

10
I29A

#425 CIVIL ENGINEERING STUDIES

C.1 STRUCTURAL RESEARCH SERIES NO. 425

UIIU-ENG-75-2031



TECHNOLOGY FOR THE FORMULATION AND EXPRESSION OF SPECIFICATIONS

VOLUME III: TECHNICAL REFERENCE MANUAL

Metz Reference Room
Civil Engineering Department
1106 C. E. Building
University of Illinois
Urbana, Illinois 61801

By
R. N. WRIGHT
J. R. HARRIS
J. W. MELIN
C. ALBARRAN

A Report on a Research Project
Sponsored by
THE NATIONAL BUREAU OF STANDARDS
Contract No. 5-35844

UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN
URBANA, ILLINOIS
DECEMBER 1975

CIVIL ENGINEERING STUDIES
STRUCTURAL RESEARCH SERIES NO. 412

UIIU-ENG-74-2027

THE SUBSTITUTE STRUCTURE METHOD
FOR EARTHQUAKE-RESISTANT DESIGN OF REINFORCED CONCRETE FRAMES

By

Akenori Shibata
and
Meté A. Sozen

A Report on a Research
Project Sponsored by
RESEARCH APPLIED TO NATIONAL NEEDS PROGRAM
THE NATIONAL SCIENCE FOUNDATION
Research Grant GI 29934

UNIVERSITY OF ILLINOIS
at URBANA-CHAMPAIGN
URBANA, ILLINOIS
October 1974

CONTENTS

INTRODUCTION.....	1
SMOOTHED RESPONSE SPECTRA.....	2
DESCRIPTION OF THE METHOD.....	3
NUMERICAL EXAMPLE.....	8
TESTS OF FRAMES WITH RIGID BEAMS.....	11
TESTS OF FRAMES WITH FLEXIBLE BEAMS.....	16
SUMMARY.....	18
ACKNOWLEDGMENTS.....	19
APPENDIX I. - REFERENCES.....	19
APPENDIX II. - NOTATION.....	21
TABLES.....	22
FIGURES.....	25

INTRODUCTION

The substitute-structure method is a procedure for determining the design forces, corresponding to a given type and intensity of earthquake motion represented by the design spectrum, for a reinforced concrete structure. The method is explicitly a design (and not an analysis) procedure: its objective is to establish the minimum strengths the components of the structure must have so that a tolerable response displacement is not likely to be exceeded.

The central and significant feature of the substitute-structure method is that it provides a simple vehicle for taking account of inelastic response of reinforced concrete in the design of multi-degree-of-freedom structures. The specific advantages are: (1) use of linear-response models for dynamic analysis, (2) choice in setting limits of tolerable response in different elements of the structure, and (3) deliberate consideration of displacements in the design process.

This paper demonstrates the application of the method to structures satisfying the following:

1. The system can be analyzed in one vertical plane.
2. No abrupt changes in geometry or mass along the height of the system.
3. Columns, beams, and walls (represented as columns) may be designed with different limits of inelastic response, but the limits should be the same for all beams in a given bay and all columns on a given axis.
4. All structural elements and joints are reinforced to avoid significant strength decay as a result of repeated reversals of the anticipated inelastic displacements.
5. Nonstructural components do not interfere with structural response.

In addition to a detailed description of the method, the paper includes a series of "tests." Frames, ranging in height from 2 to 10 stories, are designed for a particular response spectrum using the substitute structure method. These frames are then "subjected" to various earthquake motions: their responses are calculated using inelastic dynamic analysis based on a realistic hysteresis for reinforced concrete.

SMOOTHED RESPONSE SPECTRA

Figures 1 and 2 contain acceleration response spectra for eight recorded ground-motion components listed in Figure 3. Response data are shown for two damping factors, $\beta = 0.02$ and 0.10 , with each record normalized to an acceleration of $0.5g$.

The first six motions were grouped together because their linear-response spectra have similar shapes. As indicated by the plot in Fig. 3,

there is a comparable proportionality, for these six records, between the maximum acceleration and spectrum intensity (4) which provides a rationale for normalizing them with respect to maximum acceleration. Linear-response spectra for motions 7 and 8 are distinctly different from those for the first six. They have been treated individually.

It is not the object of this work to propose a generalized response spectrum. However, to test the proposed procedure it is necessary to work with a smoothed set of spectra which describe the calculated response for the individual strong motion records. For that purpose, the curves shown by heavy solid lines in Fig. 1 and 2 were selected.

It was assumed that the design response acceleration for any damping factor, β , could be related to the response for $\beta = 0.02$ using Eq. 1.

$$\frac{\text{Response acceleration for } \beta}{\text{Response acceleration for } \beta=0.02} = \frac{8}{6 + 100\beta} \quad (1)$$

In choosing the design spectra, more weight was placed in making them comparable to the calculated values at $\beta = 0.10$ than at $\beta = 0.02$, because values of the damping factor on the order of 0.1 rather than 0.01 are typical in applications of the substitute-structure method. An effort was also made to select curves, especially those in Figure 1, described by simple expressions.

DESCRIPTION OF THE METHOD

Main characteristics of the substitute-structure method are (a) definition of a substitute frame, with its stiffness properties related to but differing from the actual frame, and (b) calculation of design forces from a modal spectral analysis of the substitute frame using a linear-response spectrum (or from a linear-response-history analysis for a given

ground motion.) The operations may be divided into three steps:

(1) Based on tolerable limits of inelastic response, determine the stiffnesses of the substitute-frame members.

(2) Calculate modal frequencies and damping factors for the substitute structure.

(3) Determine design forces.

Details of the procedure for each step are described below, followed by a numerical example.

It is assumed that preliminary member sizes of the actual structure are known from gravity-load and functional requirements, precedent, or a previous trial.

The Substitute Structure. The flexural stiffnesses of substitute-frame elements are related to those of actual-frame elements in accordance with Eq. 2.

$$(EI)_{si} = (EI)_{ai}/\mu_i \quad (2)$$

where $(EI)_{si}$ and $(EI)_{ai}$ are cross-sectional flexural stiffnesses of the element i in the substitute and actual frame, respectively, and μ_i is the selected tolerable "damage ratio" for element i .

Physical interpretation of the damage ratio for a particular condition, a moderately reinforced slender beam subjected to antisymmetrical end moments, is illustrated in Fig. 4. The solid curve in Fig. 4c represents the relationship between the applied moment, M , and the end rotation, θ , caused by flexural deformation within the span.

The term $(EI)_a$ is calculated using the fully cracked section (linear stress-strain curves and no tensile strength for concrete). The $M-\theta$ curve, based on $(EI)_a$, corresponds approximately to a line drawn from the

origin to the "yield point" of a section with compactly placed tensile reinforcement having a definite yield stress. The damage ratio, μ , sets a lower slope and implies that a rotation, approximately $\mu\theta_y$, will be attained if the effective or average stiffness of the member is changed as indicated in Eq. 2. In that respect, the damage ratio, μ , is comparable to but not exactly the same as "ductility" based on the ratio of maximum to yield rotation. Quantitatively, damage and ductility ratios are identical only for elasto-plastic response. It must be emphasized that a damage ratio of, say, six requires a larger ratio of "ductility" based on curvature or strain in members with moment gradients.

Choice of tolerable damage ratios for structural elements is governed by the nature, cost, and function of the entire building as well as on the type and detailing of the elements. Recommendation of specific values is beyond the scope of this paper. To permit quantitative demonstrations, it will be assumed that tolerable damage ratios are unity for columns and six for the beams, in keeping with the approach that energy should be dissipated primarily in the beams which are often more convenient to detail for sustained resistance through many cycles of response into the inelastic range.

Modal Frequencies and Damping Factors. Periods or frequencies and mode shapes and modal forces for the undamped substitute structure are obtained from a linear response analysis.

The modal damping factors for the substitute structure are calculated as described below.

It was observed (3) that the maximum inelastic earthquake response of single-degree-of-freedom reinforced concrete systems could be estimated

by analyzing a linear model with reduced stiffness and a substitute damping factor related to the damage ratio approximately as follows.

$$\beta_s = 0.2 (1 - (1/\mu)^{1/2}) + 0.02 \quad (3)$$

where β_s = substitute (equivalent viscous) damping factor and μ = damage ratio.

Equation 3 is based on dynamic tests of reinforced concrete elements (16) and one-story frames (3). The form of the expression was derived (3) from a model by Jacobsen (6). It provides a quantitative estimate of the amount of equivalent viscous damping required to simulate the observed effect of hysteretic damping on the response of a reinforced concrete element to earthquake excitation. Various approaches to the linear representation of nonlinear response are discussed in ref. 1, 2, 5, 6, 8, 11, 15, and 17.

If the individual elements of a frame are designed for different values of μ , individual values of β_s have to be combined to obtain a single "smeared" value for use in modal analysis. In the substitute-structure method this is done by assuming that each element contributes to the modal damping in proportion to its relative flexural strain energy associated with the modal shape:

$$\beta_m = \sum_i \frac{P_i}{\sum_i P_i} * \beta_{si} \quad (4)$$

$$P_i = \frac{L}{6(EI)_{si}} (M_{ai}^2 + M_{bi}^2 - M_{ai}M_{bi}) \quad (5)$$

where β_m = smeared damping factor for mode m , L = length of frame element, $(EI)_{si}$ = assumed stiffness of substitute-frame element i , M_{ai} and M_{bi} = moments at ends of substitute-frame element i for mode m .

An alternate method of obtaining modal damping factors for the substitute structure is provided by elements with complex stiffness (10, 14).

$$k_{si} = \frac{k_{ai}}{\mu_i} [1 + 2\beta_{si} * (-1)^{1/2}] \quad (6)$$

where k_{si} = stiffness of substitute-frame member i , k_{ai} = stiffness of actual-frame member i , μ_i = tolerable damage ratio for member i , and β_{si} = substitute damping for member i from Eq. 3.

Dynamic equilibrium of the entire substitute structure can then be expressed by Eq. 7.

$$[M] \{\ddot{x}\} + ([K_1] + [K_2]) \{x\} = 0 \quad (7)$$

where $[M]$ represents the mass matrix, $[K_1]$ and $[K_2]$ represent the real and imaginary parts of the stiffness matrix, and x refers to the displacements. Modal frequencies and damping factors are determined by solving for eigenvalues of the complex matrix.

Both methods give closely comparable answers. The method based on strain energy was used in this paper because of its simplicity and because of its direct relationship to the physical interpretation of the substitute structure.

Design Forces. Design forces in individual elements are based on the root-sum-square combination amplified by a factor given in terms of the base shear.

$$F_i = F_{irss} * \frac{V_{rss} + V_{abs}}{2 V_{rss}} \quad (8)$$

where F_i = design force in element i , $F_{i\text{rss}}$ = square root of the sum of the squares (RSS) of the modal forces for member i , V_{rss} = base shear based on RSS of modal base shears, V_{abs} = maximum value for absolute sum of any two of the modal base shears.

To reduce risk of excessive inelastic action in the columns, the design moment from Eq. 8 should be amplified for columns by a factor of 1.2.

NUMERICAL EXAMPLE

Consider the three-story planar frame described in Fig. 5b. Design forces are to be determined for response spectrum A shown in Fig. 1 (Characteristic ground acceleration = 0.5g). It is assumed that $\mu = 1.0$ for columns and $\mu = 6.0$ for beams. Let $E = 3.6 \times 10^6$ psi = 2.5×10^4 MPa for concrete.

Moments of inertia indicated in Fig. 5a refer to gross plain cross section. Because the amount of reinforcement in the frame members is not known at this stage of design, it is assumed that the ratio of cracked-to gross-section moment-of-inertia is 1/2 for columns and 1/3 for beams.

Moments of inertia of the substitute frame are obtained from Eq. 2, noting that I_{ai} refers to cracked section.

$$\text{For the columns, } I_c = 1.33/2 = 0.67 \text{ ft}^4 = 5.8 \times 10^{-3} \text{ m}^4$$

$$\text{For the beams, } I_b = 1.95/(3 \times 6) = 0.11 \text{ ft}^4 = 9.5 \times 10^{-4} \text{ m}^4$$

Modal periods, shapes, and forces (for a nominal response acceleration of 1.0g) are calculated for the substitute structure using a linear

dynamic response analysis.*

For the three-story frame (Fig. 5), the calculated periods were 0.85, 0.19, and 0.078 sec. The moments calculated for an arbitrary nominal response acceleration of 1.0g are shown in Fig. 5c for each mode.

Substitute damping factors are obtained from Eq. 3.

For the columns, $\beta_c = 0.02$

For the beams, $\beta_b = 0.2 (1 - 1/(6)^{1/2}) + 0.02 = 0.14$

The smeared damping factor for each mode is determined using Eq. 4 and 5. Because strain energy is involved as a relative magnitude, quantities in Fig. 5c can be used. To demonstrate a step in the calculations, consider P_i for a first-story column for the first mode (Eq. 5),

$$P_i = \frac{11.0}{6.0 * 5.2 * 10^5 * 0.67} [(1170)^2 + (256)^2 + (1170 * 256)] = 9.1 \text{ k-ft} = 12.3 \text{ kN-m}$$

Performing the above operation for each member in each mode, the following relative proportions are obtained.

	<u>Mode 1</u>	<u>Mode 2</u>	<u>Mode 3</u>
$\Sigma P_{\text{girders}} / \Sigma P_{\text{girders} + \text{col.}}$	0.55	0.21	0.04
$\Sigma P_{\text{columns}} / \Sigma P_{\text{girders} + \text{col.}}$	0.45	0.79	0.96

Modal damping factors for the substitute structure are (Eq. 4)

$$\beta_1 = 0.14 * 0.55 + 0.02 * 0.45 = 0.086$$

$$\beta_2 = 0.14 * 0.21 + 0.02 * 0.79 = 0.045$$

$$\beta_3 = 0.14 * 0.04 + 0.02 * 0.96 = 0.025$$

*Use a standard computer program for linear dynamic analysis available at the accessible computer center. Examples are TABS (Univ. of Cal., Berkeley), APPLE PIE (MIT, Cambridge, Mass.) and SUSHI (Univ. of Ill., Urbana).

The spectral acceleration response for each mode is then calculated using Eq. 1 with the pertinent damping factor. Resulting curves are shown schematically in Fig. 5a.

Base shears are most conveniently handled in terms of the "base shear coefficient", ratio of response base shear to weight of building.

$$v_m = (v_m \text{ for } 1.0 \text{ g}) * (S_{Am}/g) \quad (9)$$

where v_m = base shear coefficient for mode m , S_{Am} = design response acceleration for mode m , g = acceleration due to gravity. Values for $(v_m \text{ for } 1.0g)$ are obtained directly from the dynamic analysis.

$$v_1 = 0.77 * 0.48 = 0.37$$

$$v_2 = 0.18 * 1.4 = 0.25$$

$$v_3 = 0.053 * 0.92 = 0.049$$

$$v_{abs} = 0.37 + 0.25 = 0.62$$

$$v_{rss} = \sqrt{(0.37)^2 + (0.25)^2 + (0.049)^2} = 0.45$$

Design moments are calculated (Eq. 8) using the values in Fig. 5c modified for the appropriate design response acceleration. For example, the moment at the base of the first-story column for the first mode becomes $1170 * (0.48) = 560 \text{ kip-ft} = 760 \text{ kN-m}$. Thus, the design moment at the same section is

$$F_1 = 1.2 * \sqrt{(560)^2 + (200)^2 + (30)^2} * \frac{0.45 + 0.62}{2 * 0.45}$$

$$= 850 \text{ kip-ft} = 1150 \text{ kN-m}$$

The factor 1.2 is used for columns only. The actual design moment (earthquake) depends also on load factors deemed necessary in relation to

likelihood of design motion, expected quality of construction, and level of design stresses.

Lateral displacements of the frame are obtained from dynamic analysis of the substitute structure, with the appropriate damping factors β_1 , β_2 , and β_3 and response accelerations. Calculated modal displacements are listed below in ft.

	<u>Mode 1</u>	<u>Mode 2</u>	<u>Mode 3</u>
Level 3	0.37	-0.015	0.0004
Level 2	0.22	0.019	0.0011
Level 1	0.07	0.018	0.0015

As would be anticipated, the first mode governs the response. In this case, calculation of RSS values is unnecessary. Thus, the maximum displacement at Level 3 is estimated to be approximately 4.5 in. (0.11m). On the same basis, maximum relative story displacement is expected to approach two in. (0.05m) in the event of the design earthquake.

The response of a three-story frame, designed to resist the forces obtained as described above at yield level, to various ground motions is evaluated later in this paper along with other frames designed similarly.

TESTS OF FRAMES WITH RIGID BEAMS

Even though the substitute-structure method constitutes only a part of the entire design process, its result, a particular set of design forces, represents a synthesis of various decisions and the method is therefore best judged by the end product: whether the resulting system fulfills the original intent. This and the following sections describe "tests" of frames "designed" using the method. The tests were analytical. Design

forces for a series of frames were determined. Then, inelastic responses of these frames, with members having flexural yield capacities determined by the design process, to various ground motions were calculated.

The first series of frames, described in this section, were limited to frames with rigid beams, in order to permit investigation of several variables within a reasonable computation budget. The frames ranged from two to ten stories of eleven ft (3.35 m) with a weight of 72 kips (320 kN) concentrated at each story. The initial story stiffness (corresponding to cracked-section properties of reinforced concrete columns) was determined by assuming that the natural period of the system to be $0.1N$, where N is the number of stories. Two different groups of frames were considered. For the first group, story stiffness was assumed to be the same at all stories (designated as "uniform"). For the second group, story stiffnesses were assumed to vary so as to produce a linear first-mode shape.

The frames were "designed" for a target damage ratio, μ , of six using spectrum A (Fig. 1). Base and top-story shear coefficients are listed in Table 1. The effective damping factor was the same for all modes (0.14) because energy is assumed to be dissipated only by one class of elements. Each frame was then "tested" using the first six ground motions, normalized to a characteristic acceleration of 0.5 g, indicated in Fig. 3.

The response history was calculated at each level using the hysteretic-response rules defined in reference 16. Tensile strength of the concrete was ignored. The yield moment was set at the design requirement, the initial stiffness being determined by the selected period. Stiffness beyond yield was taken as five percent of the initial stiffness. The

analysis was made with an equivalent viscous damping proportional to stiffness, amounting to a damping factor of 0.02 for the first mode.

Results Test. Results of inelastic-response calculations are summarized in Fig. 6 for both series of frames ("uniform stiffness" and "varying stiffness"). The calculated maximum damage ratio at each level is plotted using a different symbol for each ground motion. A bar at each level indicates, for all six ground motions, the mean damage ratio (left end of the bar) and the mean plus one standard deviation (right end).

The data in Fig. 6 show that the distribution of the characteristic damage ratio (mean plus one standard deviation) was reasonably uniform over the height of the structures, with the values at the top story typically low. It is also seen that given a certain structure, the damage ratio at different levels was quite different for different ground motions, even though the six ground motions selected had generally similar response spectra.

Histograms of all calculated damage ratios are shown in Fig. 7. The overall means were less than 7.5 for both series (6.9 for frames with uniform stiffness and 7.3 for frames with varying stiffness). Considering that these mean values can be controlled by the choice of the spectral-response curve, method of summing modal components, or by a load factor, the results are positive in that, on the whole, the distribution of the mean and characteristic values over the heights of the buildings (Fig. 6) are reasonably uniform.

Strength Distribution over Height of Structure. A study was made of the distribution of story strength over the height of the building. Two frames (one with ten and the other with six stories) of each series were

redesigned with the same base shear strength but with the column strength varying as the story shears calculated for the first mode, rather than according to the RSS distribution.

Inelastic response of these four frames were calculated using motion no. 1 (Fig. 3). The results are illustrated in Fig. 7. It appears that even if the RSS distribution is slightly more complicated to use and resulted usually in an overdesign (Fig. 6) of the top-story columns, it is preferable to the FM distribution which resulted in large damage ratios in the top stories (Fig. 7). Figure 7 also illustrates the sensitivity of the calculated inelastic response displacement to variations in strength. Top-story shear strength was reduced less than 40 percent in going from RSS to FM distribution, but the response displacements at this level increased by an order of magnitude.

