Computers for the Arts

DICK HIGGINS

Computers are like most tools—deaf, blind and incredibly stupid. So stupid, in fact, that they cannot imagine how to make a mistake once they are programmed to do what is expected of them. This makes them different from other tools. Imagine a hammer which, once programmed to build a table, could do so on its own, without the possibility of damaging or splitting the wood. It would leave the carpenter free to concentrate on the design of his table with no worry about the difficulties of execution. The role which computers can play is analogous to this.

Computers when used for the arts are doing what they are not normally designed to do. Their main use is economic — in science and business. However, their uses are sufficiently versatile to justify looking into a number of the special techniques for the solution of creative problems.

We do not communicate with a computer by telling it verbally what to do. Instead we provide a structure on the basis of which it processes whatever data we provide. The artificial languages in which these programs are written vary, and some are less comprehensible to the mathematically unskilled than others. In my own experience I have found that, in spite of my lack of formal training in logic or mathematics, it is easier to pick up the basics of communicating with a computer in one of the commonest of these artificial languages, Fortran, than to try to explain my intentions to an engineer who lacks my motivations.

Suppose my project is a verbal one, resulting in a listing of words which will resemble verse. There are two parts to achieving this: (1) my materials (verbs, nouns, or whatever I happen to be using), which, for the machine, are simply

data; (2) the method of treating these materials – the program itself. Within the program I will have to define a number of variables, one of which, for example, might represent the sequential order of each line, another the number of units (words) per line. If I wish this number to vary, I will be able to calculate a value of a third variable to cause this to happen.

To compose such a program, I will have to understand the logic of the process involved, what calculations take place in what order, etc. If I wish to have five words per line, I must decide that "word(1)" represents the first word of the line, "word(2)" represents the second, and so on. "Line" would therefore represent a set of five values for "word," and would be calculated (and written out by the machine) one fifth as often. The calculation to determine the value of "word" would be logically "nested" inside the calculation to determine the value of "line."

This is, of course, paper work, and is independent of the physical nature of the computers. The artist seldom sees the computer, just as an author, traditionally, seldom sees the printing press that produces his book. The physical operation of a computer is done by an "operator," who may or may not know anything about the program he is feeding into the machine. It is the programmer who prepares the machine to function, just as the musicians and recording engineers have taken care of the necessary artistic and technical questions by the time one places a phonograph record in a turntable.

For many artistic applications the computer system can make use of "analogue devices." While the output of a digital computer consists of a printout on long sheets of paper or magnetic tape containing data, the analogue device can interpret this data in other ways. Once interpreted it can be fed back into a digital computer for further analysis. A drawback of analogue devices is that they are not always available in the main computer systems, and the more specialized machines are extremely expensive. There are times when the use of special equipment is vital. For example, a composer might connect a microphone to some form of analogue device which analyzes all the sounds present when he says "hello"; he might also have the device analyze the sounds of a string quartet. He could then feed the resulting analysis through another analogue device (a sound synthesizer), and obtain a tape or broadcast over a monitoring system of the string quartet saying "hello." Though this is so simple that one could find other, less expensive, electronic means of doing it, in more complex versions of this process it is possible that only analogue devices could solve the composer's problems.

Another interesting analogue device can scan any body of visual material, then distort it systematically along any axis, simple or curved. For example, a line drawing can be scanned, then systematically rotated to become a 45° angular distortion of its original self, and then a straight line. Such devices are used in the design of aircraft parts. But similiar analyses can be made from photographs and projected back according to the desired distortions onto a screen and rephotographed by an animation camera for use in inexpensive cartoons for television, or for special typographic effects.

Digital computers can absorb information only from special tapes which have been encoded with punches, from punched cards, or from magnetic tapes with information from previous computer runs. Usually, in communicating with them, one starts with what is called a "dimension" statement. This tells the machine how to check that it has absorbed all the required information and, in its own "memory mechanism" how much space to allot to the required program. Next it must be told what exceptions there will be to its usual methods. For example, in Fortran it is a convention that variables whose initial letters run from "I" through "N" represent integers, while all the others represent "real" num-

bers. So, if we are anxious (in the verbal project described) to use 5 units called "WORD" to a line, we must either include the statement in the program "INTEGER WORD," which causes WORD to stand for an integer, or rename the variable to be initialed like an integer, for example – "IWORD" so that the machine will know it is calculating word (1), word(2), etc, and not word (1.85), word (2.6), etc.

