,B,C.
Do part 1 for x=1(1)5(5)20.

          x =        1
          A =        2.7
          B =        -.8
          C =       10

          x =        2
          A =       10.9
          B =       -3.9
          C =       16.3

          x =
          A =
          B =
          C =      280.2

          x =       20
          A =      500.5
          B =      282.3
          C =      471.7
```

A Do iteration involving the Demand command is a convenient way to set a number of variables, since JOSS then helps with the typing. To input the data for the example in Sec. 2.6, we could type (for 5 weights)

```
1.1 Demand x(i).
1.2 Demand w(i).
Do part 1 for i=1(1)5.
        x(1) =
```

(Only one variable may be listed in each Demand step.) The Do iteration has begun. JOSS has set i = 1, typed the first part of the set x(1) command, and returned control to us for typing the value. When we type a value and strike carrier return, w(1) will be demanded. The process may be interrupted at any point by striking carrier return without entering a value. The Go command resumes the process, which is completed when all variables through w(5) are set.

```
        x(1) = 3
        w(1) = 5
        x(2) = 3.654
        w(2) = z*2
z = ???
        w(2) =
I'm at step 1.2.
z=23.5
Go.
        w(2) = z*2
        x(3) = z/7
        w(3) = 5
        x(4) = 3•z+4
        w(4) = 5.331
        x(5) = 7/3.551
        w(5) = 6
```

If we desire to see a printout of all the x(i) and w(i) that have been set, we may type "Type all." (JOSS will type steps and values), "Type all values.", or

```
Type x,_,w.
        x(1) =        3
        x(2) =        3.654
        x(3) =        3.35714286
        x(4) =       74.5
        x(5) =        1.9712757

        w(1) =        5
        w(2) =      552.25
        w(3) =        5
        w(4) =        5.331
        w(5) =        6
```

Since i is a real variable that has actually been set to the values in the iteration phrase, it now has the value 5.

```
Type i, x(i), w(i).
           i =        5
        x(i) =        1.9712757
        w(i) =        6
```

Do iterations using the ☐ times phrase find use in approximations requiring a repetitive process (see the last example on p. 39; note the comma between the step or part number and the ☐ times phrase).

3.4. FORMS AND FIELDS

The first printout on p. 31 gives one value per line, occupies considerable space, and is relatively slow. To improve the situation, define a form with fields. Forms are entered on two lines, the first of which must be "Form ☐:". The colon reminds us that the next full line is committed to the form definition. For example,

```
Form 1:
__._   __._   __._   __._
```

allows JOSS to type the four values on one line. The

form number must be an integer, followed by a colon; the form definition is typed on the next line. The formulas for A, B, and C are still in our immediate memory; before we use them again, we should delete the program steps no longer needed.

```
Delete all steps.
1.1 Type x, A, B, C in form 1.
Do step 1.1 for x=1(1)5(5)20.
  1.0       2.7       -.8       10.0
  2.0      10.9      -3.9       16.3
  3.0      21.1      -5.0       24.6
  4.0      33.3      -4.1       34.9
  5.0      47.5      -1.2       47.2
 10.0     148.5      43.3      138.7
 15.0     299.5     137.8      280.2
 20.0     500.5     282.3      471.7
```

The fields in the example have three places before and one place after the decimal point, so they will accept numbers less than 999.95. Significant figures less than .1 will be lost.

Fields for a tabular form of scientific notation may be indicated by strings of periods.

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

They are thus more convenient to use if the approximate size of the answers cannot be predicted. Compare the previous output with the following:

```
Do step 1.1 for x=1(1)5(5)20.
  1.0     2.700 00    -8.000-01     1.000 01
  2.0     1.090 01    -3.900 00     1.630 01
  3.0     2.110 01    -5.000 00     2.460 01
  4.0     3.330 01    -4.100 00     3.490 01
  5.0     4.750 01    -1.200 00     4.720 01
 10.0     1.485 02     4.330 01     1.387 02
 15.0     2.995 02     1.378 02     2.802 02
 20.0     5.005 02     2.823 02     4.717 02
```

The last three characters in each field represent the exponent part. The number of significant figures is 5 less than the number of periods in the field.

Sometimes it is desirable to include text with the typed numbers, such as

```
Form 2:
When x is ___.___, then A is ..... and B is .....
```

When using a form to format the data output, the Type command must list no more values than there are fields in the form; an underscore may be used to indicate that a field is to be left blank.

```
x=7.253
Type x,A,B in form 2.
When x is   7.253, then A is  8.682 01 and B is  1.266 01
Type x,A in form 2.
When x is   7.253, then A is  8.682 01 and B is
Type x, _, B in form 2.
When x is   7.253, then A is           and B is  1.266 01
```

We may also use a form without field indication for a heading, such as

```
Form 3:
Elevation Angle   Distance   Height
Type form 3.
Elevation Angle   Distance   Height
```

3.5. PROGRAMMED PRINTOUT

We will now deal with the use of programs to control the setting of independent variables and to obtain certain arrangements of data printout. Calculations will be defined by expressions and formulas as they were in Chap. 2.

We have seen that the Do part command results in the

execution, in numerical sequence, of each step of an entire part. A Do step command causes the single step to be done. A To step command (if executed while doing a part) causes the current part to be discontinued, and the new part to be done starting with the step named. The for phrase is used only with Do commands, and a Do iteration is formed if it includes an iteration expression.

The actions to be accomplished, such as demanding variables, typing a heading, and typing a block of data, can be made to occur in the desired order by the proper arrangement of program steps.

Indirect commands should be used for operations that must be done repetitively or in a specified sequence.

The defining of a formula, function, or proposition is required only once, and its timing is not critical. The Let command therefore should usually be given directly.

EXAMPLES

1. There may be a requirement to perform the same operation repeatedly on many data. A program such as the following might be used:

```
1.1 Demand a.
1.2 Demand b.
1.3 Type a•b, a/b, b/a in form 1.
Form 1:
a•b = ........., a/b = ........., b/a = .........
1.4 To part 1.
Do part 1.
          a = 3.5
          b = arg(1,1)
a•b =  2.749 00, a/b =  4.456 00, b/a =  2.244-01
          a = 3.7
          b = b
a•b =  2.906 00, a/b =  4.711 00, b/a =  2.123-01
          a =
I'm at step 1.1.
```

JOSS continues demanding values for a and b, as before, until the program is interrupted by striking carrier return without entering data.

2. This program gives a family of curves. The computation is by formula.

```
Delete all.
Let x = a*2•sin(b/a+5).
1.1 Type _,a.
1.2 Type "  b         x".
1.3 Do part 2 for b=1(1)4.
2.1 Type b,x in form 1.
Form 1:
       .........
Do part 1 for a=10(1)13.

          a =       10
 b        x
 1     -9.2581 01
 2     -8.8345 01
 3     -8.3227 01
 4     -7.7276 01

          a =       11
 b        x
 1     -1.1243 02
 2     -1.0791
```

3.6. PROGRAMMED CALCULATION; IF PHRASE

Until now we have done all computation simply by defining our problems in terms of formulas and functions, and have used programming as a means of obtaining the desired order of printout. In some instances, it is necessary or advantageous to fix the required order of calculations by means of a program.

The if phrase is frequently used in this type of program. Any command may become conditional by the addition of the if phrase. JOSS ignores the command unless the if phrase is satisfied.

EXAMPLES

1. We can find the prime numbers between 100 and 200 by dividing each number by 2, 3, 5, 7, etc., up to the square root of the number, and checking for integral quotients.

```
Delete all.
1.1 Do part 2 for w = 2,3(2)ip[sqrt(i)].
1.2 Type i in form 1 if i≠0.
2.1 Set i=0 if fp(i/w) = 0.
Form 1:

Do part 1 for i=100(1)200.
101
103
107
109
113
127
131
137
```

2. The following program finds the approximate real roots of an equation by Newton's method:

```
Delete all.
1.1 Set x = x - f(x)/d.
1.2 To part 1 if |f(x)/d|>e.
1.3 Type x, f(x), d in form 1.
Form 1:
.............     ........     ........
e=10*(-7)
Let d = [f(x+.00001)-f(x)]/.00001.
```

For example, to solve the equation

$$x^3 - 4x^2 \log x = 0,$$

we first explore the shape of the function:

```
Let f(x) = x*3 - 4•x*2•log(x).
Do step 1.3 for x=1(1)10.
 1.00000000 00      1.00 00     -1.00 00
 2.00000000 00     -3.09 00     -7.09 00
 3.00000000 00     -1.26 01     -1.14 01
 4.00000000 00     -2.47 01     -1.24 01
 5.00000000 00     -3.59 01     -9.40 00
 6.00000000 00     -4.20 01     -2.00 00
 7.00000000 00     -3.84 01      1.00 01
 8.00000000 00     -2.03 01      2.69 01
 9.00000000 00      1.71 01      4.88 01
 1.00000000 01      7.90 01      7.58 01
```

The columns are x, f(x), and d, since this is their order in step 1.3. By exploring the function in this way, we have found two zero crossings. We may refine these two approximate roots by

```
Do part 1 for x = 1,8.
 1.42961186 00     -1.30-07     -3.68 00
 8.61316937 00     -1.00-06      3.96 01
```

These are good approximations as evidenced by the small values for f(x). They may be checked by

```
Do part 1, 3 times.
 8.61316940 00     -2.00-06      3.96 01
 8.61316945 00      1.00-06      3.97 01
 8.61316942 00     -2.00-06      3.98 01
```

This is a manually directed program. Unless we know the general shape of our function, we may step over some roots during our original exploration. Also, for some functions, the value of e must be made larger in order to ensure convergence.

3.7. SEQUENCING PROGRAM STEPS; STOP, DONE, AND QUIT COMMANDS

There are four other useful programming devices that bear mentioning at this time. A conditional expression for

the step number could be used in step 1.2 of the preceding example to provide a printout for the condition d = 0 before we get the error message for a zero divisor in step 1.1.