Ground-Motion Characteristics. The plot in Fig. 3 comparing maximum acceleration with spectrum intensity for the eight ground motions indicates that motions 7 and 8 have special characteristics. Motion 7, which plots below the line representing spectrum A, is evidently a more severe ground motion than indicated by its characteristic maximum acceleration. Using spectrum A, with acceleration as the index value, would underrate its effect. On the other hand, the same approach would overrate motion 8.

Open circles in Fig. 8 indicate response damage ratios for frames designed using spectrum A and then "subjected" to motions 7 and 8 normalized to a maximum acceleration of 0.5g. Although results are within extreme values shown in Fig. 6, it is evident that the frames have been underdesigned for motion no 7 and overdesigned for motion no. 8, a result to be anticipated from Fig. 3. This represents a failure of the design

spectrum A rather than of the substitute-structure method.

Solid circles in Fig. 8 indicate response damage ratios for frames proportioned according to design spectra B or C fitted to calculated response spectra for motions 7 and 8 (Fig. 2). These results are satisfactory, despite the tendency to overdesign the upper stories.

In relation to the success of the final design, the shape of the design spectrum is more critical than its magnitude. Overall magnitude of the spectrum can be compensated plausibly at another level of the design process. But the only stage where the frequency content (but not sequence) of the ground motion can be intelligibly anticipated is in the spectrum shape.

In this context, a critical feature of the substitute-structure method should be discussed. The method becomes plausible only with the understanding that the force response decreases as the structure becomes more flexible. If the characteristics of the ground motion are such that, in the range of the lower modes of the structure, the spectral acceleration response increases with an increase in period, it becomes necessary to assume a constant acceleration response for design up to that period at which response starts decreasing, unless the method is used iteratively, an alternative which is usually not desirable. In design spectra used in this paper, the portion of the spectrum at frequencies higher than approximately six Hz was chosen to decrease with increase in frequency because contributions of modes in this range were small and because they were more than compensated for by the reductions in the contributions of the lower modes. However, if the concern had been with structures having their lowest modes in this range, it would have been necessary to assume a flat response acceleration at the maximum amplification for all

frequencies above six Hz. In effect, the response curve must be such that the total base shear reduces as the structure becomes more flexible.

TESTS OF FRAMES WITH FLEXIBLE BEAMS

Three frames (Fig. 9) were designed for spectrum A with target damage ratios of $\mu = 6.0$ in the beams and $\mu = 1.0$ in the columns. Calculated periods and modal damping factors are shown in Table 2. Design forces for each element were determined as discussed specifically for the three-story frame. Frame elements were assigned yield moment capacities indicated by the design procedure, columns being designed for the governing top or bottom design moment. The design base shear coefficients, v , were 0.54, 0.30, and 0.15 for the three-, five- and ten-story frames respectively.

Response histories of each frame to motions 1-6 were calculated by an inelastic dynamic analysis program for frames, SAKE (13). Results of such analyses are compared with dynamic test results in reference 12. The assumed hysteresis and viscous damping was the same as those for the frames with rigid beams except that the stiffness after yielding was three percent of the initial stiffness.

Test Results. Mean beam damage ratios shown in Fig. 10 (left edges of the rectangles) present a favorable picture. They were all less than the target value of six and their distribution over the height of the structure was reasonably uniform. The same was true of the characteristic damage ratios (mean plus one standard deviation) which did not exceed seven. However, there was one motion which resulted in relatively large damage ratios (motion 2). Tuning the design method to maintain damage ratios

below six for all motions considered is possible but uneconomical. Furthermore, unless the design is ridiculously conservative, there is the possibility of another ground motion, with the same characteristic maximum acceleration and response spectrum, which may result in larger damage ratios than motion 2.

Maximum damage ratios indicated for the columns may be evaluated from two different viewpoints. One viewpoint would be to tolerate the few locations where the damage ratio has exceeded unity because (a) mean values for the six motions were always less than unity and (b) individual maxima, again for motion 2, were barely over two. In this light, the test results are considered as positive.

Another viewpoint would be to consider any column rotation into the inelastic range as unacceptable, in view of the difficulties involved in developing "ductility" in axially loaded reinforced concrete elements.

As indicated earlier in the paper, to maintain the columns in the elastic range requires special precautions (9). Consider, for example, a one-bay one-story frame. Lateral-load analysis of any type would result in equal moment-capacity requirements in the beam and the columns. If the two types of members are proportioned to have the same flexural strength, either one may develop inelastic rotations. In the design procedure described, the column moments are determined by amplifying the results of Eq. 4 by a factor of 1.2. Table 3 shows calculated damage ratios for three five-story frames subjected to ground motion 2. Frame Y was based on the described procedure. Frames X and Z were based on column design moments obtained by multiplying the results of Eq. 4 by 1.1 and 1.4, respectively.

Column damage ratios for frame Y exceeded unity at three joints, reaching a maximum value of 3.7. As discussed earlier in reference to Fig. 10, frame Y damage ratios exceeded unity at two joints but were close to two. All columns of frame Z had damage ratios less than unity. It is evident that columns may be maintained in the linear range by increasing the design moments. However, considering that a column "overstrength" factor of 1.2 resulted in damage ratios exceeding unity for only one of the six ground motions (Fig. 10) and that these were not intolerably large, it is considered to be adequate. Gravity-load or other requirements may also change the relative strengths of beams and columns in favor of the latter. If the strength ratio goes in favor of the beam and if the designer desires $\mu = 1.0$ or less for the columns, he must "override" the final design proportions.

SUMMARY

From the observation that the inelastic response to earthquakes of reinforced concrete elements could be represented by a linear-response model, a procedure was developed in reference 3 which incorporated the effects of inelastic energy dissipation to determine the design force for a single-degree-of-freedom structure using the ordinary linear-response spectrum. The substitute-structure method extends this procedure to multi-degree-of-freedom structures.

The proposed method can be used to determine earthquake design-force requirements for individual elements of a R/C structure given a design linear-response spectrum and explicit decisions about tolerable inelastic response, with the option of different limits of inelastic

response in different structural elements.

The paper includes a numerical example demonstrating the determination of design forces in a three-story frame and a series of analytical tests of the method using two- to ten-story frames.

ACKNOWLEDGMENTS

The work leading to the substitute-structure method was sponsored by Research Applied to National Needs Program of the National Science Foundation through Grant G129934 at the Civil Engineering Department of the University of Illinois, Urbana. Dr. Shibata's participation was made possible by the US-Japan Cooperative Science Program under the auspices of Japan Society for Promotion of Science and U. S. National Science Foundation. The IBM 360/75 computer system of the Department of Computer Science, Univ. of Ill., was used for computations.

The writers are indebted to Professor S. Otani, Univ. of Ill., for his critical thoughts which have influenced the paper and for use of computer programs SAKE and SUSHI.

APPENDIX I. - REFERENCES

1. Caughey, T. K., "Random Excitation of a System with Bilinear Hysteresis," Journal of Applied Mechanics, ASME, Vol. 27, No. 4, Dec. 1960, pp. 649-652.
2. Caughey, T. K., "Sinusoidal Excitation of a System with Bilinear Hysteresis," Journal of Applied Mechanics, ASME, Vol. 27, No. 4, Dec. 1960, pp. 640-643.
3. Gulkan, P. and Sozen, M. A., "Inelastic Response of Reinforced Concrete Structures to Earthquake Motions," Journal of the American Concrete Institute, v. 71, No. 12, December 1974, pp. 601-609.

4. Housner, G. W., "Behavior of Structures during Earthquakes," Journal of the Engineering Mechanics Division, ASCE, Vol. 85, No. EM4, October 1959, pp. 108-129.
5. Hudson, D. E., "Equivalent Viscous Friction for Hysteretic Systems with Earthquake-Like Excitations," Proceedings, Third World Conference on Earthquake Engineering, New Zealand, 1965, pp. 11 185-202.
6. Idris, I. M., Seed, H. B., "Seismic Response of Horizontal Soil Layers," Journal of the Soil Mechanics and Foundation Division, ASCE, Vol. 94, No. SM4, July 1968, pp. 1003-1031.
7. Jacobsen, L. S., "Steady Forced Vibration as Influenced by Damping," Transactions, ASME, Vol. 52, Part 1, 1930, pp. APM 169-181.
8. Jennings, P. C., "Equivalent Viscous Damping for Yielding Structures," Journal of the Engineering Mechanics Division, ASCE, Vol. 94, No. EM1, Feb. 1968, pp. 103-116.
9. Kobori, T., Minai, R. and Fujiwara, T., "Earthquake Response of Frame Structures Composed of Inelastic Members," Proceedings, Fifth World Conference on Earthquake Engineering, Rome, June 1973, Session 5B, No. 221.
10. Meirovitch, L., "Analytical Methods in Vibrations," The MacMillan Company, 1967.
11. Newmark, N. M., and Rosenblueth, E., "Fundamentals of Earthquake Engineering," Prentice-Hall, 1971.
12. Otani, S., Sozen, M. A., "Behavior of Multistory Reinforced Concrete Frames During Earthquakes," Structural Research Series No. 392, Dept. of Civil Engineering, University of Illinois, Urbana, November 1972.
13. Otani, Shunsuke, "SAKE. A Computer Program for Inelastic Response of R/C Frames to Earthquakes," Civil Eng. Studies, Structural Research Series No. 413, Univ. of Illinois, Urbana, Nov. 1974.
14. Roesset, J. M., Whitman, R. V., Dobry, R., "Modal Analysis for Structures with Foundation Interaction," Journal of the Structural Division, ASCE, Vol. 99, No. ST3, March 1973, pp. 399-416.
15. Tajimi, H., Ishimaru, T., "Ductility Factor Control Method," Trans. of Architectural Institute of Japan, No. 214, Dec. 1973, pp. 17-28.
16. Takeda, T., Sozen, M. A., and Nielsen, N.N., "Reinforced Concrete Response to Simulated Earthquakes," Journal of the Structural Division, ASCE, Vol. 96, No. ST12, Dec. 1970, pp. 2557-2573.
17. Umemura, H., Osawa, Y., Shibata, A., "Study on Shearing Forces in Structures Caused by Medium Earthquakes Recorded in Japan," Proceedings, Third World Conference on Earthquake Engineering, New Zealand, 1965, pp. IV 539-554.

APPENDIX II. - NOTATION

- A_{\max} = Maximum characteristic acceleration for a ground motion record
 E = Young's modulus for concrete
 F_i = Design force for frame member i
 F_{irss} = Square root of the sum of the squares of design forces for member i
 FM = First mode
 g = Acceleration due to gravity
 I = cross-sectional moment of inertia; $-_c$ of column; $-_b$ of beam; $-_{ai}$ of element i in actual frame
 k = frame member flexural stiffness; $-_{ai}$ of actual-frame member i ; $-_{si}$ of substitute-frame member i
 L = length of frame member
 M = moment; $-_{ai}$ at end a of member i ; $-_{bi}$ at end b of member i
 N = number of stories
 P_i = strain energy of frame member i
 RSS = square root of sum of the squares
 S_{Am} = spectral response acceleration for mode m
 V = base shear; $-_{r_{ss}}$ square root of sum of the squares of modal base shears; $-_{abs}$ maximum value for absolute sum of any two of the modal base shears.
 β = damping factor, ratio of value of equivalent viscous damping to the critical value.
 β_s = substitute damping (Eq. 3), $-_{si}$ for frame member i ,
 β_m = smeared damping (Eq. 4) for mode m of substitute structure
 μ = damage ratio (Eq. 2), $-_i$ for member i
 θ = end rotation; $-_y$ at yield
 v = ratio of base shear to weight of structure; $-_m$ for mode m ; $-_{abs}$ maximum for sum of any two v_m ; $-_{r_{ss}}$ Square root of sum of v_m^2

Table 1

Design Shear Coefficients for Frames with Rigid Beams,
Design Spectrum A, $\mu = 6$

No. of stories	Uniform Stiffness ^a		Variable Stiffness	
	Base Shear	Top Story Shear	Base Shear	Top Story Shear
	Total Weight	Base Shear	Total Weight	Base Shear
2	0.61	0.63	0.60	0.67
4	0.31	0.41	0.31	0.49
6	0.21	0.34	0.20	0.39
8	0.16	0.29	0.15	0.34
10	0.13	0.26	0.12	0.30

^aAll stories have the same stiffness

^bStiffness distributed to have a linear first-mode shape

Note: Shear coefficients given refer to yield capacity of structural elements and a characteristic base acceleration of 0.5g.

Table 2

Calculated Periods and Smeared Damping Factors for
3, 5 and 10-Story Frames with Flexible Beams

Mode	Period			Damping Factor ^d	Mode	Period			Damping Factor ^d
	Uncracked ^a	Cracked ^b	Substitute ^c			Uncracked ^a	Cracked ^b	Substitute ^c	
3-Story Frame					10-Story Frame				
	sec	sec	sec						
1	0.34	0.50	0.85	0.086	1	0.95	1.58	3.18	0.106
2	0.10	0.14	0.19	0.045	2	0.30	0.49	0.87	0.081
3	0.056	0.074	0.078	0.025	3	0.17	0.27	0.39	0.050
					4	0.11	0.17	0.22	0.038
					5	0.075	0.11	0.14	0.032
5-Story Frame					6	0.056	0.083	0.093	0.027
1	0.53	0.85	1.58	0.099	7	0.043	0.063	0.068	0.024
2	0.17	0.26	0.41	0.068	8	0.035	0.051	0.053	0.022
3	0.090	0.14	0.18	0.041	9	0.030	0.043	0.044	0.021
4	0.059	0.087	0.097	0.028	10	0.028	0.039	0.040	0.020
5	0.046	0.065	0.067	0.022					

^aBased on gross plain section

^bBased on transformed cracked section (assumed).

^cBased on stiffness properties from Eq. 2 with $\mu = 6.0$ for beams and $\mu = 1.0$ for columns.

^dFrom Eq. 3, 4, and 5 and for the substitute structure.

Table 3

Effect of Column Overstrength Factor on Damage Ratio
5-Story Frame, El Centro 1940 EW 0.5G

		<u>Calculated Damage Ratios</u>		
		<u>Frame X</u>	<u>Frame Y</u>	<u>Frame Z</u>
Beam	5	5.2	6.9	7.9
	4	6.1	7.3	8.3
	3	8.1	8.4	8.6
	2	8.2	8.3	8.2
	1	6.7	7.0	7.5
Column	5T	1.5	0.96	0.85
	B	0.35	0.36	0.38
	47	3.7	2.3	0.97
	B	0.39	0.42	0.40
	3T	1.0	0.94	0.81
	B	0.77	0.78	0.75
	2T	0.54	0.49	0.43
	B	3.0	2.2	0.95
	1T	0.31	0.30	0.28
	B	0.95	0.90	0.79

T: top of column

B: bottom of column

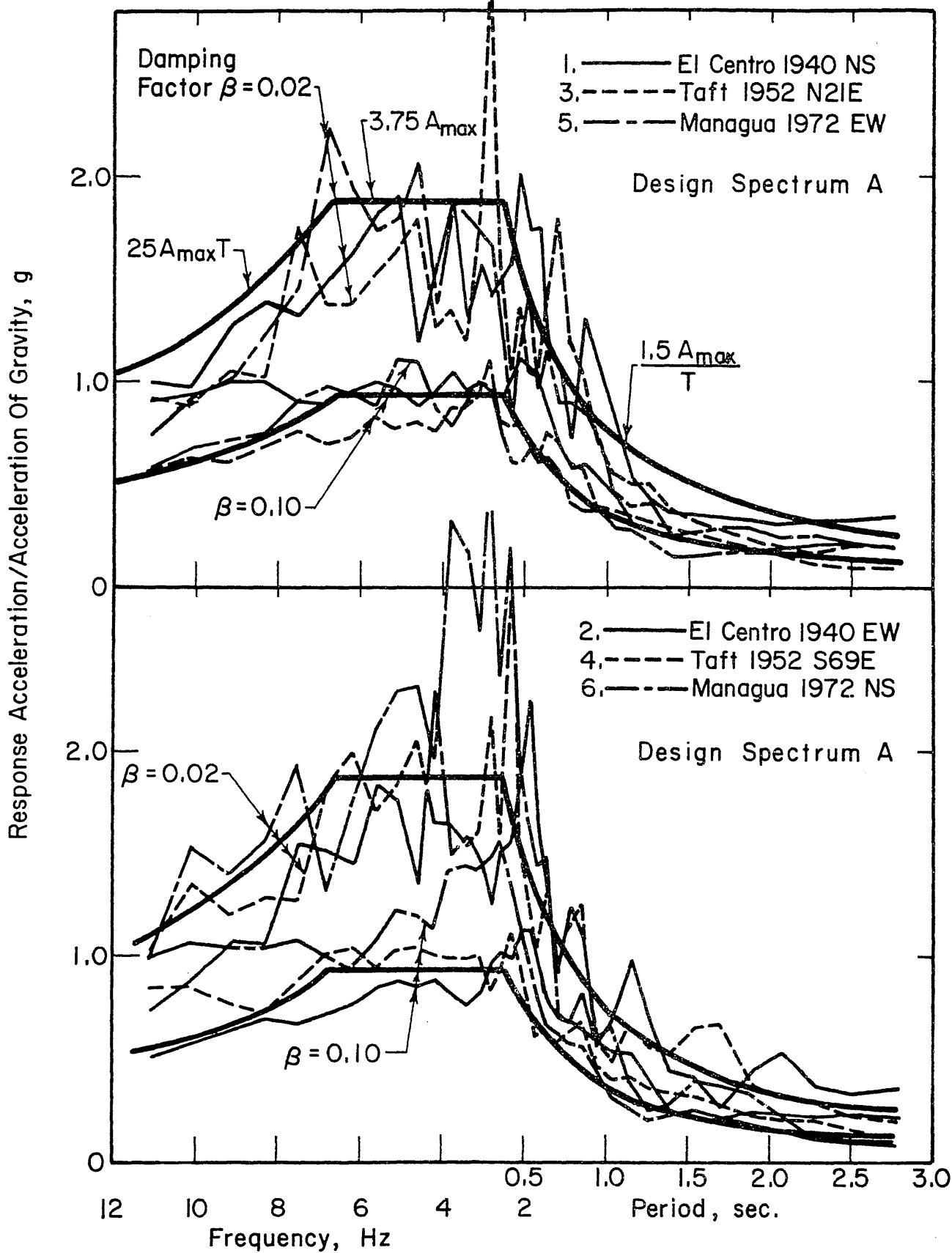


Fig. 1 Acceleration Response to Ground Motions 1 through 6 (Normalized to 0.5g) and Design Acceleration-Response Spectrum A.