Next we ask the computer to read the data we are going to work with, indicating the format – e.g., the card, tapes, or both, etc. The last preliminary step is to ask the computer to write out the data and program that has been fed in, for checking.

Because the ending portions of the programs are similar to the starting procedures – they are both mechanical – I will mention them before returning to the main body of the programs. There is the WRITE statement, which usually has a form like this:

```
50  WRITE (6,102) (LINE(J), J=1, 5)
```

The "50" is simply the number of the statement in the program. The WRITE transfers the calculated information to the output. The "6" indicates an output device number which translates the binary information stored by the machine into our normal decimal and letter system. The "102" indicates that the format of the written statement is given in statement 102. The "LINE(J), J=1, 5" tells the machine what it is to write (LINE), but that LINE is actually a set containing five numbers, each indexed by a value of J that runs from one to five.

The format is defined by equally mechanical conventions. The FORMAT statement can appear anywhere in the program, so long as it has a statement number, but many programmers like to put the FORMAT statement after the WRITE statement it refers to, or to put them all in one place in the program in order to be able to find them easily and quickly. The format referred to in the sample WRITE statement might look something like this

```
102  FORMAT (10X,5A5)
```

if it were intended to produce not a four-word line, as in the illustration *Hank and Mary*, but a five-word one. The FORMAT statement is one of the most difficult things in Fortran to work out, because it is involved in the conception of "Conversion" which indicates how the binary information the machine has now calculated is to be converted into usable data. The "5A5" means that there are 5 units when LINE is written (see the WRITE statement), each of which has 5 letters. The "A" indicates "A-type conversion," which, in the Fortran language, indicates a body of Alphameric (alphabet or numbers) being fed into the machine, indexed, and, then fed out in exactly the same form the data was fed in – that is, the letter will not be reversed, and no cryptographic codes will be used to disguise them. "H" conversion in the FORMAT statement causes the letters (blanks count as characters just as numbers and letters do) following the "H" to be printed out as written in the FORMAT statement. This is important when one wants titles, but note the more significant use in *Proposition No. 2 for Emmett Williams* (see p. 000). "X" conversion simply introduces however many blanks are desired. When numerical outputs are wanted, there are not only obvious ones ("I" conversion for integers, or "F" conversion for real numbers) but very special ones for very minute decimal places, mixtures of real numbers and exponents, etc. These last, however, are seldom needed by artists.

Normally the end of a program is indicated by the statement to STOP. If one were using one program as a subprogram of another, it might say RETURN (to the program from which the sub-program departed), but this means of ending a program is somewhat less common in applications to graphics, poetry, music. The last statement of all programs is END which is a convention telling the machine that no further calculations are required of it, and it may move on to

(continued on page 11)

ISNO. For labelling, the program was called simply WORDS. "List" asks the machine to print out the program, for convenient reference or to check for error. "Ref" causes a cross-reference index of each variable.

ISN1. This DIMENSION statement asks the machine to reserve five and only five possible locations in its "memory" for variable "IPT" and four for variable "LINE."

ISN2. The fact that there are only four words and a blank is fed into the machine. The "5H" means the machine is to read and store in "memory" the next five characters of each IPT, with the blank coming before each four-letter word in order to keep them from printing too close together. Note that the first word consists of five blanks.

ISN3. This WRITE statement simply causes the machine to head the printout with the full title. The number "6" referred to is, by convention, reserved for printed output (as opposed to tape, for example). The number "100" refers to the format statement numbered 100, which causes the title to be printed, in which 1H1 simply starts a new page. "20X" means 20 blanks will be printed before the title, causing an indentation. "39H" indicates the 39 letters (counting blanks as characters) of the full title, etc.

ISN4. "J=0" simply places variable J at the starting point [of which we shall see more later.] J itself is the index to the number of lines that have been printed.

ISN5. It is at this point that we enter a nest of DO loops. N1, N2, N3 and N4 are the indices of the four DO loops.

ISN6. This statement places the five possible values of the first member of the set LINE, designated LINE(1), successively into correspondence with the five members of set IPT. ISN10, 12 and 14 do the same in their respective DO loops.

ISN7. This statement sets up the structure parallel to ISN5 for the second unit in each line, as do ISN11 and ISN13 in theirs. Note that N2 has nothing to do with N1 — the names have been made similar in order to emphasize their parallel uses.