```
1.2 To step [d=0: 3; |f(x)/d|>e: 1.1; 1.3].
1.3 Do part 2, 3 times.

2.1 Type x, f(x), d in form 1.
2.2 Set x = x-f(x)/d.

3 Type d, x, f(x).

Do part 1.
 8.61316947 00      0           3.98 01
 8.61316947 00      0           3.98 01
 8.61316947 00      0           3.98 01
```

Here we have specified three alternate program sequences in a single step. The 3 times printout used manually in the previous example has been written into step 1.3.

The other three devices to be noted are Stop, Done, and Quit. The indirect Stop command may be placed at any point in the program sequence, and JOSS will return control to us at that point. We may take a look at conditions, make changes as desired, and then resume the program with Go.

The Done command allows us to omit the remaining steps of a part. Done is usually given conditionally, except when added temporarily for part-program operation.

The Quit command, like Done, omits the remaining steps within the same part. In addition, Quit satisfies the command to Do that part, stopping any further iteration.

EXAMPLES

1. This program will give the solution to n simultaneous equations of the form

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= c_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= c_2 \\
\cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot & \\
\cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot & \\
\cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot \quad\quad \cdot & \\
a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= c_n
\end{aligned}
$$

if S is set to 1. It will perform the inversion of the n x n matrix a_{ij} and stop if S is not 1. The program will demand the necessary inputs; it types either x_i or a_{ij}^{-1}.

```
Delete all.
1.1 Demand n.
1.2 Demand S.
1.3 Do part 2 for i=1(1)n.
1.4 Do part 4 for k=1(1)n.
1.5 To step 9.1 if S≠1.
1.6 Do part 8 for i=1(1)n.
1.7 Type x.

2.1 Do part 3 for j=1(1)n.
2.2 Demand c(i) if S=1.

3.1 Demand a(i,j).

4.1 Do part 5 for j=1(1)n.
4.2 Set a(k,k) = 1/a(k,k).
4.3 Do part 6 for i=1(1)n.

5.1 Set a(k,j) = a(k,j)/a(k,k) if j≠k.

6.1 Done if i=k.
6.2 Do part 7 for j=1(1)n.
6.3 Set a(i,k) = -a(i,k)•a(k,k).

7.1 Set a(i,j) = a(i,j)-a(i,k)•a(k,j) if j≠k.

8.1 Set x(i) = sum(j=1(1)n:a(i,j)•c(j)).

9.1 Type a.
```

This program may be initiated by "Do part 1."

An inverted square matrix may be checked by adding the step

```
10 To step 1.4.
```

and initiating reinversion of the matrix by "Do part 10."

2. Let us use the Done command in a program to find the prime numbers less than 100 (see Example 1, Sec. 3.6):

```
Delete all.
1.1 Done if fp(n/2) = 0 and n≠2.
1.2 Done if fp(n/3) = 0 and n≠3.
1.3 Done if fp(n/5) = 0 and n≠5.
1.4 Done if fp(n/7) = 0 and n≠7.
1.5 Type n in form 1.
Form 1:
__
```

Initiate the program with "Do part 1 for n = 1(1)100." The Type command is reached only if none of the Done conditions is satisfied.

Chapter 4

TECHNIQUES

4.1. INTRODUCTION

The JOSS language and software are constructed so that programs may be evolved through trial and error. The machine cannot be hurt or the program "hung up" by combinations of invalid or illegal commands at the console. All that is necessary to solve most problems is a knowledge of the problem, a list of JOSS commands and functions (see Apps. A and C), and patience.

There are many alternate ways of writing a correct program, however, and this chapter aims at helping us with a few of the choices. Error message explanations are also given to help shorten the trial-and-error process of program debugging.

4.2. DIRECT VERSUS INDIRECT COMMANDS

Many commands may be given either directly or indirectly. The choice is simple if we remember the general rule: Use direct commands unless the operation is to be done repetitively or in some specified sequence.

Let is used to define formulas, functions, and propositions. Although it is possible to give the Let command indirectly, the better practice is to give this command directly. Generally, it is necessary to make these definitions only once during the course of a problem, and timing is not critical.

Consider some of the difficulties encountered with an indirect Let command.

Suppose that we enter

```
1.4 Let x = a/b.
```

as a step in our program, and JOSS later tells us that there is an

```
Error at step 4.3 (in formula x):  b = ???
```

because we should have written x = a/c. We may attempt to correct the trouble with

```
1.4 Let x = a/c.
Go.
```

But JOSS will still respond

```
Error at step 4.3 (in formula x):  b = ???
```

because x is still defined as a/b until step 1.4 is executed again.

Another possible mistake is to correct the formula directly:

```
Let x = a/c.
Go.
```

without correcting step 1.4, in which case formula x will be correct until it is redefined incorrectly by the execution of step 1.4.

Finally, it may be that the formula x is defined correctly but at too late a point in the program. In this

case we might get, for example,

Error at step 1.1: x = ???

because step 1.4 has not yet defined x. This is one reason for the general rule that Let should be used directly.

4.3. SET VERSUS LET

The Let command must be used when we need a function. However, we may choose between Set and Let when defining a variable.

The command

Let x = □.

allows x to follow the value of the formula as its independent variables change. This frees us from having to reset x repeatedly. Program sequence can be disregarded in the definition of x. However, the commands "x = □.", "1.1 Set x = □.", and "1.1 Demand x(i)." also have their advantages.

Set is more efficient than Let if the value is to be used repeatedly at each setting. (JOSS must recalculate a variable-defined-by-formula each time its value is used.) Set allows us to retain a value while independent variables change, as may be necessary in some programmed calculations. Set variables may be indexed.

The Demand command, under control of a Do command with a for phrase, is a good way to enter a long sequence of values under program control.

4.4. DEGREES VERSUS RADIANS

It is good practice to let all identifiers of angles be defined in terms of radian measure. This eliminates the need for conversion factors within formulas and reduces the possibility of programming error. Any necessary conversion should be made at the time of inputting and/or outputting data.

4.5. MINIMIZING FUNCTION ARGUMENTS

Considerations of space and convenience make it wise to use no more dummies in the definition of a function than are needed, particularly in expressions containing many function symbols. Consider the following examples:

EXAMPLES

1. In Sec. 2.5 we could have defined the derivative function as follows:

```
Delete all.
Let D(f,x,d) = [f(x+d)-f(x)]/d.
```

and its use would have required

```
Let T(x) = sin(x)/cos(x).
Type D(T,0,.00001).
D(T,0,.00001) =       1
```

or

```
d=.00001
Type D(T,0,d).
    D(T,0,d) =        1
```

2. But, since d will remain .00001, we omit it from the list of dummies in the argument of the function D, and type

```
Let D(f,x) = [f(x+d) - f(x)]/d.
d=.00001
Type D(T,0).
      D(T,0) =        1
```

3. Suppose our intended use of D is to set x to some value and examine (or use) the derivatives of several functions of x at that value. We could use D(f,x), as in Example 2, and type

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

but, since x is actually a defined variable, we may redefine the derivative function using one less dummy:

```
Let D(f) = [f(x+d)-f(x)]/d.
Type D(T), D(sin), D(cos).
       D(T) =        1
     D(sin) =        1
     D(cos) =        0
```

4. Suppose also that only one function, say, the tangent, needs differentiation in our problem, but we need symbols for the derivative of tan(x) at any value of x. We could define f as a tangent function and then let D be a function of x (only) by typing

```
Let f(x) = sin(x)/cos(x).
Let D(x) = [f(x+d)-f(x)]/d.

d=.00001
k=arg(-1,0)/180

Type D(0), D(15•k), D(30•k), D(45•k), D(60•k).
       D(0) =        1
    D(15•k) =        1.0718
    D(30•k) =        1.3334
    D(45•k) =        2
    D(60•k) =        4
```

5. If x and f will always be defined, D need only be a variable defined by the formula:

```
Let D = [f(x+d)-f(x)]/d.
d=.00001
Let f(x) = sin(x)/cos(x).
x=0
Type D.
          D =        1
```

4.6. DRESSING UP THE OUTPUT

There are two ways to direct JOSS to print headings with output: (1) using forms and a

2.2 Type form □.

command, and (2) using the

2.2 Type " ... our heading".

command. In either case JOSS repeats, stroke for stroke, the form definition or the material between the quotes.

To speed up the data printout, the tabulator can be used in the definition of the form for data. If we desire,

we may reset the mechanical tabs on the typewriter after the form has been defined. The same tabs may also be used in the definition of the heading. Backspacing during the definition of the heading will cancel spaces and tabs in reverse order.

The Line command is used to separate blocks of data.