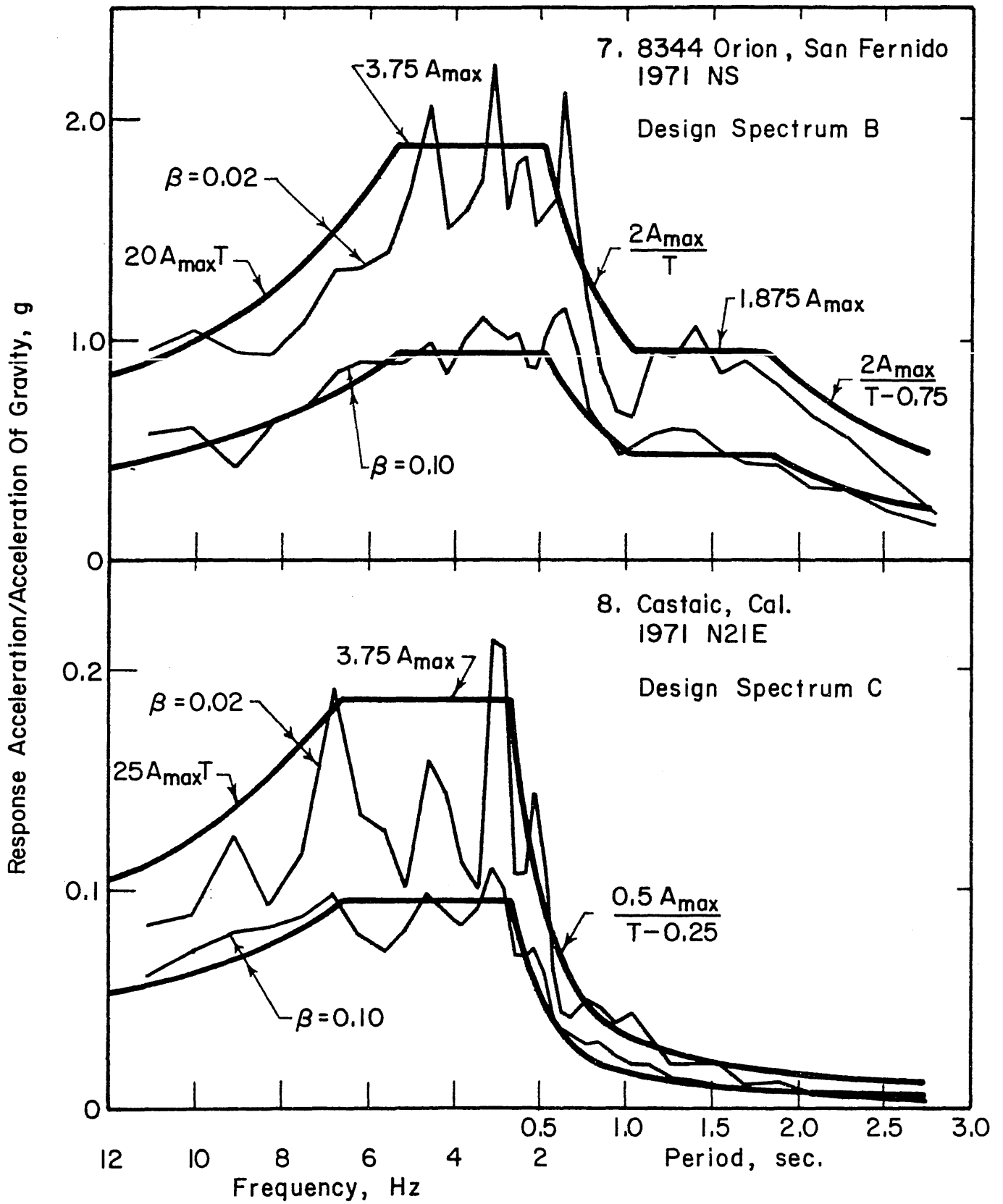


Fig. 2 Acceleration Response to Ground Motions 7 & 8 and Design Acceleration-Response Spectra B and C.

Mark	Location	Date	Direction	Max. Acc./g
1	El Centro, Calif.	28 May 1940	NS	0.31
2	El Centro, Calif.	28 May 1940	EW	0.22
3	Taft, Calif.	21 July 1952	N2IE	0.18
4	Taft, Calif.	21 July 1952	S69E	0.16
5	Managua, Nicaragua	23 Dec. 1972	EW	0.38
6	Managua, Nicaragua	23 Dec. 1972	NS	0.33
7	8344 Orion, San Fern'do	9 Feb. 1971	NS	0.26
8	Castaic, Cal.	9 Feb. 1971	N2IE	0.32

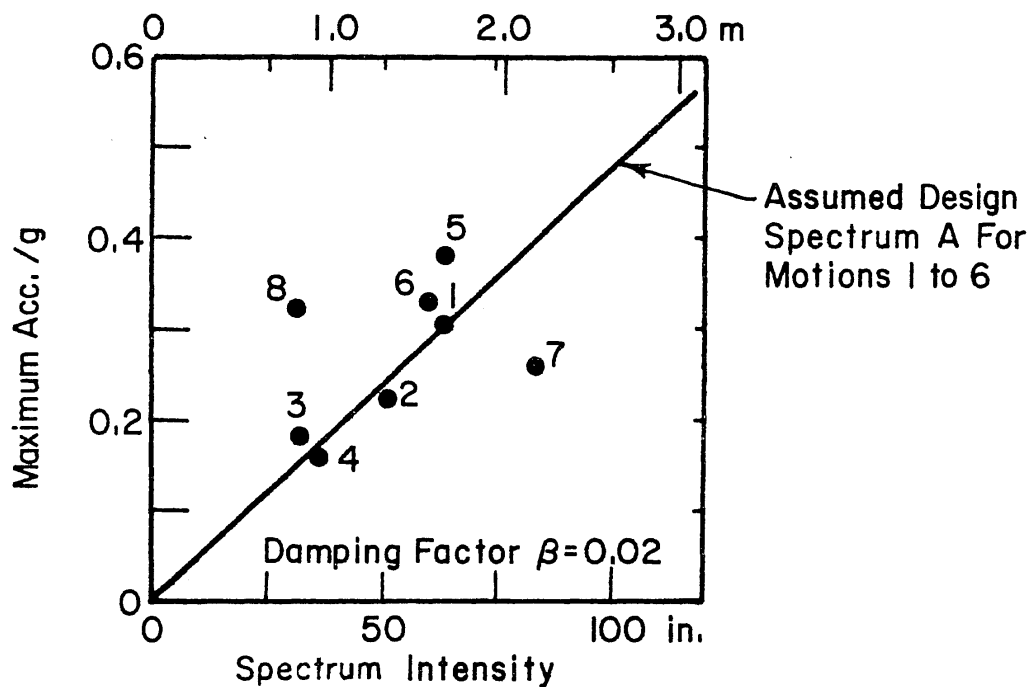


Fig. 3 Comparison of Maximum Acceleration with Spectrum Intensity

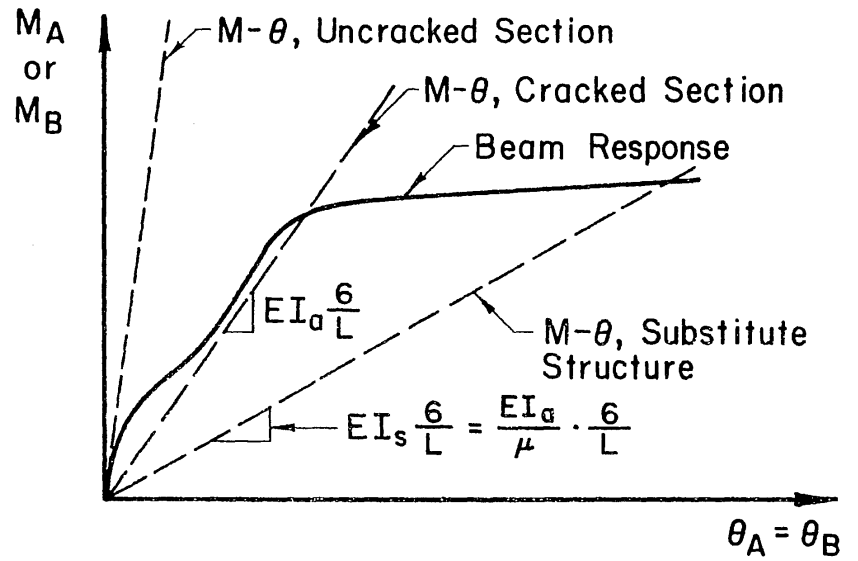
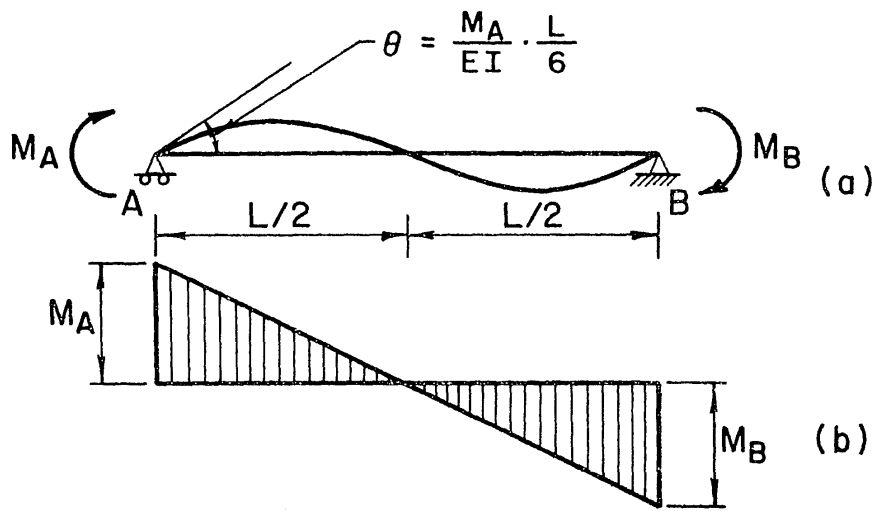
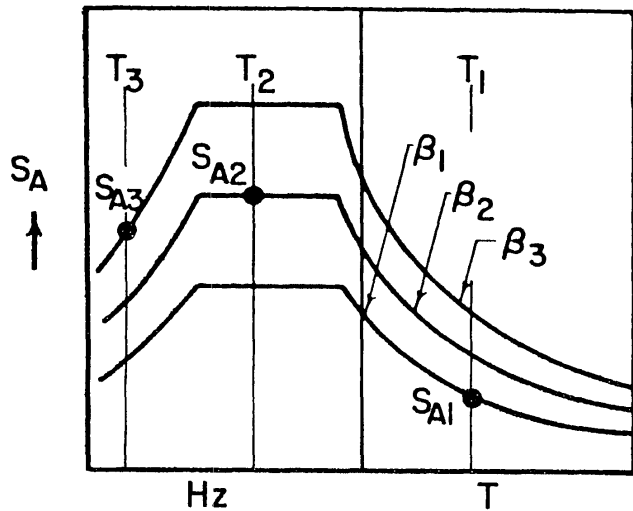
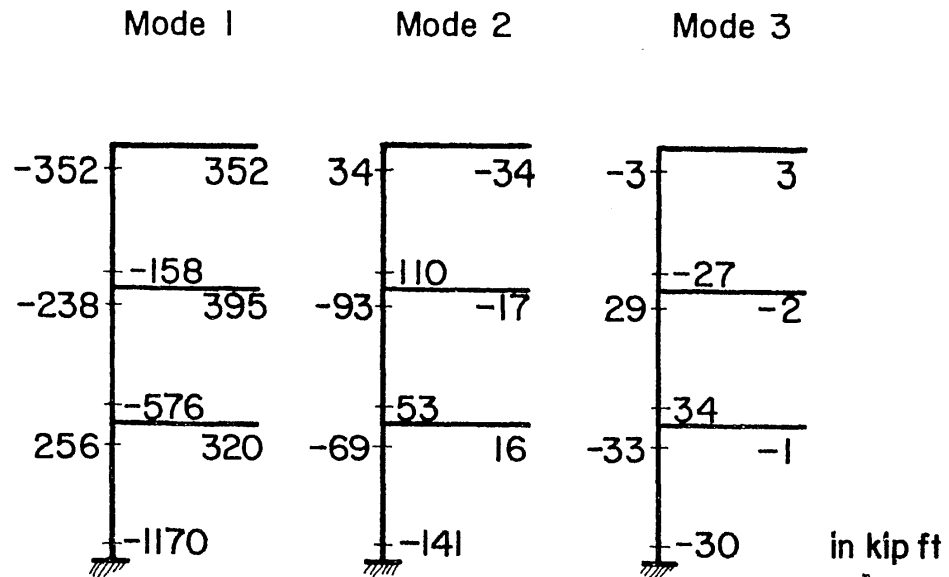


Fig. 4 Interpretation of Damage Ratio

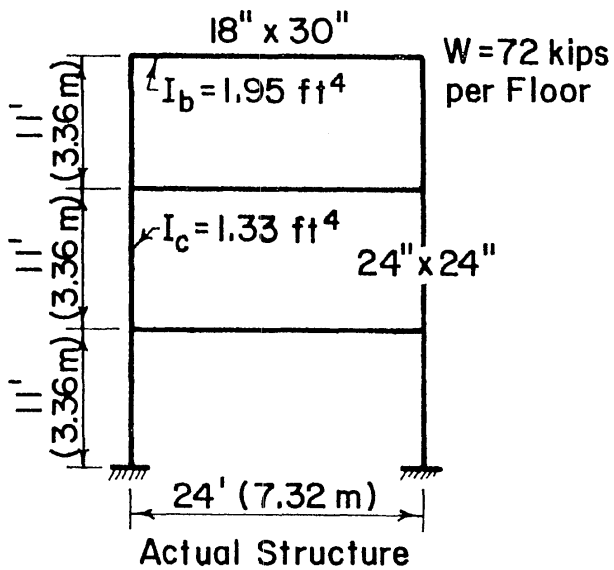
University of Illinois
 Metz Reference Room
 E106 NCEM
 203 N. Romaine Street
 Urbana, Illinois 61801



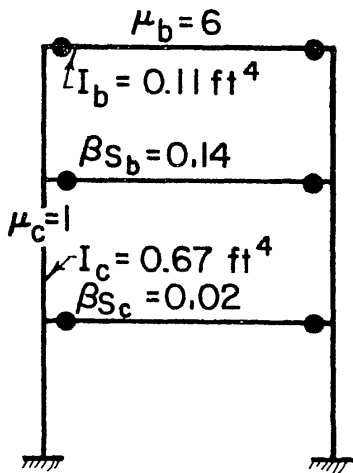
(a) Design Spectrum



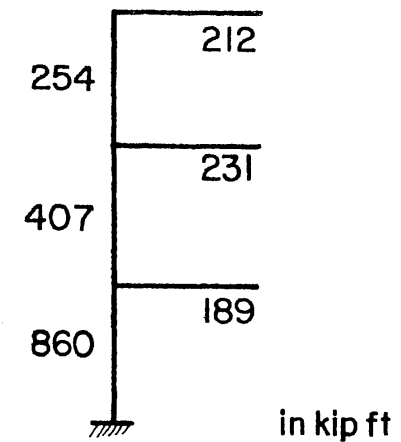
(c) Modal Moments Of Substitute Structure For Spectral Acc. S_A Of I_g



(b) Substitute Structure



Substitute Structure



(d) Design Moments

Fig. 5 Data for Numerical Example (1.0 kip = 4.45 kN; 1.0 ft = 0.30m; 1.0 k-ft = 1.36 kN-m)

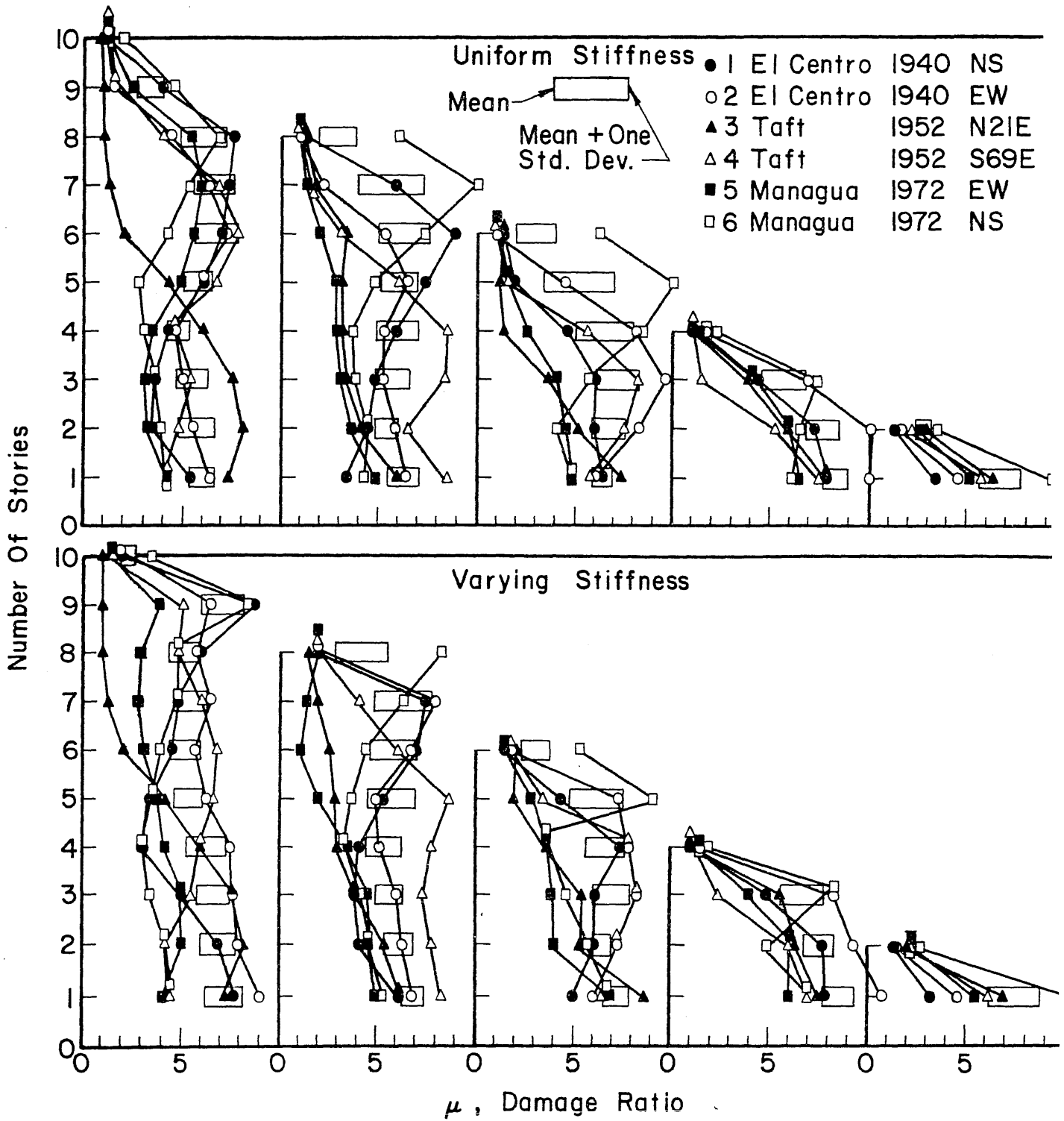


Fig. 6 Calculated Damage Ratios for Frames with Rigid Beams (Ground Motions 1 through 6)