ISN15. Each time we fill in the four members of the line, we want to see how many lines we have done, whether or not we actually want to number all the lines. "J" — as mentioned in ISN4 — indicates the number of the line. Here each time we wish to increase the number of each line by one. If, hypothetically, we wanted to increase the number of each line by two, as written (not as executed), we would write "J=J+2."

ISN16. This stage decides if the line is to be numbered. It causes the machine to test for the value of J by using an IF statement. The statement IF(J)40,45,40 (hypothetically again) would exemplify the IF structure in its simplest form, which follows the convention of providing three possible courses, depending on whether J is negative (sending the machine to statement 40), zero (sending the machine to statement 45), or positive (sending it, again, to statement 40) respectively. The statement IF(MOD(J,10)-1)40,45, 40 is a rather tricky concept, testing not just if a given value is to route the machine to statement 40, 45, or 40, but whether or not this given value, if diminished by one, is an even multiple of ten. If it is, it will cause the machine to refer to a WRITE statement that uses a FORMAT statement which includes numbering the line. If it isn't, the machine will proceed to a WRITE statement that refers to a FORMAT statement which doesn't include numbered lines.

ISN17. (statement 40 of the program). This instruction tells the machine to write out the lines per FORMAT statement 102. LINE has only four members, and these are to be written out in order, from member 1 to member 4. Note that there is no instruction to write out J.

ISN24. Having written out the line, we are ready to calculate the next line. But we must therefore by-pass the second WRITE statement which is reserved for the lines in which the value of J is listed. So we simply route the machine on to ISN32 (statement 50).

ISN25. As we noted in discussing ISN24, this is the WRITE statement which includes printing out J. However, it requires a slightly different FORMAT to refer to, because the machine must be told how and where to write the value of J.

ISN32. CONTINUE sends the machine back to the beginning of the DO loop nested closest to the end of the nest structure. If, of course, the five possible values in that loop have all been used up, it moves on to the next outer loop and starts the inside loop again; if both all the N4 and N3 values have been used up, it changes N2 and starts both N4 and N3 loops again, etc.

ISN37 (statement 100). We have already discussed this FORMAT statement in our discussion of ISN3. But I would like to point out the way the line carries over. The "1" listed causes the information to continue past the 72-character maximum which a single IBM card can accommodate.

ISN40. The 10X of the format causes the machine to print ten spaces, so as to allow the word to align in columns. The 4A5 means that there will be four units of five characters each (see ISN2).

ISN41 (statement 103). Here we have told the machine to print J, which will always be an integer, with up to six numbers (more than needed, but this is how it is customarily done), followed by 4X (four blanks) totalling the 10 blanks we called for in ISN40.

ISN42 and 43. All FORTRAN IV programs end in this way, by convention, to indicate to the machine that there are no more instructions in this program.

some other program on its schedule.

Now that we have mentioned some of the mechanical aspects of programs and have seen two examples of unconventional uses of computers, I want to make some general observations on the logic of the less mechanical portions of programming. For example, the meaning of the equation in FORTRAN. Consider the following:

N=N + 1

In Algebra this would be a contradiction. But in FORTRAN, in the context of a program in which "N" has already been defined, it simply redefines N as being the old N plus one. In the Emmett Williams piece, we have a situation where a program includes a calculation that is always given in terms of real numbers. A variable must therefore be used whose initial letter is reserved for real numbers. But we want a truncated form of it, an integer, thus:

A=(a real number)
I=A

Both of these uses of equations are unnecessary or "illegal" in some other systems. Here, then, what we have isn't really an equation at all but a definition in the linguistic sense, very much as if I were to say to someone "I mean the same by 'a big meal' as you do, but your 'a big meal' would have to have several more courses." The second example is analogous to saying, "Choose a man from this photograph of several men. I will now tell you a story about the man you have chosen." Linguistically this stage cannot be bypassed. The meanings expressed in programming languages are often easier to visualize, if we consider their verbal analogues.