```
1.71 Line.
1.72 Line.
1.73 Line.
```

Also, a blank line is given for each underscore in a Type command, such as

```
1.7 Type _,_,_.
```

We use a Page command in order to avoid dividing a block of data between two pages. For example, if the data block and heading together comprise ten lines, we would enter

```
6.25 Page if $>44.
```

just before the step that types the heading. The $ symbol carries the current line number. JOSS pages automatically after line 54.

Condensing the printout to get as many figures as possible on a page is a useful technique when we are producing extensive data. One way of accomplishing this is to present output in tabular form. As an example, let us use the cylindrical volume formulas.

```
Delete all.
Let v = h•A.
Let A = p•r*2.
p=arg(-1,0)

1.1 Type r,v in form 1.

2.1 Type _,h.
2.2 Type "Radius     Volume".
2.3 Do part 1 for r=1(1)10.

Form 1:

__          ___._

Do part 2 for h = 1,2,3.
```

```
            h =         1
Radius      Volume
 1             3.1
 2            12.6
 3            28.3
 4            50.3
 5            78.5
 6           113.1
 7           153.9
 8           201.1
 9           254.5
10           314.2

            h =         2
Radius      Volume
 1             6.3
 2            25.1
 3            56.5
 4           100.5
 5           157.1
 6           226.2
 7           307.9
 8           402.1
 9           508.9
10           6[illegible]
```

Part 2 gives one table of data. This program results in one table for each value of h, with about four tables per page.

A condensed printout may be achieved by redefining the volume v as a function of r. Then, area A must be defined as a function of r.

```
Delete all.

Let A(r) = p•r*2.
Let v(r) = h•A(r).

p=arg(-1,0)

1.1 Line.
1.2 Type "Ht.  Radius -->".
1.3 Type _,1,2,3,4,5,6,7,8,9,10 in form 1.
1.4 Line.
1.5 Do part 2 for h = 1,2,3.

2.1 Type h,v(1),v(2),v(3),v(4),v(5),v(6),v(7),v(8),v(9),v(10) in form 1.

Form 1:
__ ___._ ___._ ___._ ___._ ___._ ___._ ___._ ___._ ___._ ___._

Do part 1.

Ht.  Radius -->
     1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0

 1   3.1  12.6  28.3  50.3  78.5 113.1 153.9 201.1 254.5 314.2
 2   6.3  25.1  56.5 100.5 157.1 226.2 307.9 402.1 508.9 628.3
 3   9.4  37.7  84.8 150.8 235.6 339.3 461.8 603.2 763.4 942.5
```

Part 2 gives one line of data. Step 1.3 is a heading and uses a null value, _ , in the first field because no column heading is required.

4.7. DEBUGGING; ERROR MESSAGES

The salient feature of JOSS is the manner in which it works with us to make corrections during the inputting and execution of our program. When JOSS gives an error message, we may restate our command in the proper form, as we do, for example, in conversation when the other person fails to understand our comment.

We also use the Type command to question JOSS when the reason for the error message is not understood. We can use "Type all steps.", "Type all formulas.", "Type all

values.", etc., or even "Type all." when the amount of revision makes it difficult to see the current status of our program.

If the error message contains a step number, we may use the Go command after making the needed correction.

EXAMPLES

The following examples present common indications of errors, with possible causes listed below each example to aid in pinpointing and correcting errors.

1. "Eh?" means that the previous line is incorrectly written.
   a. If the line is an indirect command, the step number has been improperly written (for example, not followed by a space).
   b. If the line is a direct Let command, the Let x= portion is improperly written--the space omitted, or the command doesn't end with a period.
   c. The command may not be legitimate or may be incorrectly written. (See App. A.)
   d. Some expressions may be incorrectly written. (See App. B.)
   e. We should immediately retype the command with any change that seems plausible if we cannot find rules that apply. When the expression is long, time may be saved by constructing short examples to check our notation. For example:

```
Type [2+5)/3.
Eh?
```

implies that brackets and parentheses must be used in matching pairs.

```
Type 2, if 1=1.
Eh?
Type 2 if 1=1.
          2 =        2
```

tells us not to use a comma before the if phrase.

f. Transmission line noise can occasionally cause an error in transmission. If we feel that the line has been correctly written, we may check for a transmission error by re-typing the line as is.†

2. "Error at step 3.1: Eh?" means that the step number is correctly written, but that the command is improperly stated. Use the Go command after attending to this error.

a. To see how this step was last defined, type

Type step 3.1.

JOSS will type step 3.1 as received.

b. If step 3.1 is a Let command, the Let x= portion is written wrong, or the period is omitted.

c. Check for errors (steps 1.c, 1.d, 1.e).

3. "Error in formula A: Eh?", following a direct

†Simultaneous space bar/shift key before carrier return makes JOSS repeat the line as received, plus ## for the illegal space/shift.

command in which A is used, usually means that the form of the expression for A is in error.

a. To see how A was last defined, type

```
Type formula A.
A: □
```

b. Check for errors (steps 1.c, 1.d).

c. The possibility exists that formula A is, in fact, correctly written, but that the definition of one or more identifiers is not consistent with its use in formula A.

4. "Error at step 3.1 (in formula A): Eh?" is corrected as in Example 3. Use the Go command after attending to this error.

5. "Error at step 3.1: I can't find the required step." or " ... required part." is given when step 3.1 refers to a nonexistent step or part. Use Go after attending to this error.

a. Examine step 3.1 with

```
Type step 3.1.
3.1 To step 7.2.
```

b. If 7.2 is the required step, it has been omitted, deleted, or its step number incorrectly written. Investigate with

```
Type part 7.
7.  2 Line.
7.1 Type "  x          y".
7.3 Do part 8 for x = 1(1)10.
```

6. "Error at step 6.3: I can't express value in your form." or " ... I have too many values for the form." Use Go after attending to this error.
   a. Have JOSS type the step and type the form.
   b. Check the fields of the form (Sec. 3.2). Do not use hyphens for underscores. Use periods, not center dot, for decimals and for fields for scientific notation.
   c. For " ... I can't express value ...", type the values without a form to find out which ones are too large. Remember that JOSS rounds numbers as required to match the field size, and may, for example, round 9.99 ... to 10.00.... Check that a position has been provided for a minus sign if required.

```
Type x,y.
          x =      (number)
          y =      (number)
```

   d. Use scientific notation fields if necessary.
7. No printout. Check program order to see why Type command is not reached.
8. Printout not as expected.
   a. Use interrupt to stop needless printout.
   b. Check program order, Do iterations, and Type commands (Sec. 3.3).
9. Obviously-incorrect answers may be due to faulty problem analysis and/or setup, or to a number of JOSS inputting errors, including disagreement between: typed formulas and problem analysis, typed

variables and intended data input, program steps and desired operations, or the Type statement and intended quantities for printout.

a. Ask JOSS to type the Type statement. Check it against the problem setup to see if it gives the quantities wanted.

b. Call for formulas with

```
Type all formulas.
```

Check expressions for violation of order of precedence (App. B). In case of doubt, short examples may be tried. For example:

```
Type (2/3)*2, 2/(3*2), 2/3*2.
     (2/3)*2 =          .444444445
     2/(3*2) =          .222222222
       2/3*2 =          .222222222
```

verifies that the exponentiation is performed before the division.

c. Have JOSS type intermediate answers, or parts of longer expressions, to isolate the source of error.

10. "I'm at step 1.1. I ran out of space." or "Revoked. I ran out of space." means that the user's immediate memory is filled.

a. Check for endless loops via Do or use of Do instead of To.

b. Check for unlimited recursive definition.

c. Check for x defined in terms of y, and y in terms of x.

d. The program may be just too large for the immediate memory to handle. "Type size." may be given to determine how many of the available cells of immediate memory have been used. The file (Chap. 5) might possibly be used to partition the program and do a part at a time.

11. "I have an overflow." means that some number has exceeded 9.99999999·10*99 in magnitude.

4.8. MORE DEBUGGING TOOLS; CANCEL; PARENTHETIC DO

Other time-saving devices that may be used during program construction or debugging are interrupt; Cancel, Done, Stop, and parenthetic Do; and flow tracers.

When a process is stopped by an error message or interrupt, we may use direct commands for tests and alterations as in the preceding examples and then resume the process with the Go command. If we do not wish to resume the process, it will be automatically canceled when a direct Do command initiates another part or step. However, the Cancel command may be used at any time to cancel a stopped process and release some immediate memory.

A suggestion was made in Example 1 that simple direct commands may be constructed to test for proper notation, without disturbing a stopped program or calculation. We may also, at such a time, wish to Do a special trial part or an existing part as a test. Since this action would normally cancel the interrupted program, a special parenthetic Do is provided. In the following example, use interrupt when JOSS enters an endless loop.