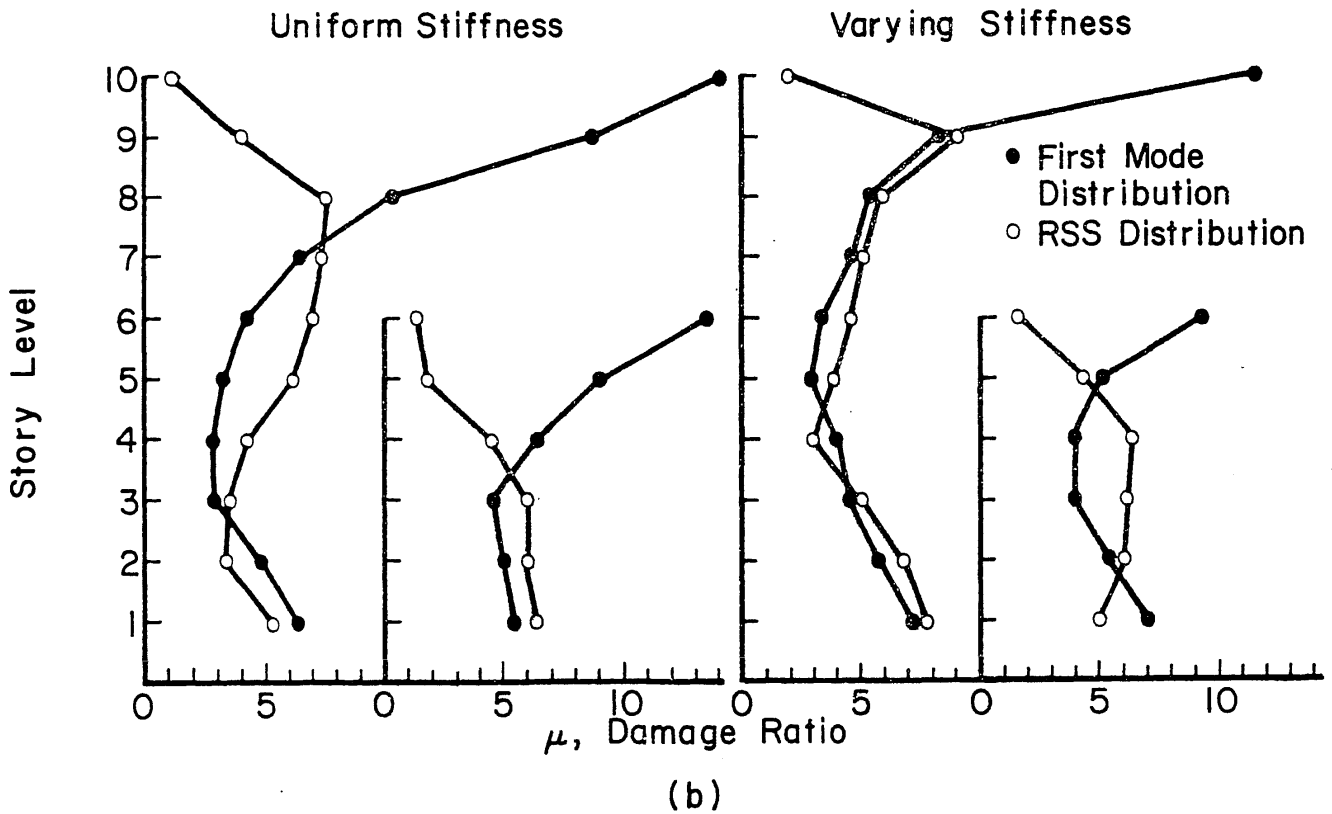
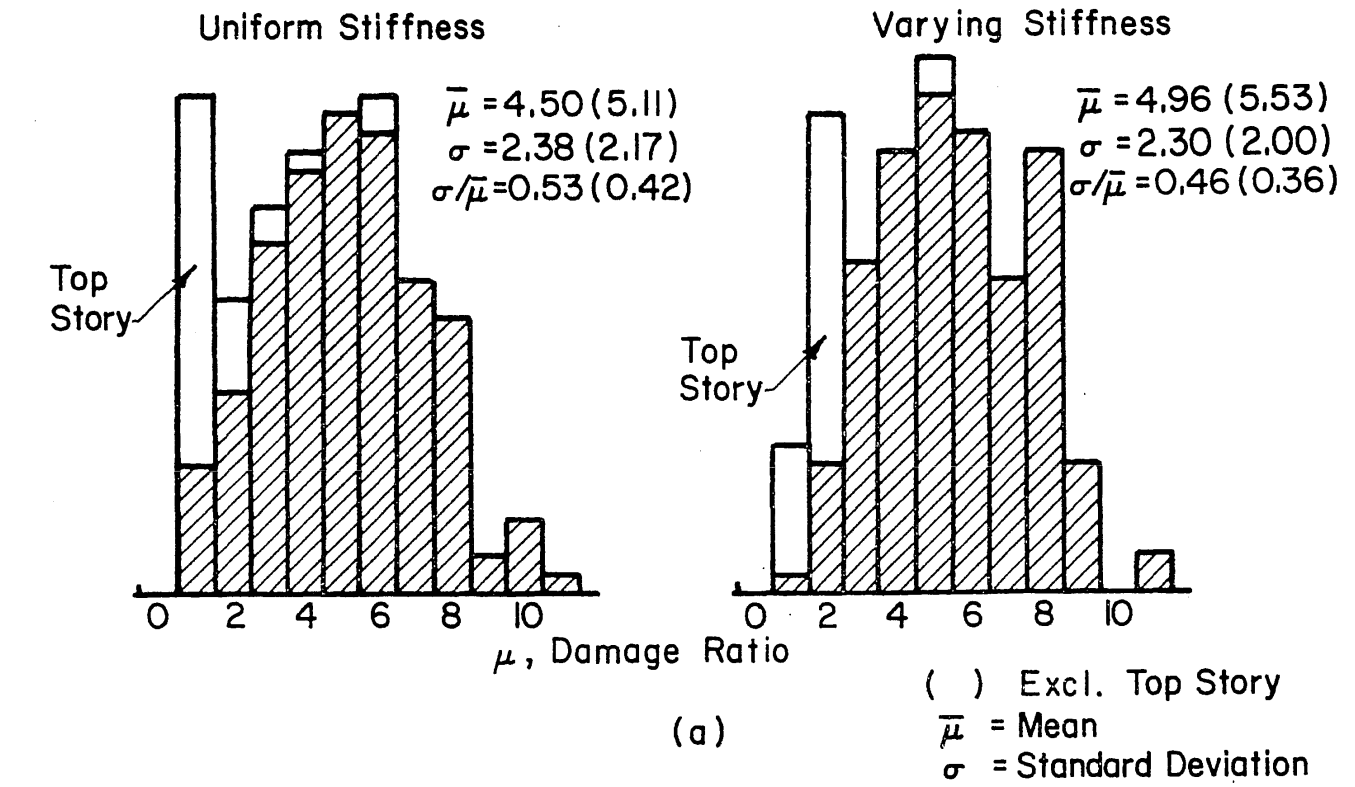


Fig. 7 Distribution of Damage Ratios and Effect of Story Shear Strength Distribution

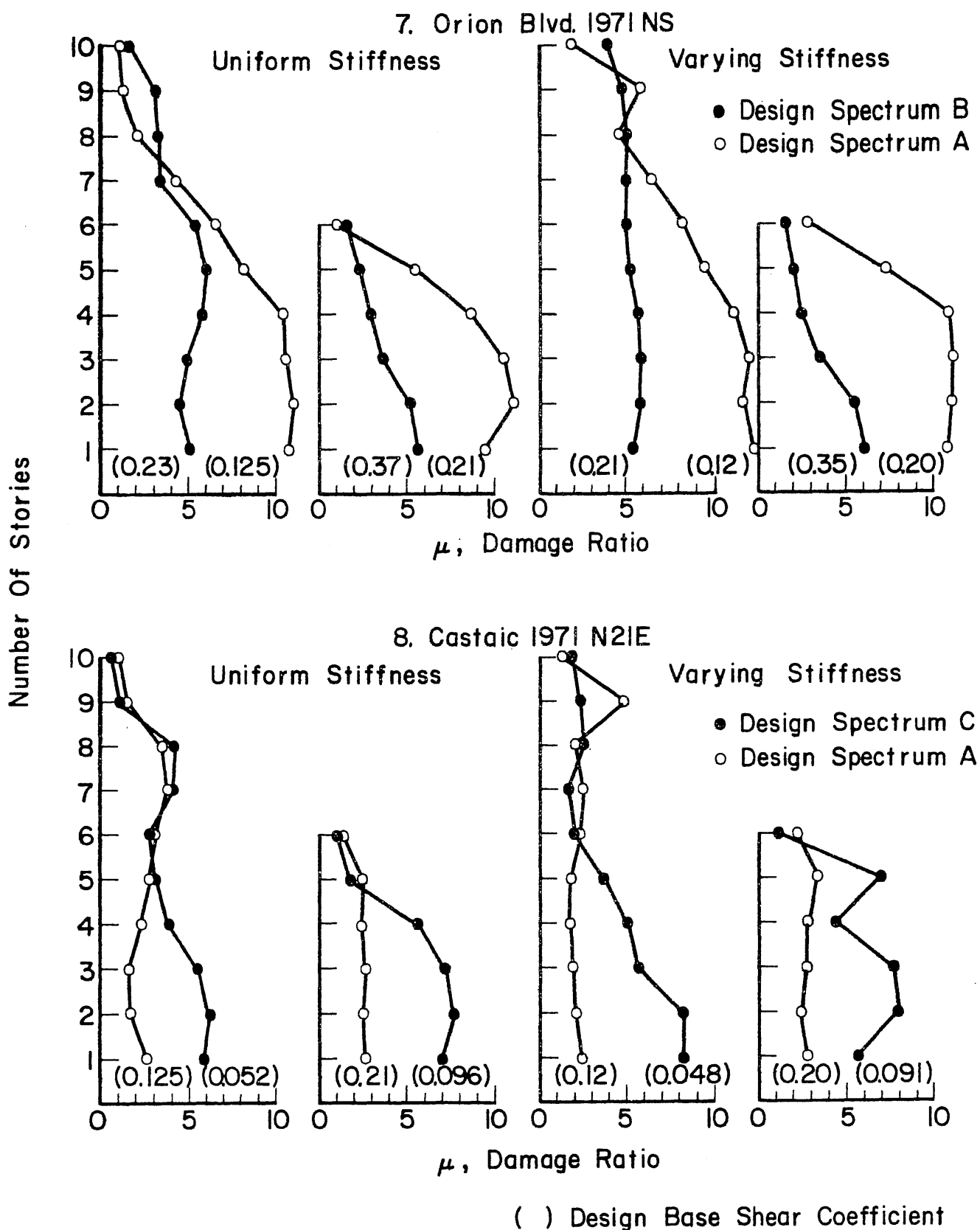


Fig. 8 Calculated Damage Ratios for Frames with Rigid Beams (Ground Motions 7 and 8)

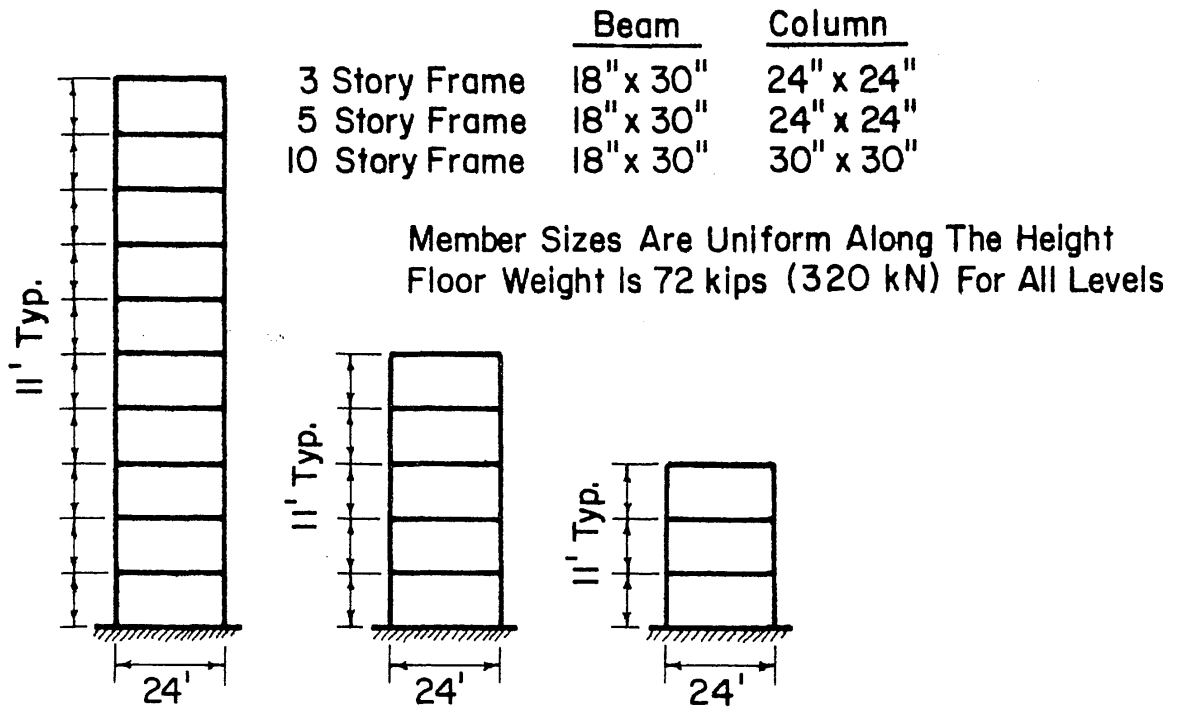


Fig. 9 Properties of Frames with Flexible Beams
 (1.0 in. = 0.025m)

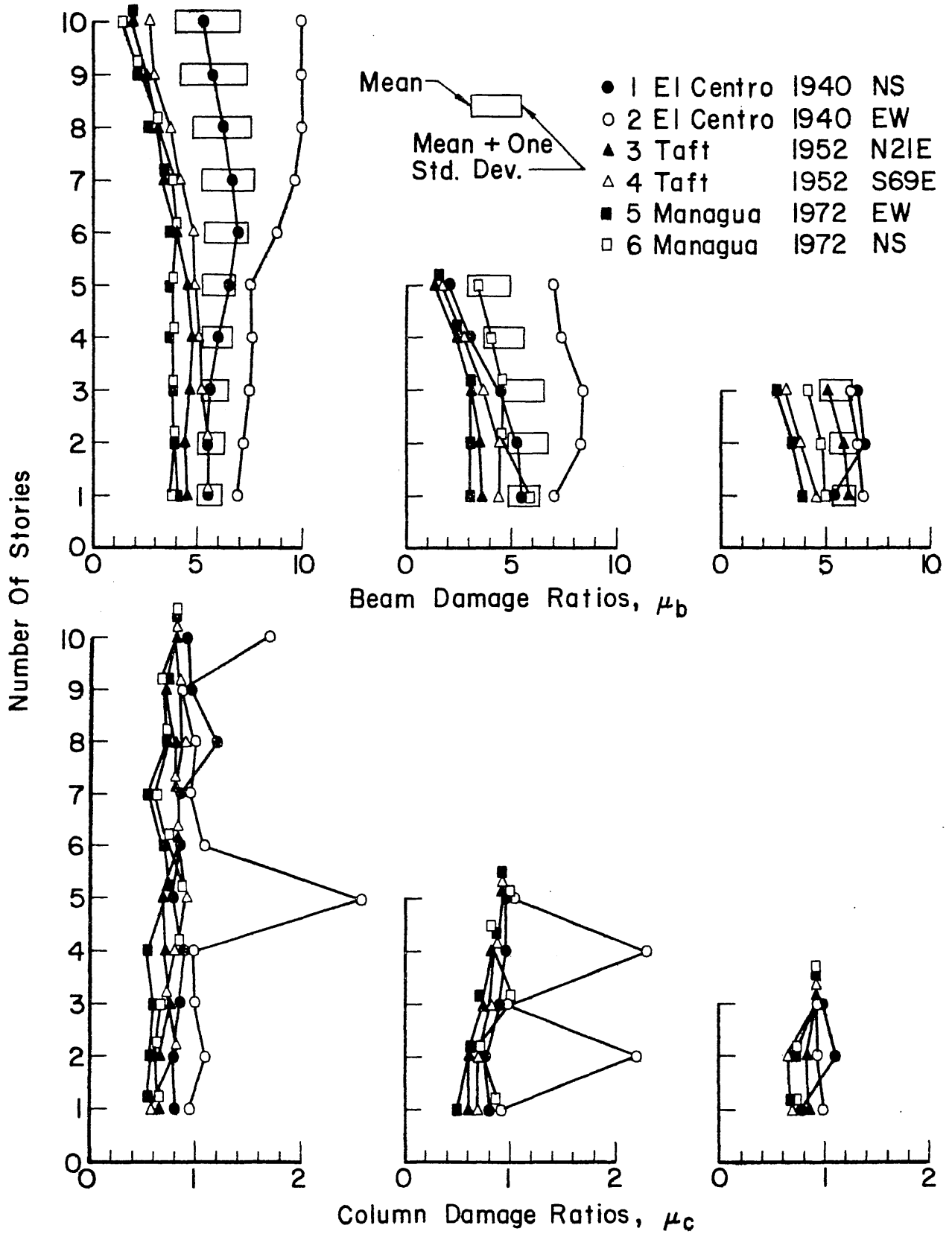


Fig. 10 Calculated Damage Ratios for Frames with Flexible Beams (Ground Motions 1 through 6)

TECHNOLOGY FOR THE FORMULATION
AND EXPRESSION OF SPECIFICATIONS

VOLUME III: TECHNICAL REFERENCE MANUAL

by

R.N. Wright, J.R. Harris, J.W. Melin, and C. Albarran

A report on a research project sponsored by:

THE NATIONAL BUREAU OF STANDARDS

Contract No. 5-35844

DEPARTMENT OF CIVIL ENGINEERING

UNIVERSITY OF ILLINOIS

URBANA, ILLINOIS

December 1975

INTRODUCTION

This report describes the three computer programs produced as a part of the study, "Technology for the Formulation and Expression of Specifications." The programs are written in FORTRAN IV and are operational on the Burroughs B6700 computer at the Civil Engineering Systems Laboratory (CESL), Department of Civil Engineering, University of Illinois, Urbana, Illinois. Transfer of these programs to other computing facilities will be aided by the contents of the manual, because the programs are not entirely machine independent. The programs make use of the characteristic word size of the Burroughs equipment (6 characters). Transfer to IBM equipment (which uses 4 characters) will require some reprogramming of the input and output routines.

The programs are designed for interactive use from a remote terminal. Most of the work done with these programs will not require large amounts of data to be input, so this mode is quite convenient. The input is entered in free format and interpreted by a package of scanning routines at CESL known as PARSE. A manual for PARSE will soon be available from CESL.

These programs are envisioned to be prototypes for a more refined computer aid for use with the technology described in volume 1 of this report. Future improvements will include the linking up of the programs to share similar subroutines and to create a common data base for a specification that would include the information network, all the decision

tables, and the outline structure. Undoubtedly many improvements will be suggested by the users of the programs as experience with them is accumulated.

Each program is described in the following manner:

1. a brief description of the important algorithms used in the program;
2. flow charts and block diagrams for the program structure and the more complex subroutines;
3. a description of the major data structure used in the program, including the permanent data stored on disk;
4. a glossary of the variable names used in the program; and
5. a complete listing of the program.

TECHNICAL REFERENCE MANUAL

TABLE OF CONTENTS

	<u>Page</u>
INTRODUCTION	i
TABLE OF CONTENTS	iii
LIST OF FIGURES	iv
CHAPTER	
1. DECISION TABLE PROGRAM	1
1.1 Algorithms	1
1.2 Logic Diagrams	4
1.3 Data Structure	15
1.4 Glossary	20
1.5 Program Listing	25
2. INFORMATION NETWORK PROGRAM	46
2.1 Algorithms	46
2.2 Logic Diagrams	48
2.3 Data Structure	57
2.4 Glossary	61
2.5 Program Listing	66
3. OUTLINE PROGRAM	86
3.1 Algorithms	86
3.2 Logic Diagrams	87
3.3 Data Structure	92
3.4 Glossary	97
3.5 Program Listing	100
REFERENCES	113

LIST OF FIGURES

FIGURE	Page
1.1 Block Diagram for the Decision Table Program	5
1.2 Subroutine Linkage (TABLE)	6
1.3 Subroutine INPUT (TABLE)	7
1.4 Input of the Rules (TABLE)	8
1.5 Input of the Conditions (TABLE)	8
1.6 Subroutine TABTRE (TABLE)	9
1.7 Selection of Condition in TABTRE (TABLE)	10
1.8 Examination of Rules in TABTRE (TABLE)	11
1.9 Subroutine TOPO (TABLE)	12
1.10 Subroutine ARRAY (TABLE)	13
1.11 Subroutine SORT (TABLE)	14
1.12 Permanent Data (TABLE)	16
1.13 Permanent Data File (TABLE)	17
1.14 Data Used in Decomposition (TABLE)	18
1.15 Data Used in Describing the Network (TABLE)	19
2.1 Block Diagram for the Information Network Program	49
2.2 Subroutine Linkage (NETWORK)	50
2.3 Subroutine INPUT (NETWORK)	51
2.4 Input of Data for a New Node (NETWORK)	52
2.5 Input of New Data for a Node (NETWORK)	53
2.6 Subroutine LEVEL (NETWORK)	54
2.7 Subroutine SORT (NETWORK)	55
2.8 Subroutine GLOBAL (NETWORK)	56
2.9 Permanent Data (NETWORK)	58
2.10 Permanent Data File (NETWORK)	59
2.11 Temporary Data (NETWORK)	60
3.1 Block Diagram for the Outline Program	88
3.2 Subroutine Linkage (OUTLINE)	89
3.3 Subroutine OUTLIN (OUTLINE)	90
3.4 Subroutine ADVANC (OUTLINE)	91
3.5 Argument Data (OUTLINE)	93
3.6 Provision Data (OUTLINE)	94
3.7 Outline Data (OUTLINE)	95
3.8 Permanent Data File (OUTLINE)	96

Chapter One

DECISION TABLE PROGRAM

1.1 Algorithms

The purpose of this program is to check the condition entry portion of a decision table to see if it is logically complete and correct. That is, to see if all possible rules are contained in the table and that none of the rules are redundant or contradictory. The program does not perform any check on the logical relations between related conditions, the actions, or the action entry. The condition stubs, action stubs, and action entry that are entered into the program are only used to aid the user's interpretations of the output.