The use of the second example will be illustrated later. But the use of the "N=N + 1" statement is presentable here. Suppose we calculated a random sampling of notes which were each to be played on a piano, but were gradually to be transposed upwards, disappearing at the top (a natural enough ending for a piece of music perhaps). We might well want to use a computer to calculate all the transpositions as we moved upwards. We could do it either by going up one note each time from the bottom of the keyboard. We might even want the degree of increment to increase. This could be done by choosing one variable, an integer, "K". We would double "K" at some point in the program, which the computer would pass through a number of times in the course of executing the program. The statement would look like this:

K = K * 2 (the asterisk is a multiplication sign in Fortran)

Later in the program would appear the following

N=N + K

This would result in each value of N becoming higher and higher at an increasingly rapid rate.

Let us now look at a common problem for which I would like to return to the first illustration we discussed, the five-unit-per-line poem, with thirty lines to a stanza. There is an important statement of instruction called a DO loop, which follows this kind of pattern:

```
10  DO 50 N=1, 30
        (the various stages of setting
        up N as an index to however
        many lines have printed would
        come here, followed, perhaps, by
        a nested DO loop which would
        say:)
20  DO 50 J=1, 5
        (now a parallel structure would
        be given, to tell us how many of
        the five values of IWORD had
        been figured, and placing these
        in one-to-one correspondence
        with words fed in the data stage
        of reading cards, tape or what-
        ever)
30  WRITE (6,102) (LINE(J),
    J=1,5)
50  CONTINUE
```

In most cases, the DO loop is an efficient method of repeating similar operations. It causes the machine to fill in five (and

no more) values to go in the line, then thirty lines, and no more. But it would also be possible to nest these two loops inside another loop if, for example, one wished to write any given number of thirty-line stanzas, each one to a page (which would be calculated by making a slight modification in the FORMAT statement referred to). The thing to worry about here is that the closest-changing calculation must be the "inside" loop (the closest to the end, the CONTINUE statement), and the slowest changing loop must be the "outside" one, with the others nested similarly according to their rates of changes.

Another useful procedure is the IF statement. Let us say that I want to see if such-and-such a number is greater than zero. If the number I want to test is M, my statement will look like this:

```
IF  (M) 40,40,60
```

By convention, this means that if M is negative or equal to zero, the machine will go to statement number 40 (where perhaps something will happen to it, followed by another IF statement conceivably which might even route it back to statement 40 if one treatment wasn't enough. If M is positive, it will skip statement 40 and proceed directly to statement 60, ignoring all intervening operations. Similarly, we see whether one number is greater than another by using a structure like the following one:

```
IF  (M-100)  40,40,60
```

This one tells us that if M is less than or equal to 100, the machine will go to statement 40, but if it is greater than 100 it will proceed directly to statement 60.

In the *Hank and Mary* program there is a slightly different kind of IF statement, in ISN16, a use of a MOD function, which tests if the line being calculated is one greater than a number evenly divisible by ten. If it is, the format in which the line is written includes the number of the line. In this way lines 1, 11, 21 etc. come to be labelled, while lines 6, 27 and 228 do not.

Finally I want to discuss shuffling or randomizing processes. Let us suppose that a theatre director has five characters, each one is given 15 seconds to make a move, there are six points to which they can go, and three rates of speed. A computer printout using a random generator number, might read like this:

1. CHARACTER 1 GOES QUICKLY TO POINT 2
   CHARACTER 2 GOES SLOWLY TO POINT 2
   CHARACTER 3 GOES QUICKLY TO POINT 6
   CHARACTER 4 GOES MODERATELY TO POINT 3
   CHARACTER 5 GOES SLOWLY TO POINT 3
2. CHARACTER 1 GOES SLOWLY TO POINT 2 (i.e., he stays there)
   CHARACTER 2 GOES MODERATELY TO POINT 4
   CHARACTER 3 GOES SLOWLY TO POINT 6 (i.e., he stays there)
   CHARACTER 4 GOES MODERATELY TO POINT 1
   CHARACTER 5 GOES QUICKLY TO POINT 6

   (etc.)