```
Delete all.
1.1 To step 1.1.
Do part 1.
I'm at step 1.1.
2.1 Type x.
(Do part 2.)
Error at step 2.1:  x = ???
x=2
Go.
          x =        2
Done. I'm ready to go at step 1.1.
Go.
I'm at step 1.1.
```

Parenthetic Cancel cancels the current subexecution. The parenthetic Do process may be carried out indefinitely subject to storage limits.

The Done, Stop, or Quit commands may be temporarily inserted for partial program operation during debugging.

Use interrupt to stop any unprofitable process.

Tracers may be placed in a multipart program if we desire to determine what parts JOSS is performing between printouts. This is done by means of such commands as

```
1 Type "Start part 1.".
2 Type "Start part 2.".
7.99999999 Type "End part 7.".
```

JOSS will then indicate when it begins or ends a part.

Chapter 5

FILING

5.1. INTRODUCTION

The immediate memory allotted to our console is cleared by "Delete all." when we wish to begin a new problem, or by operation of the on/off switch when we turn the console over to another user. We may have JOSS "Type all." to get a copy of our program before clearing the memory. We will also want to file the program, for immediate recall without the necessity of retyping, if we expect to use the same basic program in the near future. This chapter explains the JOSS file system and its manner of use.

5.2. THE FILE SYSTEM

A centrally located magnetic disc unit provides large-volume storage into which information in the immediate memory may be filed. Users are assigned files, each of which is referenced by the file number followed by a preassigned 5-character code, enclosed in parentheses.

Each file provides space for 25 items of nominal size. An item may consist of combinations of steps, parts, forms, formulas, and values. The available space in a file may be fully occupied by fewer than 25 items if some of the items are large. A completely filled immediate memory corresponds to about one-third of a file.

The user assigns a number to each item at the time it is filed. The number may be followed by a code of from 1 to 5 characters in parentheses. Both number and code (if any) must be used when recalling the information.

5.3. THE FILE COMMANDS

The five commands, Use, File, Recall, Discard, and Type item-list. accomplish all the necessary file actions. Fictitious file and item codes will be used in our examples. The command

```
Use file 123 (abcde).
Roger.
```

makes our file available for use. The acknowledgment "Roger." indicates that henceforth all item references will be to items in this file until another Use command is given or the console on/off switch is operated. If file 123 (abcde) had not existed, JOSS would have responded with an error message: "I can't find the required file." (In this example, note the mandatory space required between the file number and the code.)

We may now file all or part of the information in our immediate memory with a command such as

```
File all as item 5 (jones).
Done.
```

As a result, both file memory and immediate memory contain the information. (If we try to file an item before we have opened a file, we get the error message "You haven't told me what file to use.") We may wish to file only all values, all formulas, part 1, etc. (see App. A). (If item 5 had already been occupied, an error message would have been given: "Please discard the item or use a new item number.")

To recall the same item at a later time, we must be using file 123 (abcde) when we type

```
Delete all.
Recall item 5 (jones).
Done.
```

Once again, the item is present in both our immediate memory and the file. (If item 5 (jones) had not existed, JOSS would have responded with an error message: "I can't find the required item.") Recall is the only file command that causes a change in our immediate memory. The effect is the same as if we had retyped the item. Existing steps, forms, and identifiers will remain as is, except where redefined by the recalled item.

When we no longer need the item, we may discard it while using file 123 (abcde) by typing

```
Discard item 5 (jones).
Done.
```

and the item 5 space is cleared. The immediate memory is unchanged by the Discard command.

5.4. USING THE FILE

We may want to file information for several reasons:

- We expect to solve similar problems using the same basic program or the same data.
- We need time to evaluate the results of a first trial before deciding how to modify the program and proceeding further.

- The problem is of such length that more than one sitting is necessary to compose a program.
- We want to try a modification of a part, with the option of restoring the current setup.
- We want to keep general processes as subroutines.
- We have not finished our session by the scheduled recess hour.

An index is necessary if we are to know where we may file as well as where to find the required items. JOSS provides a basic index to the contents of a file in response to the command

```
Type item-list.
ITEM  CODE     RPN      DATE    SPACE
 5    JONES    1407   12/23/66     3
 7               1    1/02/67     1
25    INDEX    1407    1/02/67     1
```

For each item in the file, JOSS will type the item number, item code (if any), date of filing, project number, and space occupied by that item. (The total space available in a file is 100.) Since this is often insufficient information to identify the file items precisely, the user is strongly urged to maintain an additional index to his file in an otherwise unused item.

When modifying an item, the original version should not be discarded until the new version has been filed and verified.

We may wish to recall parts of a program, such as formulas or values, without upsetting existing program steps in our console. This may be done by separate filing, but other arrangements are also possible. If, for example,

we wanted the values from item 1 to replace the values in our existing program, we could

```
File all as item 20.
Done.
Delete all values.
File all as item 21.
Done.
```

which files our existing program both with and without values. To file the values from item 1 separately, we