The method used to check the decision table is the decomposition of the table into a network by condition testing. Fenves (ref. 1.1) and Pollack (ref. 1.2) describe the method, originally due to Mantalbano (ref. 1.3). Essentially, the algorithm can be summarized in the following steps:

1. Begin with the original table
2. Select a condition to test. This is the step which differs from one algorithm to the next, and it will be discussed below in more detail.
3. Discriminate on the condition to produce two subtables, each with one less condition than the previous table. One subtable contains all the rules for which the tested condition has a true or immaterial entry, the other contains those rules for which it has a false or immaterial entry.

4. If the subtable contains at least one condition and more than one rule, return to step 2; if not, go to step 5.
5.
 - a) If the subtable contains exactly one rule and no remaining condition entries that are explicitly true or false, then that rule has been isolated.
 - b) If the subtable contains one rule and some remaining explicit condition entries, return to step 2.
 - c) If the subtable contains no rules, an else rule has been isolated (that is, a rule not included in the original table.)
 - d) If the subtable contains no remaining conditions, but does contain two or more rules, those rules are redundant or contradictory (that is, they can be satisfied by the same set of condition values.)

The algorithm produces a network in which each node is a condition with one branch entering it and two branches (true and false) leaving it, except for the terminal nodes, which are rules. The topology of the network depends on the order in which the conditions are tested. Many such networks can be generated from one decision table, but the rules in all of these networks will be logically equivalent.

Algorithms for selecting conditions that produce optimum networks in terms of core storage or running time required have been of great interest to many authors, but tend to become complex and of doubtful value (1.2). The particular algorithm used in this program is directed towards producing an optimum network, but no guarantee is made that the resultant

network optimizes any quantity. The steps used in selecting the condition to be tested from a subtable are as follows:

1. Count the immaterial entries for each condition in the subtable, considering only those rules contained in the subtable. Select the condition with the smallest number of immaterial entries. If two or more are tied, go to step 2.
2. Count the explicit (true or false) entries for each of the conditions that were tied at the end of step 1. Select the condition with the largest number of explicit entries. If two or more are tied, go to step 3.
3. Find the absolute value of the difference between explicitly true and explicitly false entries among those conditions that were tied at the end of step 2. Select the condition with the largest such value. If two or more are tied, arbitrarily select the one listed first in the original table.

Note that tables with no immaterial or implicit entries will always be tested by step 3. The test used in step 3 is the "quick rule" as described by Fenves (1.1). The program gives the user the option of using the "delayed rule," also as described by Fenves. For the delayed rule, step 3 is changed so that the condition with the minimum value, rather than the maximum, is selected.

The important idea behind the algorithm for selecting the condition is to avoid testing immaterial or implicit entries, thus producing a smaller network. Not much literature has been available on algorithms for tables using implicit entries. The method used is fairly simple and has seemed to produce reasonable results.

The program displays the results to the user by means of a printed version of the network. All rules that were not contained in the original table are labeled ELSE. The output is formulated in the machine by producing an array of alphameric data, which is then simply printed out line by line.

1.2 Logic Diagrams

The following diagrams describe the structure of the program and the major subroutines. For a detailed study of the program, consult the listing of the FORTRAN code.

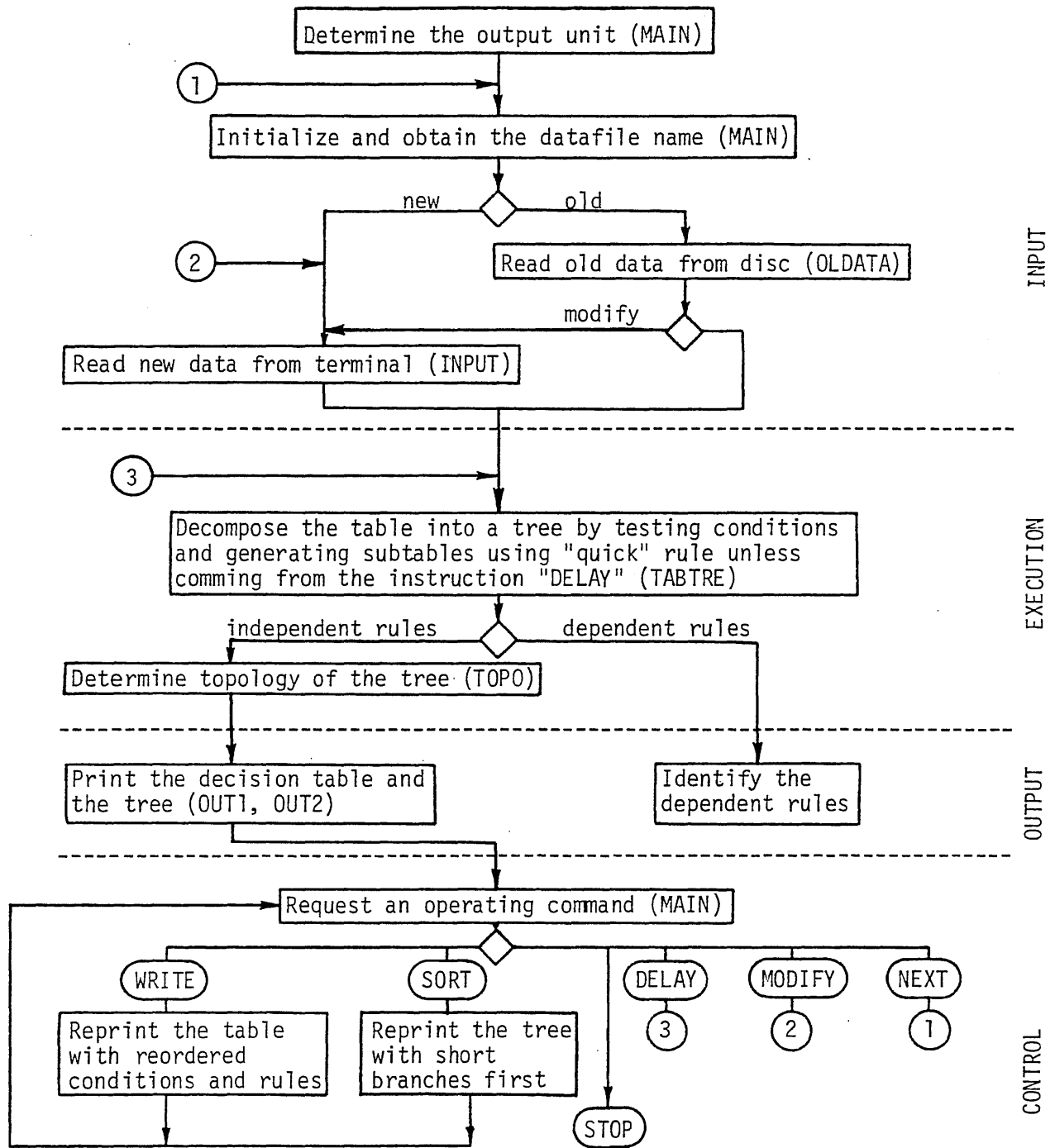
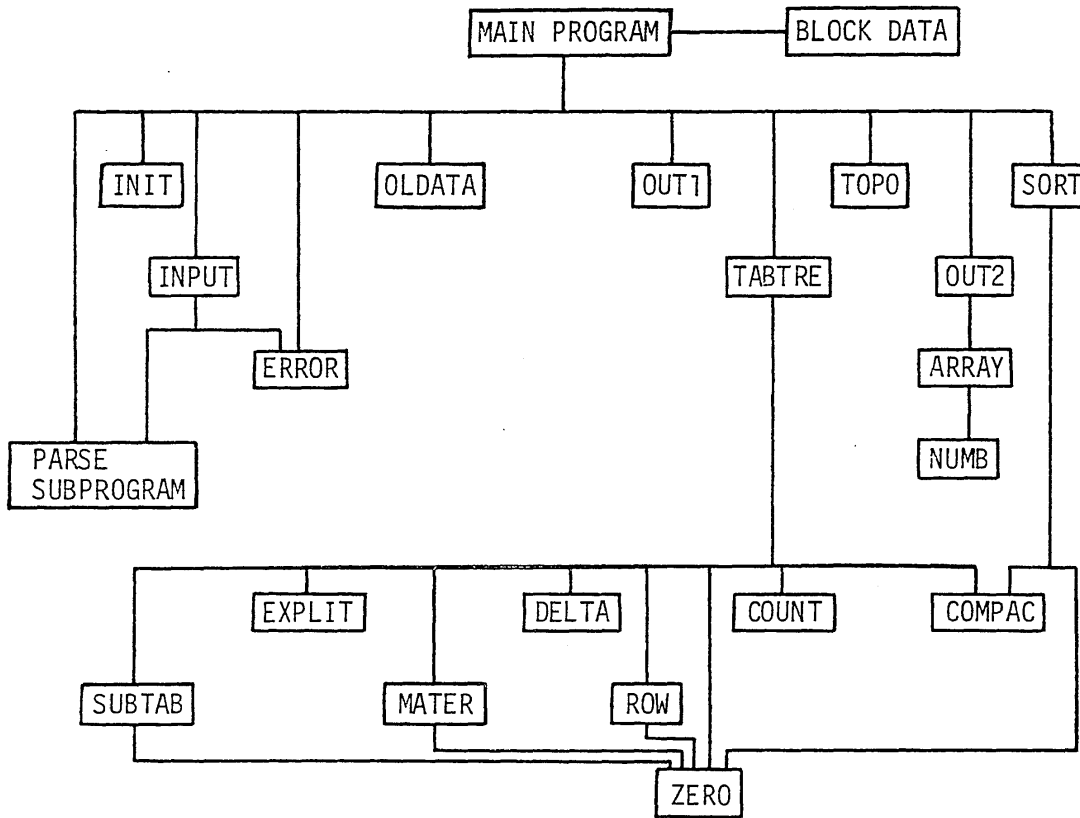


Figure 1.1 Block Diagram for the Decision Table Program



NOTES

1. All routines except ERROR, COMPAC, ZERO and the PARSE subprogram contain TABCOM, the COMMON declarations.
2. The MAIN PROGRAM and INPUT also contain PARCOM, the COMMON declarations for the use of PARSE.
3. PARSE contains several subroutines and functions, including WDINIT, RDLIN, END, MATCH, FIXED, STRING, and MODE.

Figure 1.2 Subroutine Linkage (TABLE)

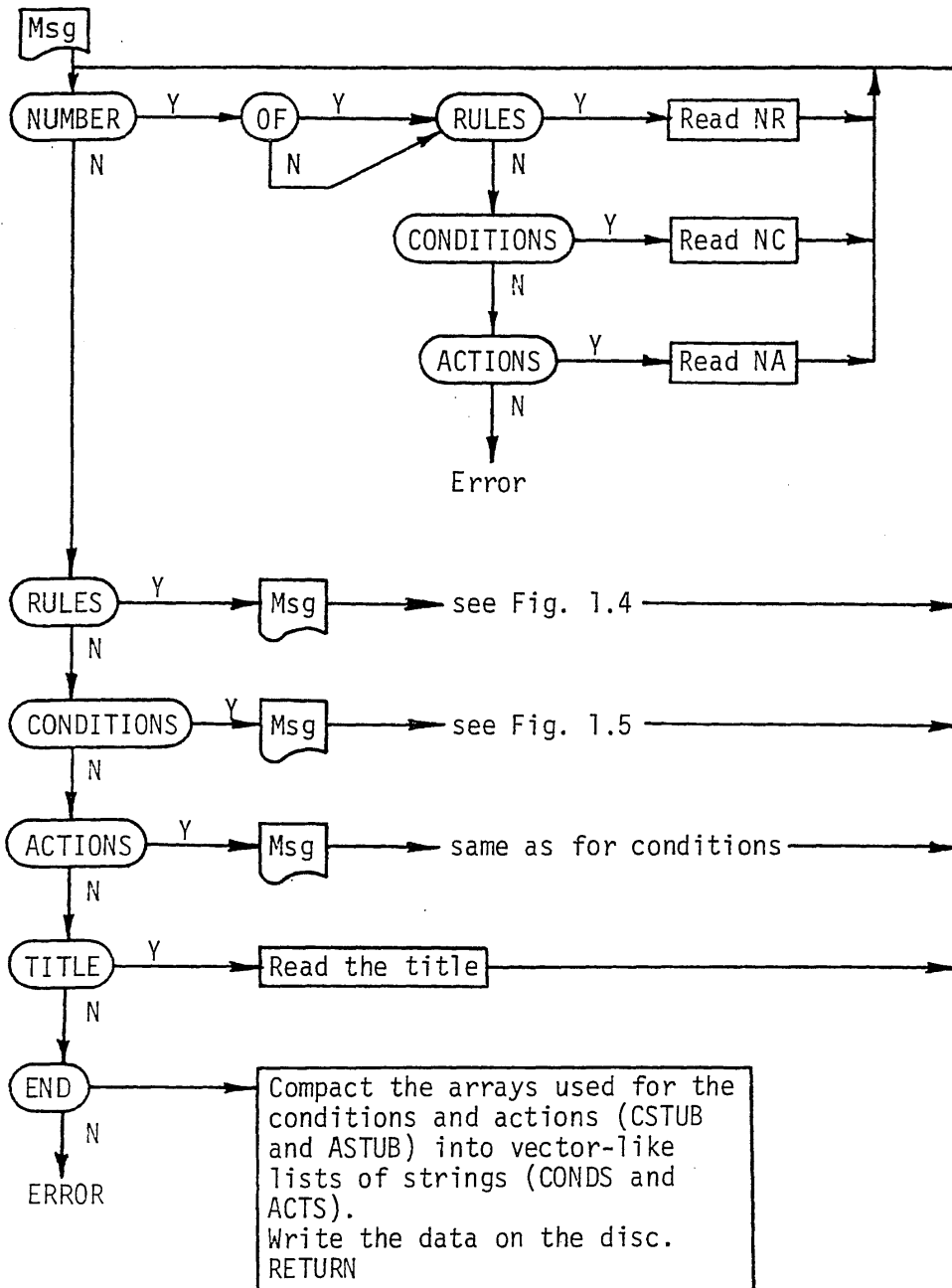


Figure 1.3 Subroutine INPUT (TABLE)

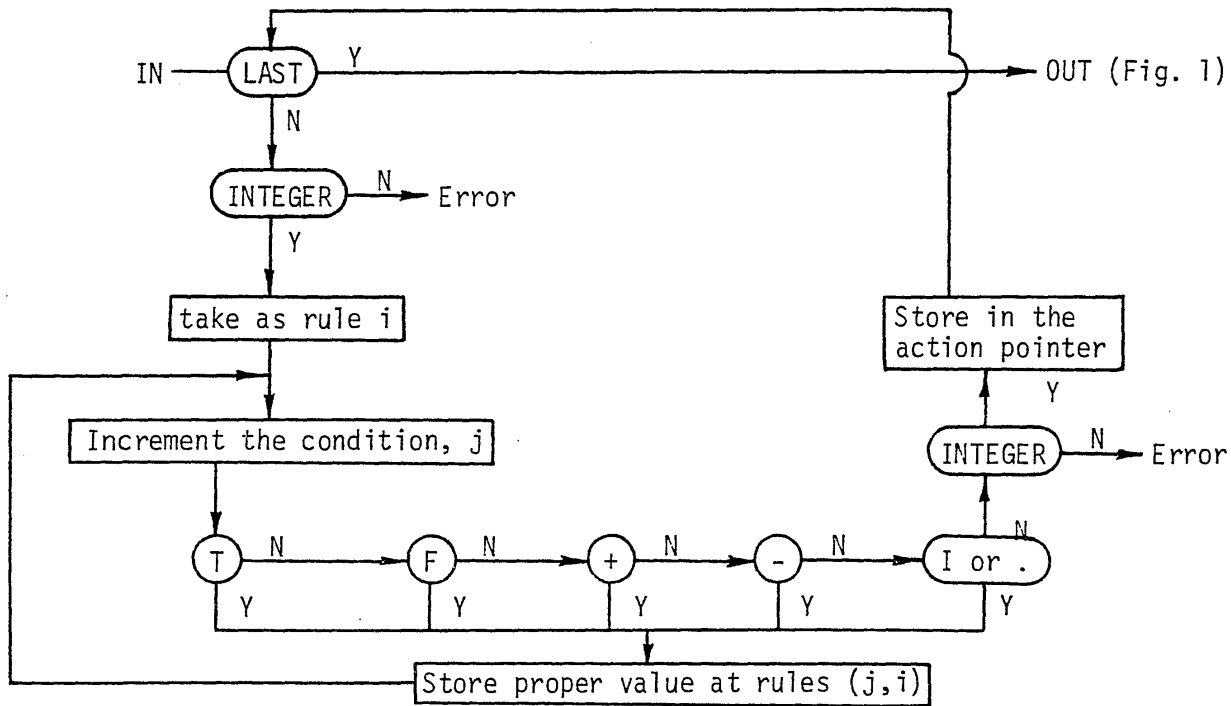


Figure 1.4 Input of the Rules (TABLE)

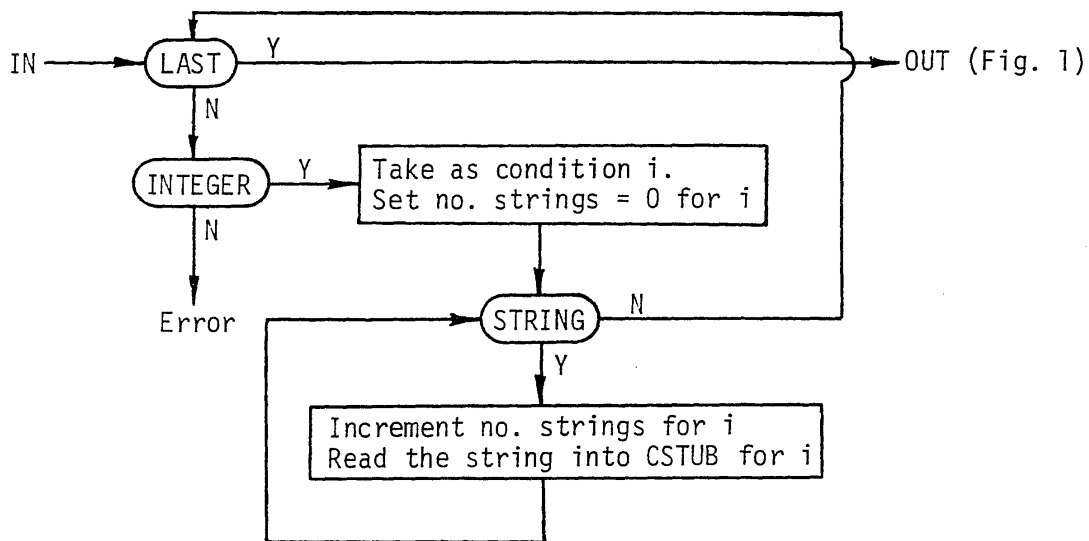


Figure 1.5 Input of the Conditions (TABLE)