A six-word unit "To everything there is a season," can be fed into a computer, and from one to six, as follows:

to (1)
everything (2)
there (3)
is (4)
a (5)
season (6)

A computer can calculate, independently, a random sequence of numbers such as the following: 1, 6, 5, 2, 4, 5, 4, 5, 6, 4, 5, 6, 5, 6, 6, 5, 2, 2, (I obtained this list not with a computer, but with dice). By setting these in one to one correspondence with the indices of the word, one would obtain (five words to a line) the following:

to season a everything is
a season is a season
is season season a everything
everything

(continued on page 17)

Program for Proposition No. 2 for Emmett Williams, **by Alison Knowles, realized by James Tenney.**

```
EE317M      TENNEY                          FORTRAN SOURCE LIST
        ISN         SOURCE STATEMENT
          0 $IBFTC ARCH    LIST,REF
          1        DIMENSION MAT(3,17),SIT(8,25),LIT(4,4),INH(11,22)
          2        INTEGER SIT
          3        CALL RAND1
          4        READ(5,101)((MAT(I1,J1),I1=1,3),J1=1,17)
         15  101   FORMAT(6X,3A6)
         16        READ(5,201)((SIT(I2,J2),I2=1,8),J2=1,25)
         27  201   FORMAT(6X,8A6)
         30        READ(5,301)((LIT(I3,J3),I3=1,4),J3=1,4)
         41  301   FORMAT(6X,4A6)
         42        READ(5,401)((INH(I4,J4),I4=1,11),J4=1,22)
         53  401   FORMAT(6X,11A6)
         54        DO 200 N=1,50
         55        WRITE(6,102)
         56  102   FORMAT(1H1)
         57        DO 200 M=1,12
         60        JM=1.+RAND(17.)
         61        JS=1.+RAND(25.)
         62        JL=1.+RAND(4.)
         63        JI=1.+RAND(22.)
         64        WRITE(6,202)(MAT(I1,JM),I1=1,3),(SIT(I2,JS),I2=1,8),(LIT(I3,JL),I3
                  1=1,4),(INH(I4,JI),I4=1,11)
        105  202   FORMAT(1HO,5X,11HA HOUSE OF ,3A6/12X,8A6/18X,6HUSING ,4A6/24X,13HI
                  1NHABITED BY ,11A6)
        106  200   CONTINUE
        111        CALL RAND3
        112        STOP
        113        END
```

The work uses a list of materials, a second list of situations (places where the houses described might be situated), a list of how each might be lit, and a list of who might inhabit them, four lists in all, each punched on cards and each to be kept track of separately throughout the program.

ISN1. These lists are represented by MAT, SIT, LIT and INH respectively, each of which is planned as a two-dimensional array.

ISN2. SIT begins with an "S," a letter which is reserved for variables representing real numbers. It is necessary to use it as an integer variable. This could be done by renaming it ISIT throughout (or KSIT, MSIT, JSIT, etc.) But to keep the whole concept easy to visualize, the statement was given defining SIT as an integer.

ISN3. See also ISN60, where RAND is discussed. But this statement belongs properly here as a means of causing the computer to read a card which has had a number punched on it which is then taken as the first number of a random-number generating program (see ISN111). The CALL RAND activates the subprogram known as RAND which computes the successive random numbers.

ISN4. The machine now reads and stores in the order of listing the seventeen possible words called MAT. Some consist of more than the six characters which the computer conventionally allots to a word (in fact they vary up to three-times-six, or eighteen characters; the machine is directed to attach its indexing numbers of the seventeen materials to every three words, the other unused characters — unneeded by "words" which, of course, may be phrases, total less than eighteen characters — to print as blanks at the end of "words") according to FORMAT statement 101 of the program.

ISN15. This format statement could have been listed later in the program, but that might be harder to keep track of what format was being referred to because there are so many characters, variables and sets of words.

ISN16. Next the machine reads into its computer "memory" all the possible values of SIT as it did with MAT in ISN4. The difference here is that the longest member of SIT requires 8 units of 6 characters each to write, given in the format statement ISN27. It therefore numbers every eight units, until it has, with appropriate blanks of course, run through the cycle for each of the 25 possible values of SIT.

ISN27. This is the FORMAT statement for SIT. Compare it with those for MAT (given in INS15), for LIT (given in ISN41), and for INH (given in ISN53).

ISN30. We now read into the machine the four types of lighting to be used, represented by LIT, but requiring a maximum of only four units of six characters each, to which ISN41, the corresponding format, is adapted.

ISN42. Finally here we read the last and longest of the four lists into the machine, which requires 22 units of 6-times-11 or 66 characters total, — up to 66 characters being needed for the longest member of the set, — and which, in its turn requires the corresponding format statement, ISN53. All the cards have now been read and each variable has been listed in order in its appropriate set.