```
Delete all.
Recall item 1.
Done.
File all values as item 22.
Done.
Delete all.
```

At this point we have filed

- The existing program as item 20, and
- The existing program without values as item 21, and
- The values from item 1 as item 22.

This example, which presents only one of several ways to replace values, is primarily intended to indicate how information may be sifted and sorted. Items without a future use should be discarded as soon as they are no longer needed.

Appendix A

JOSS COMMANDS

A.1. SOME RULES OF FORM

A command to JOSS takes the form of an imperative English sentence, obeying the standard rules for spelling, capitalization, punctuation, and spacing. "Eh?" indicates that one or more of the following rules has been violated.

- Limit commands to those listed.
- Capitalize the first word only.
- Insert a period at the end of each command.
- Use commas to separate items in a list.
- Space before and after each English word and before the first bracket of a file or item code. No space is allowed within a word or number, nor before the first bracket of a subscript or function argument. Spaces are optional elsewhere.
- Follow the rules for expressions given in App. B.

A.2. NOTES APPLICABLE TO THE COMMANDS

Commands are represented in general form in this appendix. The following notes apply:

- Single-letter identifiers may be used in place of f, x, and y.
- Indexed variables may be used in place of x and y.
- Single-letter dummies only may be used in place of a, b, and c in function definitions.
- ☐ indicates that an expression may be used (App. B).
- The if phrase may follow any command. The command

will be ignored unless the if phrase is satisfied.

- Expressions for part and form numbers (and the index of an indexed variable) must equate to integers.
- Strikeover with # deletes a character.
- JOSS ignores lines (other than the body of a form) begun or ended with *.
- $ is a single character which always carries the value of the current line number.
- (Do commands) and (Cancel) may be given directly in parentheses (Sec. 4.8).

A.3. OPERATIONAL COMMANDS

Some commands are limited to direct use only and others to indirect use only, but most may be used either way. This section groups the commands according to the category in which they fall and references their discussion in the text.

Direct or Indirect Commands	Section Reference
Set x= □.	2.3
Set x= ... a proposition.	2.9
Let f= ... a formula.	2.4
Let f= ... a proposition.	2.7
Let f(a,b,c)= ... a function of a,b,c.	2.5
General Rule: Use Let directly.	
Type □, □, □, _.	2.2-2.7
Type □, □ in form □.	2.2, 3.4
Type formula f.	2.4, 2.5
Type "ABCDE".	3.4
Type step □.	4.7

Direct or Indirect Commands (cont.)	Section Reference (cont.)
Type part □.	4.7
Type form □.	4.6, 4.7
Type all steps.	4.7
Type all parts.	4.7
Type all forms.	4.7
Type all formulas.	4.7
Type all values.	4.7
Type all.	4.7, 5.1
Type size.	4.7
Type time.	
Type users.	
Do step □.	3.2
Do step □ for x=□,□,□(□)□.	3.3
Do step □, □ times.	3.3
Do part □.	3.2
Do part □ for x=□(□)□(□)□.	3.3
Do part □, □ times.	3.3
Line.	4.6
Page.	4.6
Delete step □.	2.1
Delete part □.	2.1
Delete form □.	2.1
Delete all steps.	2.1
Delete all parts.	2.1
Delete all forms.	2.1
Delete x,y.	2.1
Delete all values.	2.1

Direct or Indirect Commands (cont.)	Section Reference (cont.)
Delete all formulas.	2.1
Delete all.	2.1
Quit.	3.7

Direct Only Commands	Section Reference
Go.	1.3, 4.7
Cancel.	4.7

Indirect Only Commands	Section Reference
1.1 To step □.	3.2
1.1 To part □.	3.2
1.1 Done.	3.7
1.1 Stop.	3.7
1.1 Demand x.	3.3

A.4. FILE COMMANDS

File commands follow the same rules as the operational commands. The format for item and file designators is an integer, a space, and a parenthetic code. File codes are 5 characters; item codes are from 1 to 5 characters. See Chap. 5 for rules of use.

Direct or Indirect Commands	Section Reference
Use file □ (code).	5.3
File f,x,y as item □ (code).	5.3

Direct or Indirect Commands (cont.)	Section Reference (cont.)
File formula f as item ☐ (code).	5.3
File step ☐ as item ☐ (code).	5.3
File part ☐ as item ☐ (code).	5.3
File form ☐ as item ☐ (code).	5.3
File all steps as item ☐ (code).	5.3
File all parts as item ☐ (code).	5.3
File all forms as item ☐ (code).	5.3
File all formulas as item ☐ (code).	5.3
File all values as item ☐ (code).	5.3
File all as item ☐ (code).	5.3
Recall item ☐ (code).	5.3
Discard item ☐ (code).	5.3
Type item-list.	5.3

A.5. DIRECT INPUTS

JOSS recognizes, in addition to the above commands, the following direct-only methods of entering values and forms:

Variable Definition	Section Reference
x= ☐	2.3
x= ... a proposition	2.9

Form Definition	Section Reference
Form ☐: distance:....., accel: ___.___	3.4

Appendix B

EXPRESSIONS AND PROPOSITIONS

B.1. EXPRESSIONS

In this text, the term expression (used alone), or the symbol □, means a mathematical expression that is reducible to a number at the time JOSS interprets that expression; Boolean expressions, or propositions, are excluded. Generally, a number may be written as an expression anywhere in the language except as the step label on an indirect command.

Expressions are made up of one number or identifier or a combination of numbers and/or identifiers and/or JOSS functions related by the usual mathematical symbols. The symbols are conventional, except for the asterisk, which indicates exponentiation.

Listed in their operational order of precedence, the symbols are

1. | |, [ ], ( )
2. *
3. ·, /
4. +, -

Conditional expressions are written

(prop: □; prop: □; □)

and take the value of the expression □ following the first true proposition (Sec. B.3).

B.2. SOME RULES OF FORM FOR EXPRESSIONS

Expressions are written in standard mathematical form, linearized to place the expression on a single line, following the order of precedence of the above section. "Eh?" indicates a violation of one or more of the following rules:

- Each bracket, parenthesis, and absolute value sign must have a matching opposing member.
- The lower-case el (l) may not be used for the one (1); the upper-case oh (O) may not be used for the zero (0).
- The dot (·) must be used to indicate multiplication.
- Spaces may not appear within numbers, between a function and its argument bracket, and between a variable and its index bracket.
- Single-letter identifiers only may be used.
- Identifier subscript use must be consistent with the identifier definition.

B.3. PROPOSITIONS

Propositions (Boolean expressions) are composed of expressions related by

$$=,\ \neq,\ \leq,\ \geq,\ <,\ >$$

and the negation

not.

If there is more than one relational statement, they are joined by the logical operators

and, or.

Each proposition has a logical value of true or false.

The order of precedence of operations within a proposition is

1. evaluation of expressions (Sec. B.1)
2. ( ) from inside outward
3. relational operations
4. not
5. and
6. or

Appendix C

FUNCTIONS

C.1. JOSS FUNCTIONS

The available JOSS functions, along with explanations of their use, are listed in this appendix. Rules of form are covered in Sec. B.2.

Function	Explanation
sqrt(□)	Square root. The argument must have a value ≥ zero.
log(□)	Natural logarithm. The argument must have a value > zero.
exp(□)	Exponential. The argument x must be such that $e^x < 10^{100}$. If $e^x < 10^{-99}$, the result will be zero.
sin(□) cos(□)	Sine and cosine. The argument is assumed to be in radian measure and must have a magnitude < 100.
ip(□)	Integer part. ip(-22026.4658) = -22026
fp(□)	Fraction part. fp(-22026.4658) = -.4658
dp(□)	Digit part. dp(-22026.4658) = -2.20264658
xp(□)	Exponent part. xp(-22026.4658) = 4
sgn(□)	The value of the signum function is +1 for an argument greater than zero, 0 for an argument equal to zero, and -1 for an argument less than zero.
tv(prop)	The truth value function converts its argument, which is a proposition, into the number 1 if p is true, and 0 if p is false.

Function	Explanation
arg(□,□)	The argument function requires two arguments, separated by a comma, and as a result gives the angle between the positive x axis of the x,y plane and the line joining the point (0,0) and a point (x,y). The result is in radian measure, $-\pi < \arg(□,□) \leq \pi$. The value of arg(0,0) is zero.
sum(i=□(□)□: ... some function of i ...)	
prod(i=□(□)□: ... some function of i ...)	
max(i=□(□)□: ... some function of i ...)	
min(i=□(□)□: ... some function of i ...)	
first(i=□(□)□: ... some proposition in i ...)	The five iterative functions, sum, product, maximum, minimum, and first, take two arguments: an iteration expression such as that in a for clause, and, separated by a colon, the expression (or proposition) to which the iteration is to apply. The variable of iteration i is a dummy; no real identifier is affected in any way. For all but the first function, the indicated operation is applied repeatedly, and the function gives the resulting numeric value. (For these functions, arguments may also be given in list fashion: sum(□, □, □, □, □) etc.) In contrast, the first function gives the first value of the index i for which the proposition is satisfied.

C.2. USER-DEFINED FUNCTIONS

We may define other functions, with 1 to 10 arguments, using the Let command. Any single-letter identifier may represent the function, and any distinct single letters may be used as dummies. The use of the letters as dummies does not affect their concurrent use as identifiers.

A few commonly used functions are listed below, but the number of possibilities is unlimited. Any convenient identifier may be substituted in place of f, D, I, S, and L. The arguments (x), (f,a,b) are dummies. (See Secs. 2.5 and 2.8.)

Tangent

```
Let f(x)=sin(x)/cos(x).
```

Arc sin

```
Let f(x)=arg(sqrt(1-x*2),x).
```

Arc cos

```
Let f(x)=arg(x,sqrt(1-x*2)).
```

Arc tan

```
Let f(x)=arg(1,x).
```

Arc sec

```
Let f(x)=arg(1/x,sqrt(1-1/x*2)).
```

Arc csc

```
Let f(x)=arg(sqrt(1-1/x*2),1/x).
```

Arc cot

```
Let f(x)=arg(x,1).
```

Derivative of a function of a variable

```
Let D(f,x)=[f(x+.0001)-f(x)]/.0001.
```

(Note: Depending on the function f, another value may be needed in place of .0001.)

Integral of a function from lower limit to upper limit

```
Let I(f,a,b)=(b-a)/n/3·(f[a]+f[b]+S[f,a,b]).
Let S(f,a,b)=sum(i=1(1)n-1: [3-(-1)*i]·f[a+i·(b-a)/n]).
n=100
```

(Note: n must be even.)

Log to base 10

```
Let L(x)=log(x)/log(10).
```

Factorial

```
Let f(x)=[x=0: 1; fp(x)=0: prod(i=1(1)x: i)].
```

C.3. CONSTANTS DERIVED FROM THE FUNCTIONS

The constant pi may be defined by

```
p=arg(-1,0)
```

Another frequently used constant, for converting degrees to radians, may then be defined by

```
k=p/180
```

or if p has not been defined as pi, the radians/degree constant is defined by

```
k=arg(-1,0)/180
```

JOSS gives us the constant e, the base of natural logarithms, directly through the exponential function, exp(1).

```
←TYPE ALL.

1.00 PAGE.
1.01 DEMAND V AS "STARTING VELOCITY".
1.02 DEMAND A AS "LOOP G-VALUE".
1.03 DEMAND L AS "PRINTOUT TIME STEP (IN SECONDS)".
1.04 SET L=IP(10&L)/10.
1.05 TO STEP 1.03 IF L=0.
1.06 SET T=L/10.
1.07 DEMAND N AS "TIME LIMIT (IN REAL-TIME MINUTES)".
1.08 SET J=FALSE.
1.09 DEMAND K AS "IS ENERGY CONSERVED?  (TRUE/FALSE)".
1.10 SET J=TRUE IF (K=TRUE).
1.11 LINE.
1.12 SET X=0.
1.13 SET Z=0.
1.14 SET Q(1)=0.
1.15 SET S=0.
1.16 SET U=FALSE.
1.17 PAGE.
1.18 TYPE V, A IN FORM 1.
1.19 TYPE "    ENERGY IS CONSERVED." IF J.
1.20 TYPE "    CONSTANT-VELOCITY LOOP." IF NOT J.
1.21 LINE.
1.22 SET E=V&V/2 IF J.
1.23 RESET TIMER.
1.24 DO PART 2 FOR I=1(1)1000.
1.25 LINE.
1.26 TYPE TIMER IF TIMER>N.
1.27 LINEIF TIMER>N.
1.28 TO STEP 1.00.