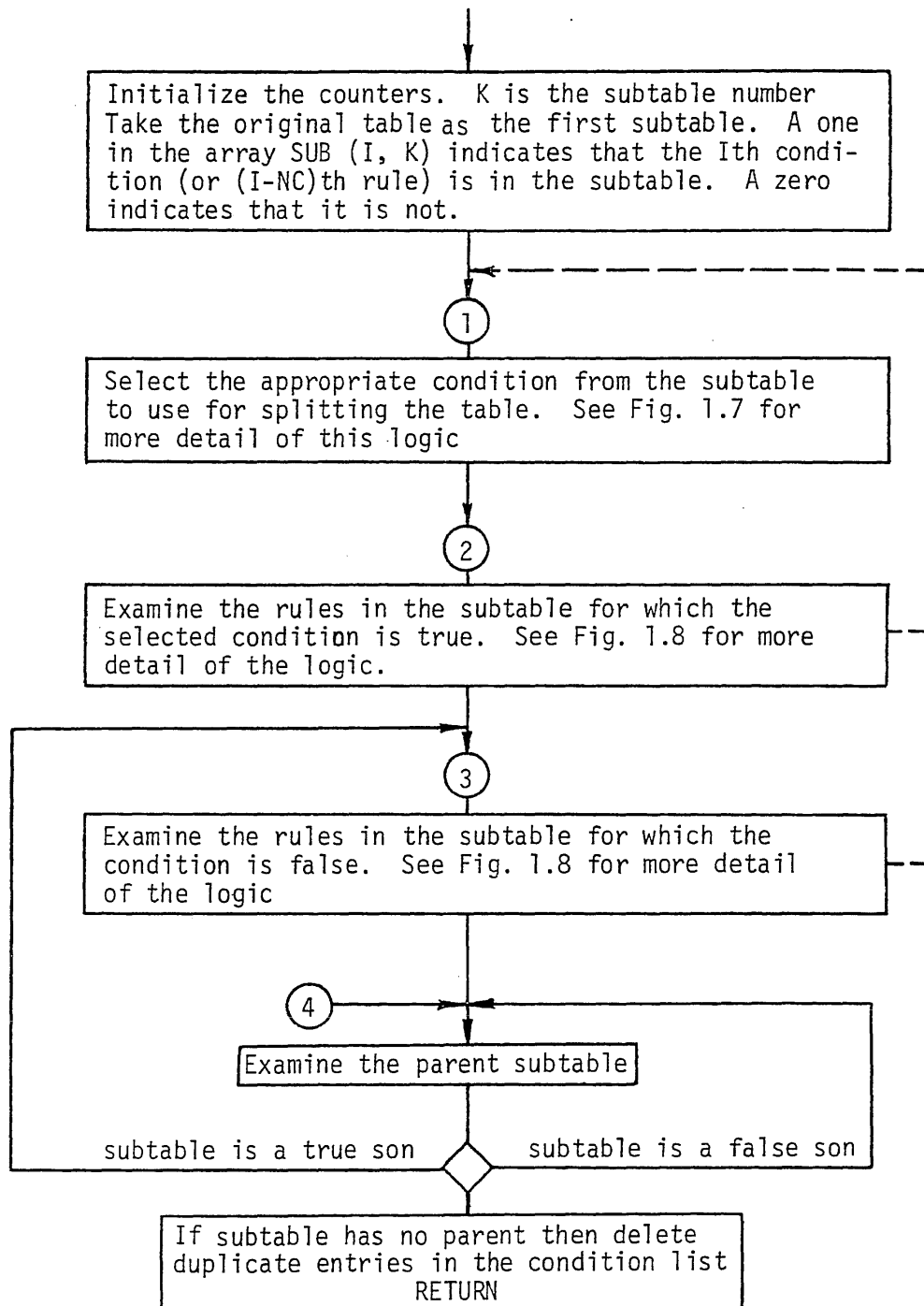


Figure 1.6 Subroutine TABTRE (TABLE)

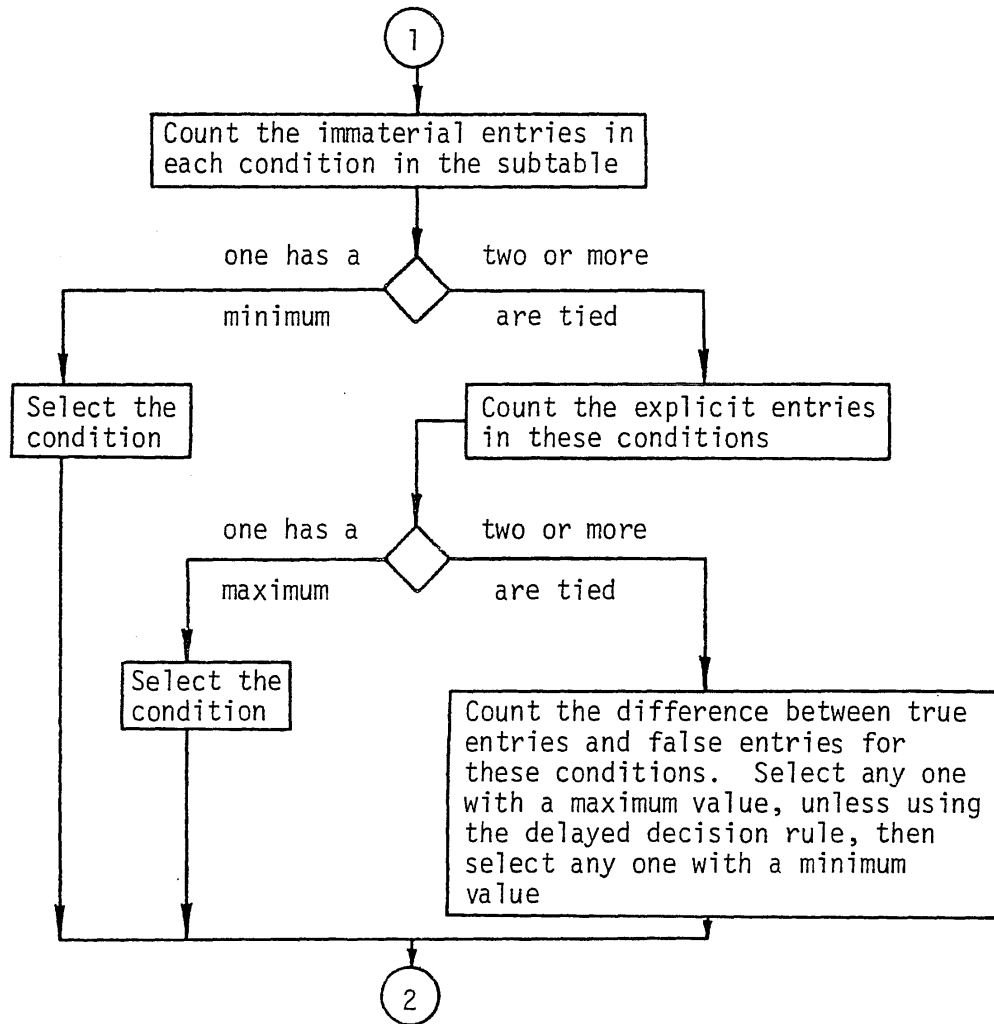
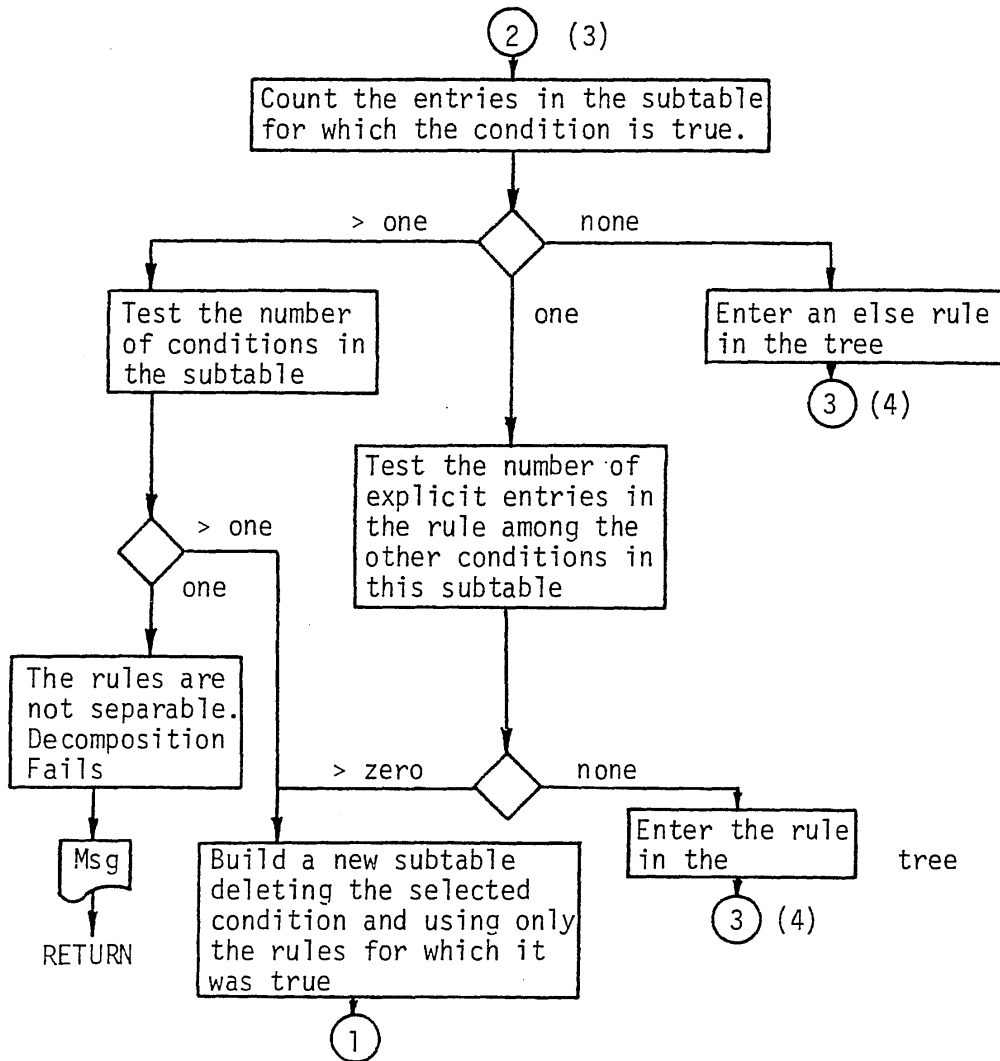


Figure 1.7 Selection of Condition in TABTRE (TABLE)



Note the logic for the rules with false entries is the same; simply replace the word true with the word false and the reference numbers with those in parenthesis.

Figure 1.8 Examination of Rules in TABTRE (TABLE)

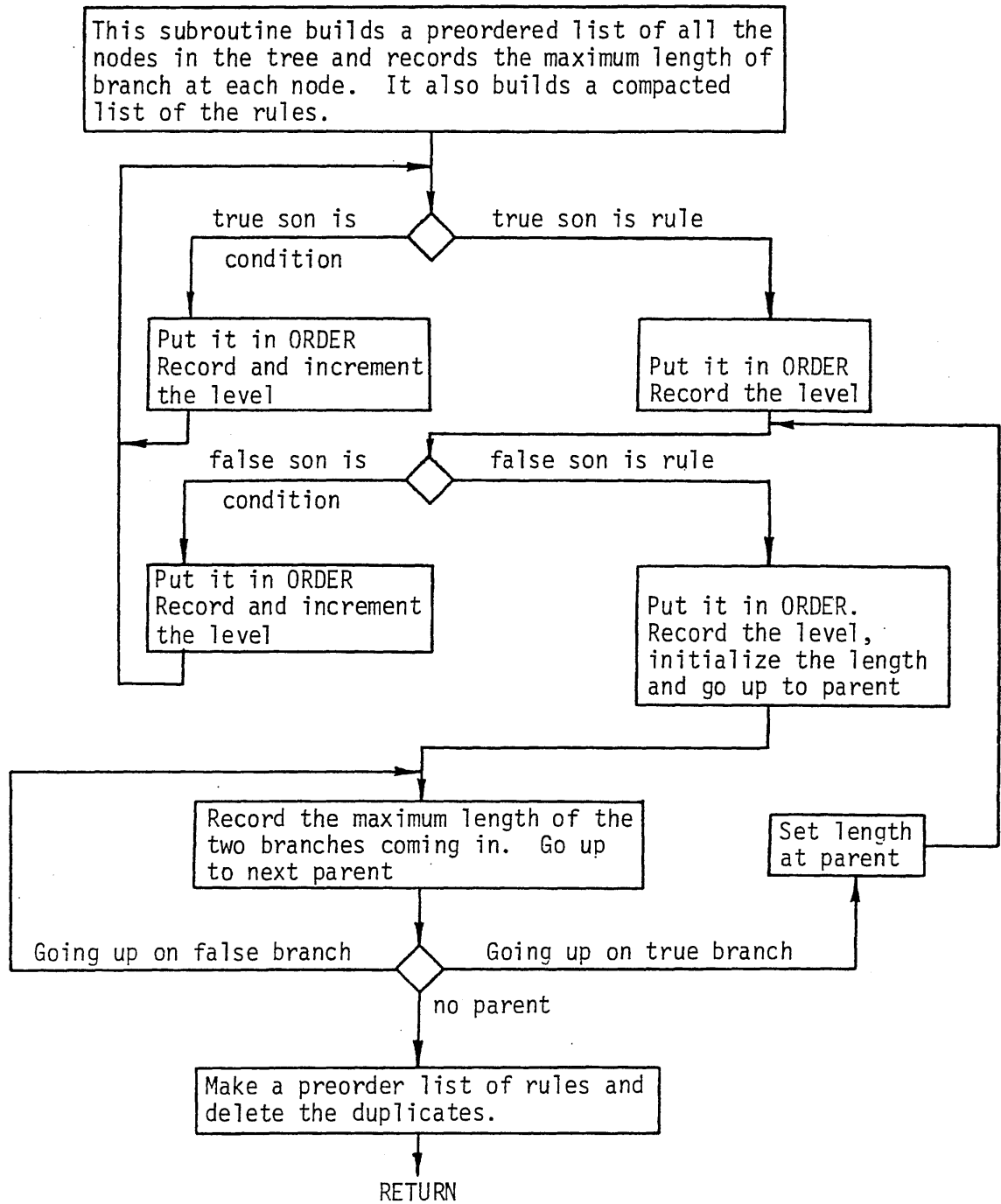


Figure 1.9 Subroutine TOPO (TABLE)

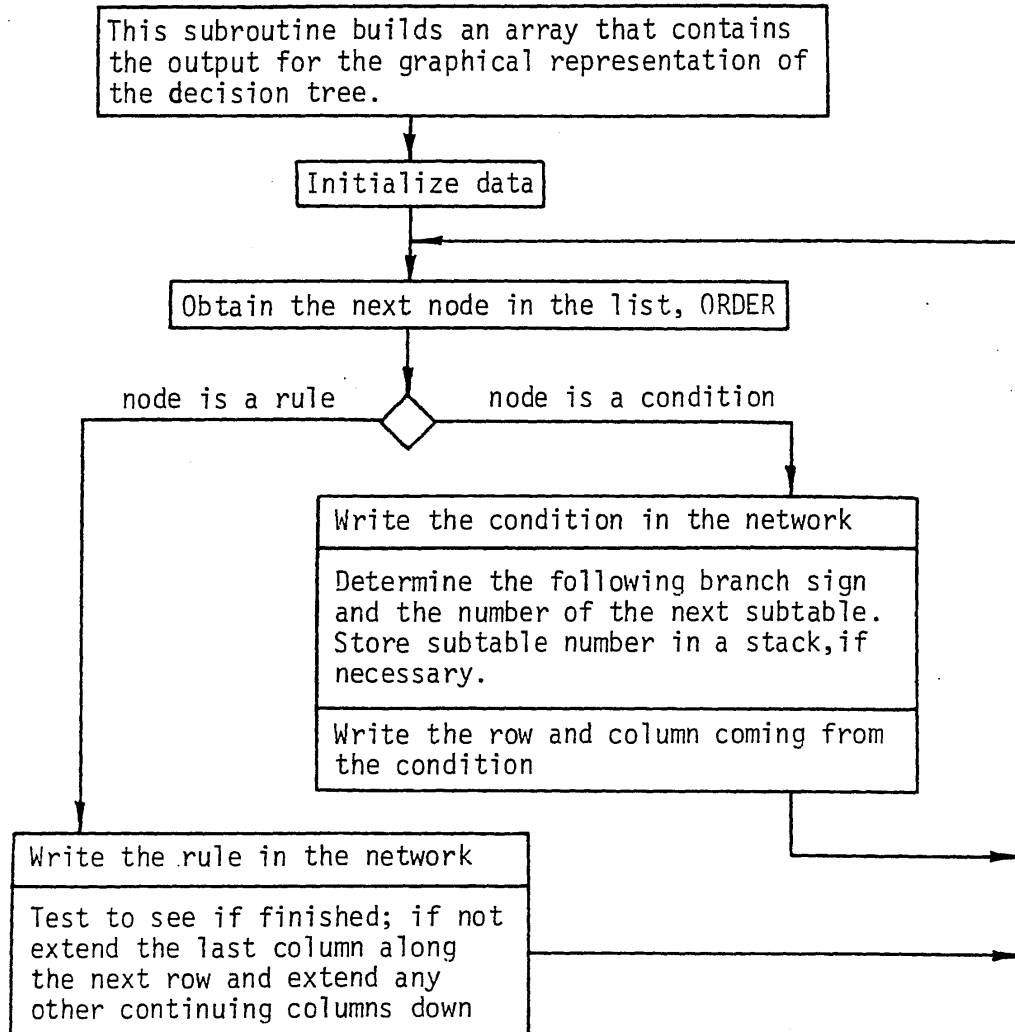


Figure 1.10 Subroutine ARRAY (TABLE)

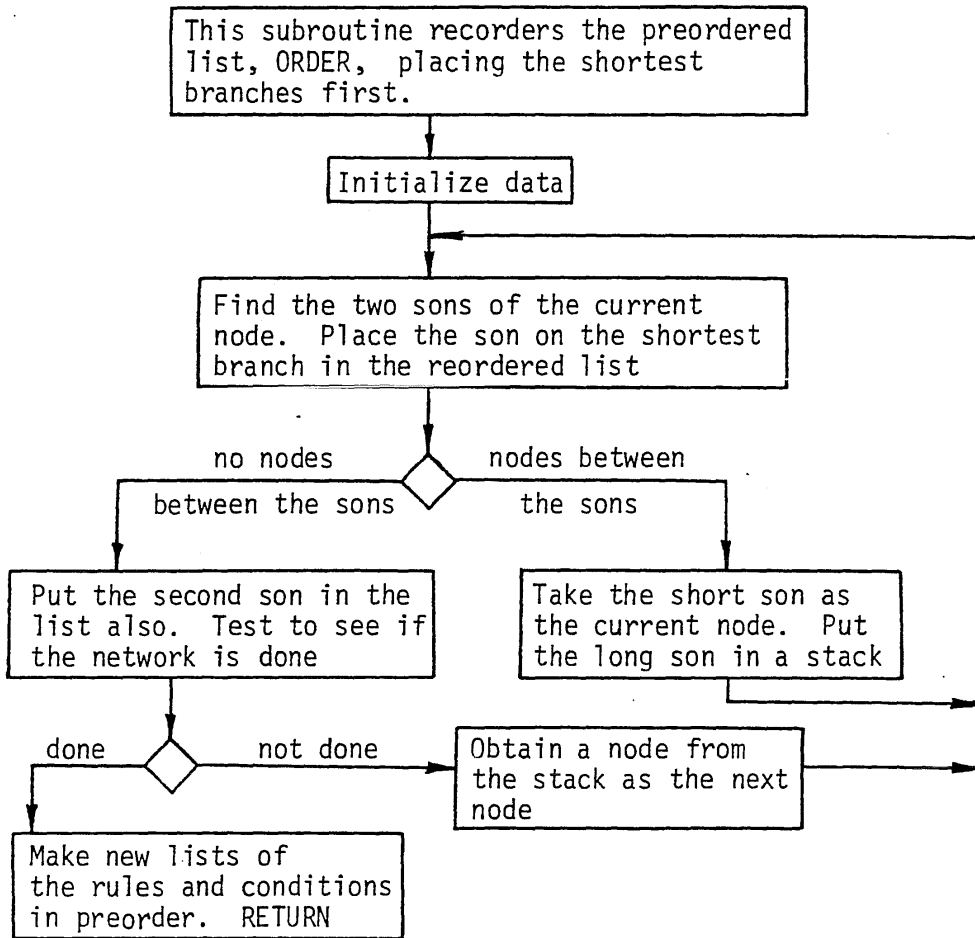


Figure 1.11 Subroutine SORT (TABLE)

1.3 Data Structure

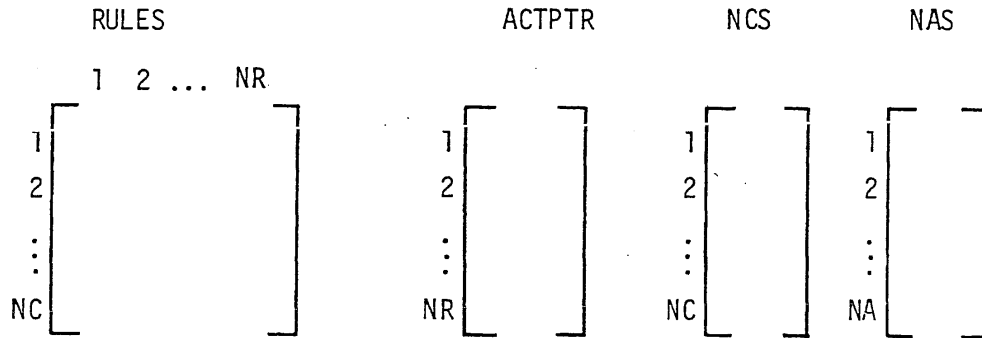
All of the manipulations in the program are based on operations with integer numbers and comparisons of logical variables. Consequently most of the data structure is composed of vectors and arrays of integers. The exceptions to this are the arrays of condition and action stubs that are stored for use in the output.