ISN54. We now enter the DO loop. The computer conventions of using DO loops require us to specify how many times the loop will be run through, so the number 50 has been selected which, as will be seen, results in a total of fifty pages to the printout, of which, necessarily, only a portion have been printed in illustration 5.

ISN55 and ISN56. The machine is told to write according to format statement 102, which simply gives the convention (1H1) that tells the machine to start a new page. But the machine "remembers" how many times it has been through the loop, and when it has gone through the 50 requisite number of pages (see ISN54), it will have completed its job and leave the DO Loop. It can also be seen that, if it had been desired, it would have been a very simple matter to write out the numbers of the pages.

ISN55 might then have said WRITE (6,102)N, and ISN56 would say, perhaps, "102 FORMAT (1H1,36X,T6)" with the 36 blanks added to make a strong visual indentation for the page numbers.

ISN57. Each line of the printout is to appear separately, four lines per stanza. There is room for only 50 lines per page of computer printout, so there is room for only 12 sets of four lines each, 48 lines in all. Therefore this DO loop varies from 1 to 12, represented by M, which is, an integer variable.

ISN60. Now we need to calculate our random integers which we are going to put into one to-one correspondence of our four lists (sets). There are seventeen members of our first set, so we need a random number from one to seventeen. If we said "JM=RAND(17.)" our first number might be zero, so we add one to it throughout. Setting an integer variable equal to an expression like "Rand (17.)" causes the number calculated to have its post-decimal portion to be truncated.

ISN61, 62 and 63. The same process is followed as in ISN60, according to how many possible values there are for SIT, LIT and INH.

ISN64. The machine now writes out, according to ISN105 (format statement 202), the materials which correspond to the values it has calculated. Note how here it is again necessary to repeat the maximum number of six-character "words" which were needed to express each maximum within the four sets of variables. The blanks printed out, of course, do not appear visually.

ISN105. Going through this format statement, "1HO" means, by convention, that there will be a new line following a double space. The writing starts after five blanks, indicated by "5X." Different instructions are separated by commas. 11H means that the following eleven characters will be printed verbatim, as they are given in the format statement, with each blank (as described before) counting as a character. "3A6" refers, as described in the text, to "A-Conversion" and to the maximum of three six-character words in the MAT set. The slash gives us a new lines, which will now begin with 12 blanks (shown by the "12X" indication, in order to offer greater legibility. Next we write out the value we determined for SIT, with its maximum of 8 six-character units, and, again, come to a new line. This new line is intended to be indented farther for legibility (18X), and starts with the 6-character (five letters plus a blank) word "using," and we print the 4A6 value we have calculated for LIT. Finally on a new Line, we print, after 24 blanks, and the 13-character phrase "inhabited by," the value computed for INH in the form corresponding to it in the A-conversion list. Note how the run-on Line of the format statement has, again, necessitated the carry-on indicator-signal of 1 (which falls into a space reserved for it on IBM cards) and which is printed out in a special column specifically reserved for it, indicating that the 72 characters of an IBM card have been used up. Since the machine has now printed out a full set of values it has calculated, it continues by returnnig to the beginning of this nested DO loop, unless it has printed out the twelve sets of values of M, referred to in ISN57, and which we can now see for the first time actually represents the twelve stanzas that can be printed on an IBM page. If these have all been printed out, control is transferred to the outer DO loop, which results in a new page being made available for printing out. And when the 50 pages allotted by the outer DO loop have been used up, the job is now printed out.

ISN106. This simply indicates the end of the DO loop and keeps the machine running along until it has used up all its instructions.

ISN111. Here we are asking the computer to print out the last value used of its RAND series for possible use as the first or initial number in a future run, so that all its runs won't start in the same way, which might be aesthetically undesirable. This is not a convention of all FORTRAN but of the specific RAND subprogram of computing random numbers.

ISN112 and 113. These are the required endings for programs.

At the end of printouts a time-used indicator is usually given, for accounting purposes. It may be interesting to note that the entire process of computing and printing out the full fifty pages of the text required only 1.93 minutes.