2.00 SET C=F(Q(1)).
2.01 SET Q(2)=Q(1)+T&V/C.
2.02 SET R=(C+F(Q(2)))/2.
2.03 SET Q(2)=Q(1)+T&V/R.
2.04 SET R=(C+F(Q(2)))/2.
2.05 SET X=X+R&(SIN(Q(2))-SIN(Q(1))).
2.06 SAT Z=Z+R&(COS(Q(1))-COS(Q(2))).
2.07 SET Q(1)=Q(2).
2.08 DO PART 3.
2.09 SET S]S+T.
2.10 SET V=SQRT(E+E-G&Z) IF J.
2.11 TO STEP 2.15 IF M(S,L)>0.
2.12 TYPE S, X, Z, R, B(Q(1)), V IN FORM 2.
2.13 SET U=TRUE IF Q(1)<0.
2.14 QUIT IF (U AND Q(1)>0 AND X>0).
2.15 QUIT IF ($>50 OR TIMER>N).

3.00 TO STEP 3.03 IF Q(1)<W.
3.01 SET Q(1)=Q(1)-2&W.
3.02 DONE.
3.03 DONE IF Q(1)>-W.
3.04 SET Q(1)=Q(1)+2&W.
```

```
3.05 DONE.

FORM 1:
   SPEED = ↑↑↑↑.↑↑ METERS/SECOND.   ↑↑↑.↑↑ "G" LOOP.

FORM 2:
  ↑↑↑↑.↑ S    ↑↑↑↑↑.↑ X    ↑↑↑↑↑.↑ Z    ↑↑↑↑↑↑.↑ R    ↑↑↑↑.↑ DEG ↑↑↑↑.↑ M/

         B(X):   X/0.017453294
       M(X,Q):   V*V/(R*(X-COS(Q)))

             A =          2
             C =      1019.63959
             E =      5000
             G =         9.807
             I =       370
             J =       FALSE
             K =       FALSE
             L =         1
             N =         5
             R =      1019.65949
             S =        37
             T =           .1
             U =       TRUE
             V =       100
             W =         3.14159292
             X =       992.035137
             Z =         4.527648&10*(-4)

          Q(1) =         9.2362363&10*(-4)
          Q(2) =         9.2362363&10*(-4)

←PAGE.

1154   4/15/71   #11   HMM  813    [19]
```

STARTING VELOCITY = ←100
LOOP G-VALUE = ←2
PRINTOUT TIME STEP (IN SECONDS) = ←1
TIME LIMIT (IN REAL-TIME MINUTES) = ←5
IS ENERGY CONSERVED? (TRUE/FALSE) = ←TRUE.

1138 4/15/71 #11 HMM 913 [12]

SPEED = 100.00 METERS/SACOND. 2.00 "G" LOOP.
ENERGY IS CONSERVED.

1.0	S	99.8	X	4.9	Z	1011.3	R	5.6	DEG	99.8	M/S
2.0	S	198.1	X	19.6	Z	983.8	R	11.3	DEG	99.0	M/S
3.0	S	293.6	X	43.8	Z	938.1	R	17.2	DEG	97.8	M/S
4.0	S	384.6	X	77.4	Z	876.0	R	23.4	DEG	96.1	M/S
5.0	S	469.7	X	120.0	Z	800.1	R	29.9	DEG	93.9	M/S
6.0	S	547.1	X	171.0	Z	713.4	R	37.0	DEG	91.2	M/S
7.0	S	615.1	X	229.6	Z	619.7	R	44.7	DEG	88.0	M/S
8.0	S	671.7	X	294.8	Z	523.1	R	53.5	DEG	84.3	M/S
9.0	S	714.9	X	364.9	Z	428.1	R	63.6	DEG	80.1	M/S
10.0	S	742.5	X	437.8	Z	339.1	R	75.4	DEG	75.5	M/S
11.0	S	752.3	X	510.3	Z	260.6	R	89.7	DEG	70.7	M/S
12.0	S	742.7	X	577.8	Z	196.4	R	107.3	DEG	65.8	M/S
13.0	S	713.2	X	634.0	Z	149.4	R	129.1	DEG	61.5	M/S
14.0	S	666.5	X	670.9	Z	121.3	R	155.1	DEG	58.5	M/S
15.0	S	610.2	X	691.4	Z	112.7	R	-176.0	DEG	57.6	M/S
16.0	S	555.7	X	693.3	Z	123.7	R	-147.6	DEG	59.1	M/S
17.0	S	513.5	X	620.5	Z	154.2	R	-122.5	DEG	62.6	M/S
18.0	S	489.7	X	560.9	Z	203.6	R	-101.9	DEG	67.1	M/S
19.0	S	485.9	X	492.0	Z	269.8	R	-85.2	DEG	71.9	M/S
20.0	S	501.2	X	419.6	Z	349.9	R	-71.6	DEG	76.7	M/S
21.0	S	533.6	X	348.0	Z	439.8	R	-60.2	DEG	81.2	M/S
22.0	S	580.9	X	279.9	Z	535.0	R	-50.5	DEG	85.2	M/S
23.0	S	640.9	X	217.3	Z	631.0	R	-42.0	DEG	88.7	M/S
24.0	S	711.7	X	161.7	Z	723.2	R	-34.5	DEG	91.7	M/S
25.0	S	791.4	X	113.9	Z	807.5	R	-27.6	DEG	94.2	M/S
26.0	S	878.1	X	74.8	Z	880.4	R	-21.1	DEG	96.3	M/S
27.0	S	970.2	X	44.8	Z	938.9	R	-15.0	DEG	978	M/S
28.0	S	1066.3	X	24.2	Z	980.5	R	-9.2	DEG	98.8	M/S
29.0	S	1164.7	X	13.2	Z	1003.6	R	-3.5	DEG	99.3	M/S
30.0	S	1264.1	X	12.1	Z	1007.4	R	2.2	DEG	99.4	M/S

```
STARTING VELOCITY = ←100
LOOP G-VALU = ←2
PRINTOUT TIME STEP (IN SECONDS) = ←1
TIME LIMIT (IN REAL-TIME MINUTES) = ←5
IS, ENERGY CONSERVED?  (TRUE/FALSE) = ←FALSE'

1144   4/15/71   #11   PMM  913   [15]
```

SPEED = 100.00 METERS/SECOND. 2.00 "G" LOOP.
CONSTANT-VELOCITY LOOP.

	1.0	S	99.8	X	4.9	Z	1015.3	R	5.6 DEG 100.0 M/S
	2.0	S	199.7	X	19.6	Z	1001.2	R	11.3 DEG 100.0 M/S
	3.0	S	295.6	X	44.1	Z	977.9	R	17,1 DEG 100.0 M/S
	4.0	S	389.5	X	78.4	Z	946.0	R	23.1 DEG 100,0 M/S
	5.0	S	479.2	X	122.5	Z	906.4	R	29.3 DEG 100.0 M/S
	6.0	S	563.5	X	176.1	Z	860.4	R	35.8 DEG 100.0 M/S
	7.0	S	641.0	X	239.3	Z	809.1	R	42.6 DEG 100.0 M/S
	8.0	S	710.1	X	311.5	Z	754.1	R	50.0 DEG 100.0 M/S
	9.0	S	768.9	X	392.2	Z	697.0	R	57.9 DEG 100.0 M/S
	10.0	S	815.5	X	480.6	Z	639.3	R	66.6 DEG 100.0 M/S
	11.0	S	847.7	X	575.2	Z	582.9	R	76.0 DEG 100.0 M/S
S	12.0	S	863.2	X	673.8	Z	529.2	R	86.3 DEG 100.0
	13.0	S	859.8	X	773.6	Z	479.9	R	97.8 DEG 100.0 M/S
	14.0	S	835.7	X	870.4	Z	436.3	R	110.4 DEG 100.0 M/S
	15.0	S	790.1	X	959.2	Z	399.7	R	124.2 DEG 100.0 M/S
	16.0	S	724.0	X	1033.8	Z	371.2	R	139.1 DEG 100.0 M/S
	17.0	S	640.4	X	1088.0	Z	351.5	R	155.1 DEG 100.0 M/S
	18.0	S	544.9	X	1116.6	Z	341.4	R	171.7 DEG 100.0 M/S
	19.0	S	445.3	X	1116.5	Z	340.9	R	-171.5 DEG 100.0 M/S
	20.0	C	349.9	X	1087.6	Z	350.2	R	-154.9 DEG 100.0 M/S
	21.0	C@	266.4	X	1033.2	Z	369.0	R	-139.0 DEG 100.0 M/S
	22.0	S	200.4	X	959.4	Z	396.8	R	-124.0 DEG 100.0 M/S
	23.0	S	155.1	X	869.5	Z	432.6	R	-110.2 DEG 100.0 M/S
	24.0	S	131.2	X	772.6	Z	475.6	R	-97.7 DEG 100.0 M/S
	25.0	S	127.9	X	672.9	Z	524.5	R	-86.2 DEG 100.0 M/S
	26.0	S	143.6	X	574.2	Z	577.9	R	-75.9 DEG 100.0 M/S
	27.0	S	175.9	X	478.7	Z	634.1	R	-66.5 DEG 100.0 M/S
	28.0	S	222.7	X	391.4	Z	691.7	R	-57.9 DEG 100.0 M/S
	29.0	S	281.6	X	310.8	Z	749.0	R	-49.9 DEG 100.0 M/S
	30.0	@S	350.8	X	238.6	Z	804.2	R	-42.6 DEG 100.0 M/S
	31.0	S	428.3	X	175.6	Z	855.9	R	-35.7 DEG 100.0 M/S
	32.0	S	512.7	X	122.0	Z	902.5	R	-29.2 DEG 100.0 M/S
	33.0	S	602.5	X	79.1	Z	942.7	R	-23.0 DEG 100.0 M/S
	34.0	S	696.4	X	43.9	Z	975.3	R	-17.0 DEG 100.0 M/S
	35,0	S	793.3	X	1[illegible]4	Z	999.5	R	-11.3 DEG 100.0 M/S
	36.0	S	892.2	X	4.9	Z	1014.4	R	-5.6 DEG 100.0 M/S
	37.0	S	990.0	X	.0	Z	1019.7	R	.1 DEG 100.0 M/S