The data that is permanently stored is described in figures 1.12 and 1.13. Other important data is described in figure 1.14 and 1.15. A complete list of all the data items used in the program is in the glossary, section 1.4.

INTEGERS:

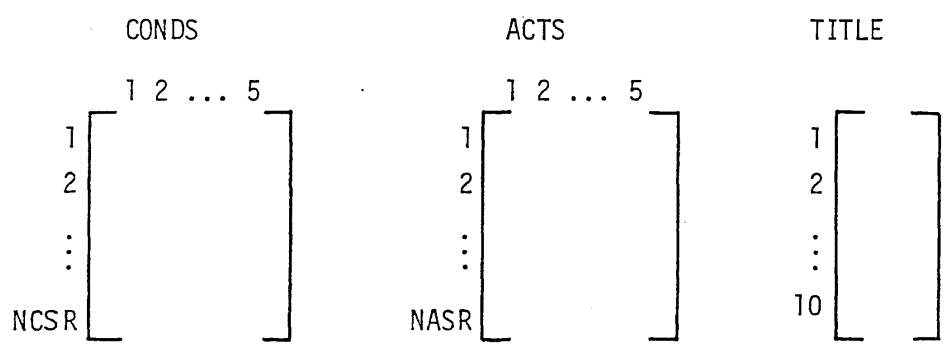
- NR - number of rules
- NC - number of conditions
- NA - number of actions
- NCSR - number of condition stub rows
- NASR - number of action stub rows

INTEGER VECTORS AND ARRAYS:



- RULES Contains the condition entry. A true is stored as a positive one, a false is a negative one, an immaterial is a zero, and so on.
- ACTPTR Contains the action number for each rule.
- NCS Contains the number of rows for each condition stub
- NAS Contains the number of rows for each action stub

ALPHAMERIC DATA:



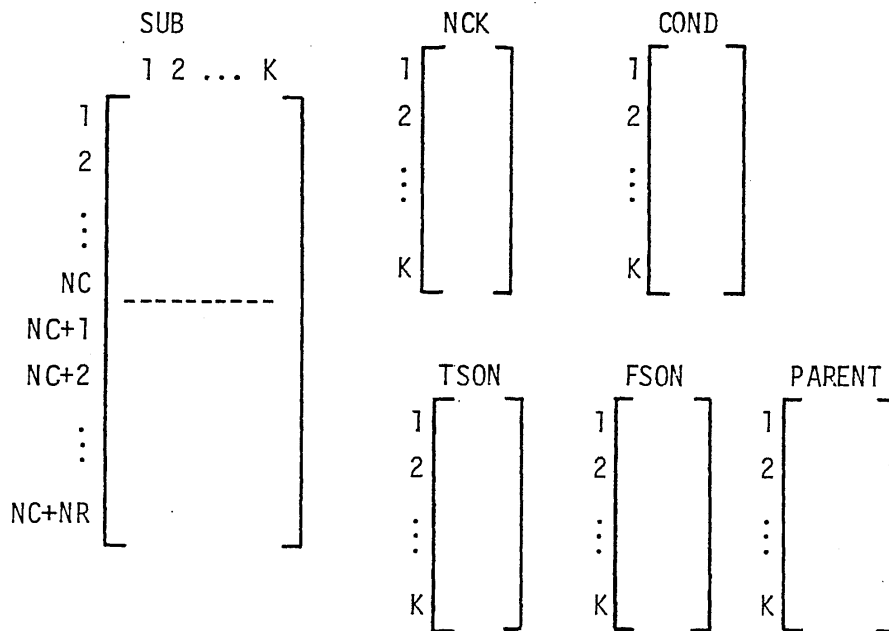
- CONDS Contains the condition stubs. Each word is six characters, so each row is 30 characters.
- ACTS Contains the action stubs
- TITLE Contains the title. Space is available for 60 characters

Figure 1.12 Permanent Data (TABLE)

Contents of the line	Format
Title	10A6
NR NC NA NCSR NASR	5I5
RULES, row 1, 1 to NR	27I3
RULES, row 2, 1 to NR	
⋮	
RULES, row NC, 1 to NR	
ACTPTR, 1 to NR	27I3
CONDS, row 1	5A6
CONDS, row 2	
⋮	
CONDS, row NCSR	
ACTS, row 1	5A6
ACTS, row 2	
⋮	
ACTS, row NASR	
NCS, 1 to NC	27I3
NAS, 1 to NA	27I3

The figure shows the manner that the permanent data are stored on the disk

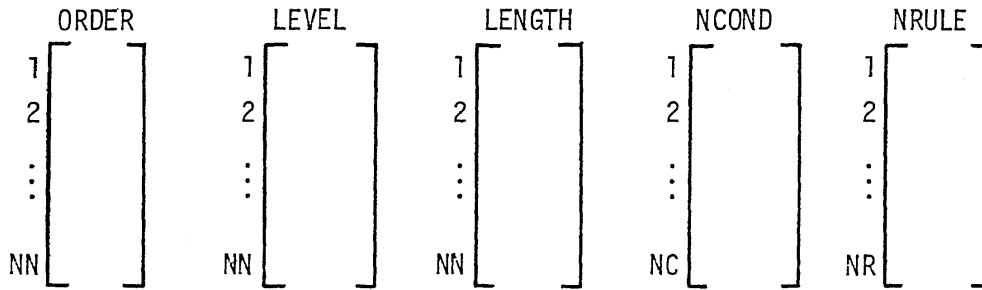
Figure 1.13 Permanent Data File (TABLE)



The above arrays are used to decompose the table into a network. The subscript of the five vectors at the right and the column index (second subscript) of SUB indicates the subtable number. The first index in SUB indicates the condition or rule number. The contents are as follows:

- SUB Contains the subtable data. A one is recorded if the condition or rule is in subtable K, a zero if it is not.
- NCK Contains the number of conditions in subtable K.
- COND Contains the condition selected for testing from subtable K.
- TSON Contains the number of the subtable (or rule) derived from the true portion of the subtable, (a negative number indicates a rule.)
- FSON Similar to TSON, but for the false portion.
- PARENT Contains the number of the subtable from which subtable K was derived. A negative sign indicates that K is the false son.

Figure 1.14 Data Used in Deomposition (TABLE)



- NN is the number of nodes in the decision network
- ORDER is a list of the nodes in the decision network arranged in "preorder" as defined by Knuth (ref. 1.4) A positive number indicates a condition, a negative number a rule, and a zero the else rule.
- LEVEL contains the number of steps away from the beginning node for each node in the network
- LENGTH contains the number of steps away from the rule along the longest branch emanating from the node for each node.
- NCOND contains a list of the conditions arranged in the order that they appear in ORDER. Nodes that appear more than once in the network are not repeated in NCOND.
- NRULE contains a list of the rules similar to NCOND.

Figure 1.15 Data Used in Describing the Network (TABLE)

100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
2600
2700
2800
2900
3000
3100
3200
3300
3400
3500
3600
3700
3800
3900
4000
4100
4200
4300
4400
4500
4600
4700
4800
4900
5000
5100
5200
5300
5400
5500
5600
5700
5800
5900

1.4 GLOSSARY

SYMBOL	:	DESCRIPTION
A, AA, OR AAA	:	DUMMY NAME FOR AN ARGUMENT
ACTION	:	VECTOR CONTAINING THE ACTION ENTRY FOR ONE ACTION AT A TIME
ACTPTR	:	VECTOR CONTAINING THE ACTION NUMBER FOR EACH RULE
ACTS	:	ARRAY CONTAINING THE ACTION STUBS
ASTER	:	THE SYMBOL *
ASTUR	:	ARRAY CONTAINING THE ACTION STUBS
B	:	DUMMY NAME FOR AN ARGUMENT
BKSTK	:	VECTOR USED IN TOPD TO STORE THE NODES IN THE NETWORK WITH ONE SON NOT YET ENTERED IN ORDER
BLANK	:	THE BLANK SYMBOL (SIX CHARACTERS)
BRANCH	:	LOGICAL VARIABLE GIVING THE SIGN OF THE NEXT BRANCH TO NEXT
C	:	THE LETTER C
CK	:	VECTOR OF CONDITIONS IN SUBTABLE K
COMPAC	:	A SUBROUTINE
COND	:	VECTOR CONTAINING THE CONDITION TESTED IN EACH SUBTABLE
CONDS	:	ARRAY OF CONDITION STUBS
CONIT	:	A SUBROUTINE
CSTUR	:	ARRAY OF CONDITION STUBS
CVR	:	VECTOR USED TO INDICATE WHAT BRANCHES ARE UNFINISHED IN THE NETWORK
DC	:	VECTOR CONTAINING THE ABSOLUTE DIFFERENCE BETWEEN TRUE ENTRIES AND FALSE ENTRIES FOR THE FLGIBLE CONDITIONS
DELTA	:	A SUBROUTINE
DIVIDE	:	VECTOR CONTAINING ASTERISK SYMBOLS

6000		:	
6100	DR	:	VECTOR CONTAINING THOSE CONDITIONS WITH MAXIMUM
6200		:	(OR MINIMUM) DC
6300		:	
6400	F	:	THE LETTER E
6500		:	
6600	FC	:	VECTOR CONTAINING THE NUMBER OF EXPLICIT ENTRIES FOR THE
6700		:	ELIGIBLE CONDITIONS
6800		:	
6900	FL	:	THE LETTER L
7000		:	
7100	FND	:	LOGICAL FUNCTION IN PARSE
7200		:	
7300	ENTITY	:	VECTOR CONTAINING AN ITEM OF INPUT FROM PARSE
7400		:	
7500	FR	:	VECTOR CONTAINING THOSE CONDITIONS WITH MAXIMUM EC
7600		:	
7700	ERROR	:	A SUBROUTINE
7800		:	
7900	EXPLIT	:	A SUBROUTINE
8000		:	
8100	FIXED	:	LOGICAL FUNCTION IN PARSE
8200		:	
8300	FLOUT	:	THE FILE NUMBER FOR OUTPUT
8400		:	
8500	FRULES	:	VECTOR CONTAINING THE RULES IN A SUBTABLE WITH FALSE OR
8600		:	IMMATERIAL ENTRIES FOR THE SELECTED CONDITION
8700		:	
8800	FSON	:	VECTOR CONTAINING THE SUBTABLE OR RULE THAT IS ON THE FALSE
8900		:	BRANCH FROM ANY SUBTABLE
9000		:	
9100	IC	:	VECTOR CONTAINING THE NUMBER OF IMMATERIAL ENTRIES FOR EACH
9200		:	CONDITION IN A SUBTABLE
9300		:	
9400	IMAX	:	THE NUMBER OF COLUMNS USED FOR THE OUTPUT OF THE NETWORK
9500		:	
9600	INIT	:	A SUBROUTINE
9700		:	
9800	INPUT	:	A SUBROUTINE
9900		:	
10000	IVALUE	:	INTEGER VALUE FROM PARSE
10100		:	
10200	JMAX	:	THE NUMBER OF LINES USED FOR THE OUTPUT OF THE NETWORK
10300		:	
10400	K	:	THE NUMBER OF THE SUBTABLE
10500		:	
10600	KK	:	THE NUMBER OF THE NEXT SUBTABLE
10700		:	
10800	LENGTH	:	VECTOR CONTAINING THE NUMBER OF STEPS TO THE RULE ALONG THE
10900		:	LONGEST BRANCH FROM THE NODE FOR EACH NODE IN THE NETWORK
11000		:	
11100	LEVEL	:	VECTOR CONTAINING THE NUMBER OF STEPS FROM THE BEGINNING TO
11200		:	THE NODE FOR EACH NODE IN THE NETWORK
11300		:	
11400	LOGIC	:	DUMMY NAME FOR AN ARGUMENT
11500		:	
11600	LVR	:	VECTOR CONTAINING THE COLUMN NUMBER OF EACH INCOMPLETE
11700		:	BRANCH IN THE NETWORK
11800		:	
11900	MATCH	:	LOGICAL FUNCTION IN PARSE
12000		:	
12100	MATER	:	A SUBROUTINE

12200	:	:
12300	MTNUS	: THE NEGATIVE SYMBOL
12400	:	:
12500	MODE	: VALUE FROM PARSE INDICATING THE TYPE OF INPUT ITEM
12600	:	:
12700	MR	: VECTOR CONTAINING THE CONDITIONS WITH THE MINIMUM IC
1 00	:	:
12900	N	: NC + NF
13000	:	:
13100	NA	: NUMBER OF ACTIONS
13200	:	:
13300	NAS	: VECTOR CONTAINING THE NUMBER OF STUBS FOR EACH ACTION
13400	:	:
13500	NASR	: TOTAL NUMBER OF ROWS OF ACTION STUBS
13600	:	:
13700	NC	: NUMBER OF CONDITIONS
13800	:	:
13900	NCHAR	: FROM PARSE: THE NUMBER OF CHARACTERS IN THE INPUT ITEM
14000	:	:
14100	NCK	: VECTOR CONTAINING THE NUMBER OF CONDITIONS IN THE SUBTABLE
14200	:	:
14300	NCOND	: VECTOR CONTAINING A PREORDER LIST OF THE CONDITIONS IN
14400	:	: THE NETWORK, DELETING REPETITIOUS ENTRIES
14500	:	:
14600	NCS	: VECTOR CONTAINING THE NUMBER OF STUBS FOR EACH CONDITION
14700	:	:
14800	NCSR	: TOTAL NUMBER OF ROWS OF CONDITION STUBS
14900	:	:
15000	NDR	: NUMBER OF CONDITIONS WITH MAXIMUM (OR MINIMUM) DC
15100	:	:
15200	NFF	: NUMBER OF EXPLICIT ENTRIES LEFT IN THE RULE IN THE SUBTABLE
15 0	:	:
15400	NFR	: NUMBER OF CONDITIONS WITH MAXIMUM FC
15500	:	:
15600	NFT	: ARRAY FOR OUTPUT OF THE NETWORK
15700	:	:
15800	NFXT	: LOGICAL FUNCTION IN PARSE
15900	:	:
16000	NF	: NUMBER OF RULES WITH FALSE ENTRIES IN THE SUBTABLE FOR
16100	:	: THE SELECTED CONDITION
16200	:	:
16300	NMF	: NUMBER OF CONDITIONS WITH MINIMUM IC
16400	:	:
16500	NN	: NUMBER OF NODES IN THE NETWORK
16600	:	:
16700	NR	: NUMBER OF RULES
16800	:	:
16900	NRULES	: VECTOR CONTAINING PREORDER LIST OF THE RULES IN THE
17000	:	: NETWORK, DELETING REPETITIOUS ENTRIES
17100	:	:
17200	NT	: NUMBER OF RULES WITH TRUE OR IMMATERIAL ENTRIES IN THE
17300	:	: SUBTABLE FOR THE SELECTED CONDITION
17400	:	:
17500	NWD	: FROM PARSE: NUMBER OF WORDS IN THE INPUT ITEM
17600	:	:
17700	DLDATA	: A SUBROUTINE
17800	:	:
17 0	ONPACK	: LOGICAL VARIABLE FOR TESTING THE PRESENCE OF A DATA FILE
18000	:	:
18100	ORDER	: VECTOR CONTAINING THE LIST OF NODES IN NETWORK IN PREORDER
18200	:	:
18300	OUT1	: A SUBROUTINE

18400		:	
18500	OUT2	:	A SUBROUTINE
18600		:	
18700	PARCOM	:	FILE CONTAINING COMMON DECLARATIONS FOR PARSE
18800		:	
18900	PARENT	:	VECTOR CONTAINING THE NUMBER OF THE SUBTABLE FROM WHICH
1 00		:	THE SUBTABLE IS DERIVED
19100		:	
19200	PLUS	:	THE ADDITION SYMBOL
19300		:	
19400	POINT	:	VARIABLE CARRYING THE SYMBOLS PLUS OR MINUS
19500		:	
19600	PRT	:	CARRIES THE LETTER OFF
19700		:	
19800	PTRBK	:	VECTOR USED AS A POINTER BACK TO THE PARENT NODE IN TABLE
19900		:	
20000	QUITCK	:	LOGICAL VARIABLE THAT IS TRUE UNLESS THE NETWORK IS BEING
20100		:	FORMED WITH THE DELAYED RULE
20200	P	:	THE LETTER R
20300		:	
20400	RDLINE	:	SUBROUTINE IN PARSE
20500		:	
20600	ROW	:	A SUBROUTINE
20700		:	
20800	RULES	:	ARRAY CONTAINING THE CONDITION ENTRY
20900		:	
21000	S	:	THE LETTER S
21100		:	
21200	SFTOUT	:	SUBROUTINE IN PARSE
21300		:	
21400	SIGN	:	VECTOR CONTAINING THE SYMBOLS USED IN PRINTING THE
21 00		:	CONDITION ENTRY
21600		:	
21700	SKC	:	VECTOR CONTAINING A STACK OF SUBTABLE NUMBERS WITH
21800		:	UNUSED SONS
21900		:	
22000	SOFT	:	A SUBROUTINE
22100		:	
22200	STRING	:	FROM PARSE: ANY INPUT ITEM ENCLOSED IN QUOTES
22300		:	
22400	SUB	:	THE ARRAY CONTAINING THE DATA FOR INCLUSION OF CONDITIONS
22500		:	AND RULES IN THE SUBTABLES
22600		:	
22700	SUBTAB	:	A SUBROUTINE
22800		:	
22900	SWITCH	:	LOGICAL VARIABLE FOR THE SUCCESS OF THE DECOMPOSITION
23000		:	
23100	SYMBOL	:	VECTOR CONTAINING THE SYMBOLS FOR THE TEN NUMERALS
23200		:	
23300	TARCOM	:	FILE CONTAINING THE COMMON DECLARATIONS
23400		:	
23500	TARTRF	:	A SUBROUTINE
23600		:	
23700	TARTWO	:	A LOGICAL VARIABLE FOR USE IN PRINTING REORDERED VERSION
23800		:	OF THE TABLE
23900		:	
24000	TITLE	:	THE TITLE OF THE TABLE
24 0		:	
24200	TOPD	:	A SUBROUTINE
24300		:	
24400	TRULES	:	VECTOR CONTAINING THOSE RULES WITH TRUE OR IMMATERIAL
24500		:	ENTRIES IN THE SUBTABLE FOR THE SELECTED CONDITION