Proposition No. 2 for Emmett Williams, **by Alison Knowles, realized by James Tenney (sample section, four out of fifty pages).**

A HOUSE OF LEAVES
ON AN ISLAND
USING NATURAL LIGHT
INHABITED BY ALL RACES OF MEN REPRESENTED WEARING PREDOMINANTLY RED CLOTHING

A HOUSE OF PLASTIC
IN A DESERTED CHURCH
USING ALL AVAILABLE LIGHTING
INHABITED BY PEOPLE WHO SLEEP ALMOST ALL THE TIME

A HOUSE OF STEEL
AMONG OTHER HOUSES
USING ALL AVAILABLE LIGHTING
INHABITED BY FISHERMEN AND FAMILIES

A HOUSE OF BRICK
BY A RIVER
USING ELECTRICITY
INHABITED BY VARIOUS BIRDS AND FISH

A HOUSE OF BRICK
IN HEAVY JUNGLE UNDERGROWTH
USING CANDLES
INHABITED BY FRENCH AND GERMAN SPEAKING PEOPLE

A HOUSE OF STEEL
UNDERWATER
USING ALL AVAILABLE LIGHTING
INHABITED BY LOVERS

A HOUSE OF SAND
IN A DESERTED CHURCH
USING ELECTRICITY
INHABITED BY PEOPLE WHO SLEEP VERY LITTLE

A HOUSE OF DISCARDED CLOTHING
UNDERWATER
USING ALL AVAILABLE LIGHTING
INHABITED BY COLLECTORS OF ALL TYPES

A HOUSE OF WEEDS
AMONG SMALL HILLS
USING ELECTRICITY
INHABITED BY COLLECTORS OF ALL TYPES

A HOUSE OF WOOD
IN A PLACE WITH BOTH HEAVY RAIN AND BRIGHT SUN
USING NATURAL LIGHT
INHABITED BY ALL RACES OF MEN REPRESENTED WEARING PREDOMINANTLY RED CLOTHING