```
STARTING VELOCITY = ←100
LOOP G-VALUE = ←2
PRINTOUT TIME STEP (IN SECONDS) = ←1
TIME LIMIT (IN REAL-TIME MINUTES) = ←5
IS ENERGY CONSERVED?  (TRUE/FALSE) = ←TRUE


1156   4/15/71   #11   HMM  813    [21]


   SPEED =  100.00 METERS/SECOND.     2.00 "G" LOOP.
   ENERGY IS CONSERVED.

1.221 SET V=-V.
1.222 SET Q(1)=-W.
     1.0 S     -79.4 X      -9.4 Z      345.1 R   -166.6 DEG  100.5 M/S
     2.0 S    -173.1 X     -46.6 Z      369.1 R   -150.3 DEG  102.3 M/S
     3.0 S    -255.1 X    -109.3 Z      412.7 R   -135.2 DEG  105.2 M/S
     4.0 S    -320.9 X    -193.2 Z      475.6 R   -121.4 DEG  109.1 M/S
     5.0 S    -367.9 X    -293.5 Z      557.3 R   -109.1 DEG  113.5 M/S
     6.0 S    -394.9 X    -405.7 Z      656.8 R    -98.3 DEG  118.2 M/S
     7.0 S    -401.9 X    -525.8 Z      772.6 R    -88.7 DEG  123.1 M/S
     8.0 S    -389.4 X    -650.3 Z      903.0 R    -80.1 DEG  128.0 M/S
     9.0 S    -358.4 X    -776.6 Z     1045.8 R    -72.5 DEG  132.7 M/S
    10.0 S    -310.2 X    -902.4 Z     1198.7 R    -65.7 DEG  137.3 M/S
    11.0 S    -245.9 X   -1025.9 Z     1359.0 R    -59.5 DEG  141.6 M/S
    12.0 S    -166.9 X   -1145.6 Z     1524.1 R    -53.8 DEG  145.7 MS
    13.0 S     -74.3 X   -1260.3 Z     1691.5 R    -48.5 DEG  149.5 M/S
    14.0 S      30.5 X   -1369.2 X     1858.4 R    -43.7 DEG  153.1 M/S
    15.0 S     146.4 X   -1471.3 Z     2022.3 R    -39.1 DEG  156.3 M/S
    16.0 S     272.3 X   -1566.1 Z     2180.7 R    -34.9 DEG  159.2 M/S
    17.0 S     407.2 X   -1653.0 Z     2331.4 R    -30.8 DEG  161.9 M/S
    18.0 S     549.9 X   -1731.6 Z     2472.2 R    -26.9 DEG  164.3 M/S
    19.0 S     699.5 X   -1801.6 Z     2601.0 R    -23.2 DEG  166.3 M/S
    20.0 S     855.2 X   -1862.5 Z     2716.2 R    -19.6 DEG  168.1 M/S
    21.0 S    1015.8 X   -1914.3 Z     2816.1 R    -16.1 DEG  169.6 M/S
    22.0 S    1180.7 X   -1956.6 Z     2899.5 R    -12.7 DEG  170.8 M/S
    23.0 S    1348.8 X   -1989.4 Z     2965.2 R     -9.4 DG   171.8 M/S
    24.0 S    1519.3 X   -2012.5 Z     3012.5 R     -6.1 DEG  172.4 M/S
    25.0 S    1691.4 X   -2025.9 Z     3040.6 R     -2.8 DEG  172.8 M/S
    26.0 S    1864.2 X   -2029.5 Z     3049.2 R       .4 DEG  172.9 M/S


1159   4/15/71   #11   HMM  813    [22]
```

STARTING VELOCITY = ←100
LOOP G-VALUE = ←2
PRINTOUT TIME STEP (IN SECONDS) = ←1
TIME LIMIT (IN REAL-TIM MINUTES) = ←5
I-S ENEPGY CONSERVED? (TRUE/FALSE) = ←TRUE

1211 4/15/71 #11 HMM 813 [31]

SPEED = 100.00 METERS/SECOND. 2.00 "G" LOOP.
ENERGY IS CONSERVED.

	1.0	S	99.9	X	4.9	Z	1011.3	R	5.6 DEG 99.8 M/S
	2.0	S	198.1	X	19.6	Z	983.8	R	11.3 DEG 98.0 M/S
	3.0	S	293.6	X	43.9	Z	938.1	R	17.2 DEG 97.8 M/S
	4.0	S	384.6	X	77.4	Z	876.0	R	23.4 DEG 96.1 M/S
	5.0	S	469.7	X	120.0	Z	800.1	R	29.9 DEG 93.9 M/S
	6.0	S	547.1	X	171.0	Z	713.4	R	37.0 DEG 91.2 M/S
	7.0	S	615.1	X	229.6	Z	619.7	R	44.7 DEG 88.0 M/S
	8.0	S	671.7	X	294.8	Z	523.1	R	53.5 DEG 84.3 M/S
	9.0	S	714.9	X	364.9	Z	428.1	R	63.6 DEG 80.1 M/S
	10.0	S	742.5	X	437.9	Z	339.1	R	75.4 DEG 75.5 M/S
"06" →	11.0	S	752.3	X	510.3	Z	260.6	R	89.7 DEG 70.7 M/S
	12.0	S	750.5	X	585.7	Z	15887.4	R	91.6 DEG 65.2 M/S
	13.0	S	748.6	X	648.7	Z	11645.6	R	91.9 DEG 60.3 M/S
	14.0	S	746.5	X	706.8	Z	8315.4	R	92.2 DEG 55.4 M/S
	15.0	S	744.3	X	759.9	Z	5756.1	R	92.7 DEG 50.5 M/S
	16.0	S	741.8	X	808.1	Z	3838.6	R	93.3 DEG 45.6 M/S
	17.0	S	739.1	X	851.3	Z	2445.7	R	94.1 DEG 40.6 M/S
	18.0	S	735.9	X	889.6	Z	1471.8	R	95.3 DEG 35.7 M/S
	19.0	S	732.4	X	922.9	Z	823.3	R	97.1 DEG 30.8 M/S
	20.0	S	708.2	X	951.2	Z	418.3	R	100.0 DEG 25.9 M/S
	21.0	S	723.2	X	974.4	Z	186.7	R	105.0 DEG 21.1 M/S
	22.0	S	717.0	X	992.3	Z	70.3	R	115.0 DEG 16.4 M/S
	23.0	S	708.9	X	1004.2	Z	22.9	R	137.3 DEG 12.3 M/S
	24.0	S	698.8	X	1008.3	Z	11.4	R	-177.1 DEG 10.6 M/S
	25.0	S	689.1	X	1003.3	Z	22.0	R	-133.3 DEG 12.7 M/S
	26.0	S	681.7	X	991.0	Z	68.6	R	-112.7 DEG 16.8 M/S
	27.0	S	676.1	X	973.0	Z	185.9	R	-103.6 DEG 21.4 M/S
	28.0	S	671.6	X	950.0	Z	423.4	R	-99.0 DEG 26.1 M/S
	29.0	S	667.9	X	921.9	Z	844.5	R	-96.3 DEG 31.0 M/S
	30.0	S	664.8	X	888.9	Z	1526.4	R	-94.7 DEG 35.8 M/S
	31.0	S	662.0	X	851.0	Z	2559.8	R	-93.6 DEG 40.7 M/S
	32.0	S	659.6	X	808.2	Z	4049.1	R	-92.9 DEG 45.5 M/S
	33.0	S	657.4	X	760.6	Z	6112.4	R	-92.4 DEG 50.4 M/S
	34.0	S	655.5	X	709.0	Z	8881.6	R	-92.0 DG 55.3 M/S
	35.0	S	653.7	X	650.5	Z	12502.3	R	-91.7 DEG 60.2 M/S
	36.0	S	652.0	X	588.2	Z	17133.8	R	-91.4 DEG 65.0 M/S
	37.0	S	650.5	X	521.0	Z	22949.1	R	-91.2 DEG 69.9 M/S
	38.0	S	649.0	X	448.9	Z	30134.9	R	-91.1 DEG 74.8 M/S
S	39.0	S	647.7	X	371.9	Z	38891.9	B	-90.9 DEG 79.7
	40.0	S	646.4	X	290.0	Z	49434.1	R	-90.8 DEG 84.6 M/S
	41.0	S	645.2	X	203.2	Z	61989.7	R	-90.7 DEG 89.5 M/S
	42.0	S@	644.1	X	111.5	Z	76800.5	R	-90.7 DEG 94.4 M/S

TIMER = 5.18

```
STARTING VELOCITY = ←
I'M AT STEP 1.01.
←DELETE ALL.
←RECALL ITEM 4 (TURN).
DOE.
←TYPE ALL.

1.00 PAGE.
1.005 SET J=0.
1.006 SET T=0.
1.01 DEMAND B AS "HEADING AZIMUTH (DEGREES)" IF J=0.
1.02 DEMAND V AS "SPEED (METERS/SECOND)" IF J=0.
1.03 DEMAND A AS "TURNING G".
1.031 TO STEP 1.07 IF A=0.
1.035 SE A=A&G.
1.04 TO STEP 1.03 IF G>A.
1.051 SET L=FALSE.
1.052 SET R=TRUE.
1.053 DEMAND R AS "TURN DIRECTION?  (L=LEFT, R=RIGHT)".
1.06 SET R=FALSE IF R%TRUE.
1.07 LINE.
1.08 DEMAND D AS "TIME INCREMENT".
1.09 DEMAND M AS "NUMBER OF TIME STEPS".
1.10 SET M=IP(M+0.5).
1.11 TO STEP 1.09 IF M<1.
1.13 DEMAND X AS "STARTING X COORDINATE" IF J=0.
1.14 DEMAND Y AS "STARTING Y COORDINATE" IF J=0.
1.15 LINE.
1.16 LINE.
1.17 SET Z=0.0174532934&B.
1.18 TYPE B IN FORM 1.
1.19 TYPE V IN FORM 2.
1.191 TYPE "STRAIGHT FLIGHT" IF A=0.
1.192 TO STEP 1.23 IF A=0.
1.20 TYPE A/G IN FORM 3.
1.21 TYPE "RIGHT TURN." IF R.
1.22 TYPE "LEFT TURN." IF (NOT R).
1.221 TYPE S IN FORM 31.
1.222 TYPE (1.5707963-F(G/A))/0.0174532934 IN FORM 32.
1.23 LINE.
1.24 LINE.
1.25 TYPE J, T, B, X, Y IN FORM 4.
1.251 TO STEP 1.32 IF A=0.
1.26 TO STEP 1.30 IF R.
1.27 DO PART 3 FOR I=1(1)M.
1.28 TO STEP 1.33.
1.29 *.
1.30 DO PART 2 FOR I=1(1)M.
1.31 TO STEP 1.33.
1.32 DO PART 5 FOR I=1(1)M.
1.33 LINE.
1.34 PAGE.
1.35 TO STEP 1.01.
```

```
2.00 SET J=J+1.
2.01 SET T=T+D.
2.02 SET C=V&D/S.
2.03 SET X=X+S&(COS(Z)-COS(Z+C)).
2.04 SET Y=Y+S&(SIN(Z+C)-SIN(Z)).
2.05 SET Z=Z+C.
2.06 DO PART 4.

3.00 SET J=J+1.
3.01 SET T=T+D.
3.02 SET C=V&D/S.
3.03 SET X=X-S&(COS(-Z)-COS(C-Z)).
3.04 SET Y=Y+S&(SIN(C-Z)-SIN(-Z)).
3.05 SET Z=Z-C.
3.06 DO PART 4.

4.00 TO STEP 4.03 IF Z@0.
4.01 SET Z=Z+6.28318584.
4.02 TO STEP 4.00.
4.03 TO STEP 4.06 IF Z<6.28318585.
4.04 SET Z=Z-6.28318584.
4.05 TO STEP 4.03.
4.06 SET B=Z/0.01745329S4.
4.07 TYPE J, T, B, X, Y IN FORM 4.
4.08 PAGE IF S>40.

5.00 SET J=J+1.
5.01 SET T=T+D.
5.02 SET X=X+V&D&SIN(Z).
5.03 SET Y=Y+V&D&COS(Z).
5.04 DO PART 4.

FORM 1:
HEADING AZIMUTH = ↑↑↑.↑↑ DEGREES.

FORM 2:
SPEED = ↑↑↑↑↑.↑↑ METERS/SECOND.

FORM 3:
TURNING G VALUE = ↑↑.↑↑.

FORM 4:
 ↑↑↑   ↑↑↑.↑↑ S   ↑↑↑.↑↑ DEG   ↑↑↑↑↑↑.↑↑ X   ↑↑↑↑↑↑.↑↑ Y

FORM 31:
TURNING RADIUS = ↑↑↑↑↑↑↑↑.↑↑ METERS.

FORM 32:
BANK ANGLE = ↑↑↑↑.↑ DEGREES.

        F(X):  ARG(SQRT(1-X*2),X)
           P:  F(1.57079646-G/A)
           S:  V*2/(A&SIN(1.57079646-F(G/A)))
```

```
A =          0
B =       188.04
C =          .166430317
D =         40
G =          9.807
I =          1
J =          1
L =       FALSE
M =          1
R =       TRUE
T =         40
V =        500
X =       6436.99642
Y =       1748.4274
Z =          3.2815174
```

←
1205 4/15/71 #11 HMK 913 [25]

```
←DO PART 1.

1205   4/15/71   #11  HMM  813   [26]



HEADING AZIMUTH (DEGREES) = ←0
SPEED (METERS/SECOND) = ←100
TURNING G = ←2
TURN DIRECTION?  (L=LEFT, R=RIGHT) = ←R

TIME INCREMENT = ←1
NUMBER OF TIME STEPS = ←5
STARTING X COORDINATE = ←0
STARTING Y COORDINATE = ←0


HEADING AZIMUTH =     .00 DEGREES.
SPEED =    100.00 METERS/SECOND.
TURNING G VALUE =  2.00.
RIGHT TURN.
TURNING RADIUS =      588.71 METERS.
BANK ANGLE =   60.0 DEGREES.

   0      .00 S      .00 DEG        .00 X        .00 Y
   1     1.00 S     9.73 DEG       8.47 X      99.52 Y
   2     2.00 S    19.46 DEG      33.65 X     196.18 Y
   3     3.00 S    29.20 DEG      74.90 X     287.18 Y
   4     4.00 S    38.93 DEG     130.74 X     369.93 Y
   5     5.00 S    48.66 DEG     199.87 X     442.02 Y


1207   4/15/71   #11  HMM  813   [27]



TURNING G = ←0

TIME INCREMENT = ←5
NUMBER OF TIME STEPS = ←1


HEADING AZIMUTH =  48.66 DEGREES.
SPEED =   100.00 METERS/SECOND.
STRAIGHT FLIGHT

   5     5.00 S    48.66 DEG     199.87 X     442.02 Y
   6    10.00 S    48.66 DEG     575.28 X     772.27 Y
```

```
TURNING G = ←2
TURN DIRECTION?  (L=LEFT, R=RIGHT) = ←L

TIME INCREMENT = ←1
NUMBER OF TIME STEPS = ←5


HEADING AZIMUTH =   45.66 DEGREES.
SPEED =    100.00 METERS/SECOND.
TURNING G VALUE =   2.00.
LEFT TURN.
TURNING RADIUS =       588.71 METERS
BANK ANGLE =    60.0 DEGREES.

   6     10.00 S     45.66 DEG      575.28 X      772.27Y
   7     11.00 S     38.93 DEG      644.41 X      844.36 Y
   8     12.00 S     29.20 DEG      700.35 X      927.11 Y
   9     13.00 S     19.46 DEG      741.50 X     1018.12 Y
  10     14.00 S      9.73 DEG      766.68 X     1114.77 Y
  11     15.00 S       .00 DEG      775.15 X     1214.29 Y


1209   4/15/71   #11  HMM  913    [29]



TURNING G = ←
I'M AT STEP 1.03.
←DELETE ALL.
←U*
←RECALL ITEM 5 (LOOP).
DONE.
←.001 SET A=0 IF Q(1)>W/2.
←DO PART 1.
```