A HOUSE OF ROOTS
ON THE SEA
USING ALL AVAILABLE LIGHTING
INHABITED BY PEOPLE WHO ENJOY EATING TOGETHER

A HOUSE OF PAPER
IN A DESERT
USING CANDLES
INHABITED BY VARIOUS BIRDS AND FISH

A HOUSE OF PLASTIC
ON THE SEA
USING NATURAL LIGHT
INHABITED BY FRIENDS

A HOUSE OF LEAVES
IN A COLD, WINDY CLIMATE
USING ELECTRICITY
INHABITED BY PEOPLE WHO LOVE TO READ

A HOUSE OF DISCARDED CLOTHING
ON THE SEA
USING CANDLES
INHABITED BY LITTLE BOYS

A HOUSE OF WOOD
IN A DESERTED CHURCH
USING CANDLES
INHABITED BY FRENCH AND GERMAN SPEAKING PEOPLE

A HOUSE OF ROOTS
IN A METROPOLIS
USING ALL AVAILABLE LIGHTING
INHABITED BY COLLECTORS OF ALL TYPES

A HOUSE OF GLASS
ON THE SEA
USING ELECTRICITY
INHABITED BY PEOPLE WHO EAT A GREAT DEAL

A HOUSE OF STRAW
IN DENSE WOODS
USING ELECTRICITY
INHABITED BY PEOPLE FROM MANY WALKS OF LIFE

A HOUSE OF WOOD
IN A HOT CLIMATE
USING ALL AVAILABLE LIGHTING
INHABITED BY AMERICAN INDIANS

A HOUSE OF BROKEN DISHES
BY A RIVER
USING ELECTRICITY
INHABITED BY PEOPLE WHO SLEEP ALMOST ALL THE TIME

A HOUSE OF STEEL
IN AN OVERPOPULATED AREA
USING CANDLES
INHABITED BY PEOPLE WHO EAT A GREAT DEAL

A HOUSE OF GLASS
IN JAPAN
USING ELECTRICITY
INHABITED BY FRIENDS

A HOUSE OF STEEL
IN A METROPOLIS
USING ELECTRICITY
INHABITED BY VERY TALL PEOPLE

A HOUSE OF STEEL
IN A COLD, WINDY CLIMATE
USING ELECTRICITY
INHABITED BY NEGROS WEARING ALL COLORS

A HOUSE OF SAND
IN SOUTHERN FRANCE
USING ELECTRICITY
INHABITED BY VEGETARIANS

A HOUSE OF PLASTIC
IN A PLACE WITH BOTH HEAVY RAIN AND BRIGHT SUN
USING CANDLES
INHABITED BY COLLECTORS OF ALL TYPES

A HOUSE OF PLASTIC
UNDERWATER
USING NATURAL LIGHT
INHABITED BY FRIENDS

A HOUSE OF BROKEN DISHES
AMONG SMALL HILLS
USING NATURAL LIGHT
INHABITED BY LITTLE BOYS

A HOUSE OF MUD
IN A HOT CLIMATE
USING ALL AVAILABLE LIGHTING
INHABITED BY FRENCH AND GERMAN SPEAKING PEOPLE

A HOUSE OF MUD
IN A HOT CLIMATE
USING NATURAL LIGHT
INHABITED BY COLLECTORS OF ALL TYPES

A HOUSE OF GLASS
IN MICHIGAN
USING NATURAL LIGHT
INHABITED BY FRIENDS

A HOUSE OF SAND
IN A HOT CLIMATE
USING NATURAL LIGHT
INHABITED BY LITTLE BOYS

A HOUSE OF BRICK
IN HEAVY JUNGLE UNDERGROWTH
USING CANDLES
INHABITED BY AMERICAN INDIANS

A HOUSE OF DISCARDED CLOTHING
IN DENSE WOODS
USING NATURAL LIGHT
INHABITED BY LOVERS

A HOUSE OF DUST
ON AN ISLAND
USING CANDLES
INHABITED BY NEGROS WEARING ALL COLORS

The fact that this is not particularly meaningful need not concern us here. Other instances of the same structure might well have greater aesthetic value. The point is that this is the basic process of randomizing a list – a process which has an increasing significance in the arts.

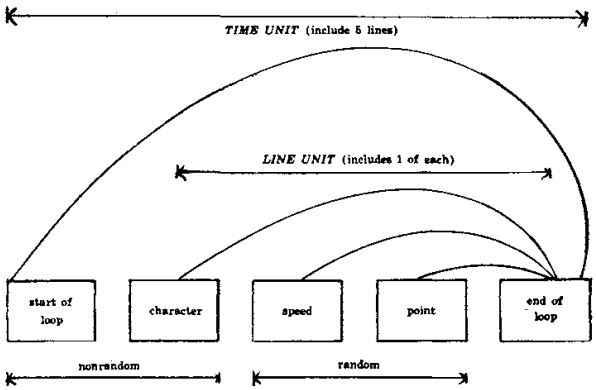
To return to our theatrical example once more, I would like to point out that both the time unit and the list of characters would be represented by variables which simply increased by one, each time we went through the cycle of the loop. By the method of randomizing the speeds of movement and of points gone to would have to be done by means of a random number-generating subprogram (as in *Proposition No. 2*). The statement might look like this:

```
A=1. + RAND(6.)
K=A
```
(later on, the K would turn out to be the index of the point)

```
B=1. + RAND(3.)
L=B
```
(similarly L would be the index of rate of movement)

The decimal point is needed because the RAND form of subprogram (used here) computes real numbers. These "K=A" or "L=B" statements tell the machine to truncate the number, that is, to ignore what comes after the decimal point. But not to round it off: therefore 0.999 would become simply 0., and since that would not make a usable index, it is necessary to increase each number by 1, in order to come up with only 1., 2., or 3. (Note that it was necessary to use one of the letters which we reserved for real numbers as the variable being calculated.) We then pick an integer variable and make it equal to the real number variable, which makes it no longer necessary to include decimal points after the numbers.

A sketch of the logic of the project would, therefore, look like this:



The ease of mixing random and nonrandom elements in a structure like this suggests that the opposition assumed to exist between nonaleotoric and aleotoric devices is rather trivial. The two represent rational poles, to which their opposites would be the irrational, the inconsistent, the taste-oriented. Computers perform only rational operations; they can never be made to do more than imitate the irrational.

When the artist is able to eliminate his irrational attitudes (if any) about the mythology of computers, and becomes willing not simply to dump his fantasies in the lap of some startled engineer, but to supply the engineer with:

1) the rudiments of his program in such a language as FORTRAN or one of the other very common ones;

2) a diagram of the logic of his program, such as I just used to illustrate the theatre example.

3) a page or so of how he would like the printout to look (perhaps alternates could be supplied).

then he will be in a position to use the speed and accuracy of computers. There will be few of the present disappointments, which are due usually more often to the artist's naivete than to the engineer's lack of information or good will. The onus is on the artist, not his tools, to do good work.