24600		:	
24700	TSDN	:	VECTOR CONTAINING THE SUBTABLE OR PULF THAT IS ON THE TRUE
24800		:	BRANCH FROM THE SUBTABLE
24900		:	
25000	VALUE	:	REAL NUMBER FROM PARSE
25100		:	
25200	WDINIT	:	SUBROUTINE IN PARSE
25300		:	
25400	YX	:	THE LETTER X
25500		:	
25600	Y, Z	:	DUMMY NAMES FOR ARGUMENTS
25700		:	
25800	ZFE0	:	A SUBROUTINE

10
 20
 30
 40
 50
 60
 70
 80
 90
 100
 200
 300
 400
 500
 600
 700
 800
 900
 1000
 1100
 1200
 1300
 1400
 1500
 1600
 1700
 1800
 1900
 2000
 2100
 2200
 2300
 2400
 2500
 2600
 2700
 2800
 2900
 3000
 3100
 3200
 3300
 3400
 3500
 3600
 3700
 3800
 3900
 4000
 4100
 4200
 4300
 4400
 4500
 4600
 4700
 4800
 4900
 5000

1.5 PROGRAM LISTING

```

$ RESET FREE
FILE 5=INCOM,UNIT=REMOTE,RECORD=14
FILE 6=OUTCOM,UNIT=REMOTE,RECORD=14
FILE 1=OUTPUT,UNIT=PRINTER,RECORD=23
FILE 2=TREELIST,UNIT=DISKPACK,RECORD=23,BLOCKING=30
FILE 3=DUMMY,UNIT=DISKPACK,RECORD=14,BLOCKING=30,SAVE=1
C
$ INCLUDE 'PARSE'
C
$ INCLUDE 'TABCOM'
C
C THE FOLLOWING LINES ARE A LISTING OF TABCOM
C
C THIS FILE DIMENSIONS ALL THE ARRAYS USED IN THE PROGRAM
C AND ASSIGNS THEM TO COMMON.
C
C INTEGER RULES(30,30), ACTPTR(30), SUB(60,60), COND(60),
C 1 TSUN(60), FSUN(60), PARENT(60), NCK(60), TRULES(30),
C 2 FRULES(30), IC(30), MR(30), EC(30), ER(30), DC(30),
C 3 DR(30), ORDER(60), LEVEL(60), LENGTH(60), LVB(15), FLJUT,
C 4 NRULE(30), NCOND(30), CVB(15), PTRBK(60), BKSTK(60), CK(30),
C 5 NCS(30), NAS(30)
C
C REAL TITLE(10), CONDS(5,40), ACTS(5,40), SIGN(5),
C 1 SYMBOL(10), NET(132,120), DIVIDE(132), ACTION(30), MINUS
C
C LOGICAL SWITCH, QUICK, BRANCH, UNPACK, TABTWO
C
C COMMON/INT/DF, NR, NC, NA, RULES, ACTPTR, SUB, K, KK, N,
C 1 COND, TSUN, FSUN, PARENT, NCK, TRULES, NT, FRULES, NF,
C 2 IC, MR, NMR, EC, ER, NER, DC, DR, NDR, NEE, ORDER,
C 3 LEVEL, LENGTH, NN, JMAX, LVB, NRULE, NCOND, CVB, PTRBK,
C 4 BKSTK, CK, FLJUT, IMAX, NCS, NAS, NCSR, NASR
C
C COMMON/REAL/TITLE, CONDS, ACTS, SIGN, SYMBOL, NET, DIVIDE,
C 1 ACTION, BLTRK, ASTER, XX, C, E, R, PLUS, MINUS, EL, S
C
C COMMON/LOG/SWITCH, QUICK, BRANCH, TABTWO
C
C END OF TABCOM
C
$ INCLUDE 'PARCOM'
C
C THE FOLLOWING LINES ARE A LISTING OF PARCOM
C
C THIS FILE LISTS THE DECLARATIONS FOR USE OF THE INPUT SCANNER
C

```

```
5100 C LOGICAL END, FIXED, MATCH, STRING, NEXT
5200 C COMMON/SCANNER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
5300 C EQUIVALENCE (IVALUE,VALUE)
5400 CC
5500 CC END OF PARCOM
5600 C
5700 C DELETE THE ECHO PRINT OF THE SCANNER
5800 C DATA PRI/"OFF"/
5900 C
6000 C SET THE LENGTH OF A LINE OF INPUT EQUAL TO 72 SPACES
6100 C AND THE NUMBER OF BLANKS REQUIRED TO SIGNIFY THE END
6200 C OF A RECORD EQUAL TO 10
6300 C CALL WDINIT(10,72,PRI)
6400 C
6500 C READ THE OUTPUT UNIT IN
6600 C FLOUT = 6
6700 C WRITE(FLOUT,1005)
6800 C CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
6900 C IF(.NOT.MATCH("P",1)) GO TO 10
7000 C FLOUT = 1
7100 C
7200 C CHECK TO SEE IF THE ECHO IS DESIRED
7300 C
7400 C WRITE(6,1200)
7500 C CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
7600 C IF(MATCH("YES",3)) CALL WDINIT(10,72,"UN ")
7700 C CALL SETOUT(FLOUT)
7800 C
7900 C INITIALIZE THE ARRAYS
8000 C 10 CALL INIT
8100 C
8200 C READ THE DATA FILE NAME IN
8300 C 20 I = 0
8400 C WRITE(6,1000)
8500 C CALL RDLINE
8600 C ENTITY(NWD+1)=6H.
8700 C CLOSE (3)
8800 C CHANGE(3,TITLE=ENTITY)
8900 C DF=3
9000 C
9100 C CHECK FOR EXISTING DATAFILE
9200 C INQUIRE(5, RESIDENT=ONPACK)
9300 C IF(.NOT. UNPACK) GO TO 50
9400 C
9500 C EXISTING FILE, DOUBLE CHECK THE NAME
9600 C WRITE(6,1010)
9700 C CALL RDLINE
9800 C IF(MATCH("YES",3)) GO TO 30
9900 C GO TO 20
10000 C
10100 C USE EXISTING FILE
10200 C 30 CALL OLDATA
10300 C GO TO 60
10400 C
10500 C READ NEW NETWORK DATA IN FROM THE TERMINAL
10600 C 50 CALL INPUT
10700 C GO TO 70
10800 C
10900 C CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
11000 C 60 WRITE(6,1020)
11100 C CALL RDLINE
11200 C IF(.NOT.MATCH("YES",3)) GO TO 70
```

```
11300      CALL INPUT
11400      C
11500      C      WRITE OUT THE ORIGINAL TABLE
11600      70 IF(FLOUT .EQ. 6) WRITE(6,999)
11700      TABTWO = .FALSE.
11800      CALL OUT1
11900      C
12000      C      DECOMPOSE THE DECISION TABLE INTO THE NETWORK
12100      75 QUICK = .TRUE.
12200      CALL TABTRE
12300      IF(.NOT.SWITCH) GO TO 110
12400      IF(FLOUT.EQ.6) WRITE(6,999)
12500      WRITE(6,1040)
12600      C
12700      C      DETERMINE THE TOPOLOGY OF THE DERIVED NETWORK
12800      CALL TOPU
12900      C
13000      C      PRINT OUT THE DATA
13100      IF(FLOUT .EQ. 6) WRITE(6,999)
13200      WRITE(FLOUT,1050)
13300      CALL OUT2
13400      IF(FLOUT .EQ. 6) WRITE(6,999)
13500      C
13600      C      BEGIN SCANNING FOR OPERATING COMMANDS
13700      C
13800      I = 0
13900      100 WRITE(6,1030)
14000      CALL RDLIN
14100      110 IF(END(X)) GO TO 100
14200      IF(MATCH("WRITE",3)) GO TO 140
14300      IF(MATCH("MODIFY",3)) GO TO 150
14400      IF(MATCH("SORT",3)) GO TO 160
14500      IF(MATCH("DELAY",3)) GO TO 170
14600      IF(MATCH("NEXT",3)) GO TO 10
14700      IF(MATCH("STOP",3)) STOP
14800      I = I + 1
14900      CALL ERRUR(I)
15000      GO TO 100
15100      C
15200      C      WRITE OUT THE REARRANGED TABLE
15300      C
15400      140 WRITE(FLOUT,1060)
15500      TABTWO = .TRUE.
15600      CALL OUT1
15700      IF(FLOUT .EQ. 6) WRITE(6,999)
15800      GO TO 100
15900      C
16000      C      MODIFY THE BASIC INPUT DATA
16100      C
16200      150 CALL INI1
16300      CALL OLDATA
16400      CALL INPUT
16500      I = 0
16600      GO TO 70
16700      C
16800      C      SORT THE BRANCHES BY LENGTH
16900      160 CALL SORT
17000      IF(FLOUT .EQ. 6) WRITE(6,999)
17100      WRITE(FLOUT,1070)
17200      CALL OUT2
17300      IF(FLOUT .EQ. 6) WRITE(6,999)
17400      I = 0
```

```

17500      GO TO 100
17600      C
17700      C      DECOMPOSE THE NETWORK USING THE DELAYED DECISION RULE
17800      170 QUICK = .FALSE.
17900      CALL INIT
18000      CALL NLDATA
18100      CALL TABIRE
18200      IF(.NOT.SWITCH) GO TO 110
18300      WRITE(6,1040)
18400      CALL TGPU
18500      IF(FLOUT .EQ. 6) WRITE(6,999)
18600      WRITE(FLOUT,1090)
18700      CALL OUT2
18800      IF(FLOUT .EQ. 6) WRITE(6,999)
18900      T = 0
19000      GO TO 100
19100      C
19200      C
19300      999 FORMAT(1H ,/,1H ,/)
19400      1000 FORMAT(1H , "ENTER THE DATA FILE NAME",/)
19500      1005 FORMAT(1H , "ENTER P FOR OUTPUT ON THE ON-SITE PRINTER,",
19600      1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL",/)
19700      1010 FORMAT(1H , "DATA FILE EXISTS WITH THIS NAME.",/,
19800      1 " DO YOU WANT TO USE IT?",/)
19900      1020 FORMAT(1H , "DO YOU WANT TO MODIFY THE EXISTING DATA?",/)
20000      1030 FORMAT(1H , "ENTER A PROGRAM CUMMAND",/)
20100      1040 FORMAT(1H , "DECISION NETWORK SUCCESSFULLY COMPLETED",/)
20200      1050 FORMAT(1H1, "DERIVED DECISION NETWORK",//)
20300      1060 FORMAT(1H1, "DECISION TABLE WITH CONDITIONS AND RULES IN PREORDER"
20400      1070 FORMAT(1H1, "DECISION NETWORK SORTED BY BRANCH LENGTH",//)
20500      1080 FORMAT(1H1, "DECISION TABLE WITH CONDITIONS AND ",/,
20600      1 " RULES IN THE PREORDER OF THE SORTED NETWORK")
20700      1090 FORMAT(1H1, "DECISION NETWORK DERIVED WITH THE ",
20800      1 " DELAYED DECISION RULE",//)
20900      1100 FORMAT(1H1, "DECISION TABLE WITH CONDITIONS AND ",/,
21000      1 " RULES IN THE PREORDER OF THE DELAYED NETWORK")
21100      1200 FORMAT(" DO YOU WANT THE PRINTOUT OF THE INPUT?",/)
21200      END
21300      C
21400      C -----
21500      C
21600      SUBROUTINE INIT
21700      S INCLUDE 'TABCOM'
21800      DU 10 I = 1, 60
21900      DU 10 J = 1, 60
22000      10 SUR(J,I) = 0
22100      DU 20 I = 1, 60
22200      20 CONG(I) = 0
22300      DU 30 I = 1, 60
22400      30 TSON(I) = 0
22500      DU 40 I = 1, 60
22600      40 FSON(I) = 0
22700      DU 50 I = 1, 60
22800      50 PARENT(I) = 0
22900      DU 60 I = 1, 60
23000      60 MCK(I) = 0
23100      DU 70 I = 1, 60
23200      70 ORDER(I) = 0
23300      DU 80 I = 1, 60
23400      80 LEVEL(I) = 0
23500      DU 90 I = 1, 60
23600      90 LENGTH(I) = 0

```

```
23700      DO 100 I = 1, 30
23800      100 NCS(I) = 1
23900      DO 110 I = 1, 30
24000      110 NAS(I) = 1
24100      DO 120 I = 1, 40
24200      DO 120 J = 1, 5
24300      120 ACTS(J,I) = BLANK
24400      DO 130 I = 1, 40
24500      DO 130 J = 1, 5
24600      130 CUNOS(J,I) = BLANK
24700      DO 140 I = 1, 132
24800      140 DIVIDE(I) = ASTER
24900      DO 150 I = 1, 10
25000      150 TITLE(I) = BLANK
25100      RETURN
25200      END
25300      C
25400      C -----
25500      C
25600      C      SUBROUTINE INPUT
25700      C
25800      C      $ INCLUDE 'TABCOM'
25900      C
26000      C
26100      C      $ INCLUDE 'PAHCOM'
26200      C
26300      C
26400      C      CSTOR AND ASTOR ARE USED TO STORE CHANGE IN THE STUBS
26500      C      REAL CSTOR(50,30), ASTOR(50,30)
26600      C      DO 3 I = 1, 30
26700      C      DO 2 J = 1, 50
26800      C      2 CSTOR(J,I) = BLANK
26900      C      DO 4 J = 1, 30
27000      C      4 ASTOR(J,I) = BLANK
27100      C      6 CONTINUE
27200      C      WRITE(6,1000)
27300      C      I = 0
27400      C
27500      C      BEGIN SCANNING FOR INPUT
27600      C      CALL PARSE(ENTITY, MODE, VALUE, NCHAR)
27700      C      10 IF(MATCH("NUMBER",3)) GO TO 20
27800      C      IF(MATCH("RULES",3)) GO TO 70
27900      C      IF(MATCH("CONDIT",3)) GO TO 160
28000      C      IF(MATCH("ACTION",3)) GO TO 200
28100      C      IF(MATCH("TITLE",3)) GO TO 240
28200      C      IF(MATCH("END",3)) GO TO 270
28300      C      IF(END(X)) GO TO 10
28400      C      I = I + 1
28500      C      CALL ERRUR(I)
28600      C      CALL ROLINE
28700      C      GO TO 10
28800      C
28900      C      HERE WE HAVE THE NUMBER OF RULES, CONDITIONS, OR ACTIONS
29000      C      20 IF(MATCH("OF",2)) GO TO 30
29100      C      30 IF(MATCH("RULES",3)) GO TO 40
29200      C      IF(MATCH("CONDIT",3)) GO TO 50
29300      C      IF(MATCH("ACTION",3)) GO TO 60
29400      C      35 I = J + 1
29500      C      CALL ERRUR(I)
29600      C      CALL ROLINE
29700      C      GO TO 10
29800      C      40 IF(.NOT.(FIXED(X))) GO TO 35
```

```
29900      NR = IVALUE
30000      I = 0
30100      GO TO 10
30200      50 IF(.NOT.(FIXED(X))) GO TO 35
30300      NC = IVALUE
30400      I = 0
30500      GO TO 10
30600      60 IF(.NOT.(FIXED(X))) GO TO 35
30700      NA = IVALUE
30800      I = 0
30900      GO TO 10
31000      C
31100      C  HERE WE HAVE THE RULES ARRAY
31200      70 WRITE(6,1010)
31300      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
31400      80 IF(MATCH("LAST",3)) GO TO 10
31500      IF(FIXED(X)) GO TO 35
31600      I = I + 1
31700      CALL ERRUR(I)
31800      CALL ROLINE
31900      GO TO 80
32000      85 I = IVALUE
32100      J = 0
32200      90 J = J + 1
32300      IF(MATCH("T",1) .OR. MATCH("Y",1)) GO TO 100
32400      IF(MATCH("F",1) .OR. MATCH("N",1)) GO TO 110
32500      IF(MATCH("+",1)) GO TO 120
32600      IF(MATCH("-",1)) GO TO 130
32700      IF(MATCH("I",1) .OR. MATCH(".",1)) GO TO 140
32800      IF(FIXED(X)) GO TO 150
32900      IF(END(X)) GO TO 80
33000      I = I + 1
33100      CALL ERRUR(I)
33200      CALL ROLINE
33300      GO TO 80
33400      100 RULES(J,1) = 1
33500      GO TO 90
33600      110 RULES(J,1) = -1
33700      GO TO 90
33800      120 RULES(J,1) = 2
33900      GO TO 90
34000      130 RULES(J,1) = -2
34100      GO TO 90
34200      140 RULES(J,1) = 0
34300      GO TO 90
34400      150 ACTPTR(I) = IVALUE
34500      CALL ROLINE
34600      I = 0
34700      GO TO 80
34800      C
34900      C  HERE WE HAVE THE CONDITION STRINGS
35000      160 WRITE(6,1020)
35100      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
35200      170 IF(MATCH("LAST",3)) GO TO 195
35300      IF(FIXED(X)) GO TO 175
35400      IF(END(X)) GO TO 170
35500      172 I = I + 1
35600      CALL ERRUR(I)
35700      CALL ROLINE
35800      GO TO 170
35900      175 I = IVALUE
36000      NCS(I) = 0
```



```
36100          CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
36200      180 IF(MODE.NE.7) GO TO 170
36300          NCS(I) = NCS(I) + 1
36400          IF(NWD .GT. 5) NWD = 5
36500          DO 190 J = 1, NWD
36600          IJ = J + 5*(NCS(I)-1)
36700      190 CSTUB(IJ,I) = ENTITY(J)
36800          CALL RDLIN
36900          GO TO 180
37000      195 I = 0
37100          GO TO 10
37200      C
37300      C   HERE WE HAVE THE ACTION STRINGS
37400      200 WRITE(6,1030)
37500          CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
37600      210 IF(MATCH("LAST",3)) GO TO 10
37700          IF(FIXED(X)) GO TO 215
37800          IF(END(X)) GO TO 210
37900      212 I = I + 1
38000          CALL ERRUR(I)
38100          CALL RDLIN
38200          GO TO 210
38300      215 I = IVALUE
38400          NAS(I) = 0
38500          CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
38600      220 IF(MODE.NE.7) GO TO 210
38700          NAS(I) = NAS(I) + 1
38800          IF(NWD .GT. 5) NWD = 5
38900          DO 230 J = 1, NWD
39000          IJ = J + 5*(NAS(I) - 1)
39100      230 ASTUB(IJ,I) = ENTITY(J)
39200          CALL RDLIN
39300          GO TO 220
39400      235 I = 0
39500          GO TO 10
39600      C
39700      C   HERE WE HAVE THE TITLE
39800      240 IF(END(X)) CALL RDLIN
39900      250 IF(MODE.EQ.7) GO TO 255
40000          I = I + 1
40100          CALL ERRUR(I)
40200          CALL RDLIN
40300          GO TO 240
40400      255 DO 260 J = 1, NWD
40500      260 TITLE(J) = ENTITY(J)
40600          CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
40700          I = 0
40800          GO TO 10
40900      C
41000      C
41100      C
41200      C   COMPACT THE CONDITION AND ACTION STUB ARRAYS FOR STORAGE
41300      C
41400      270 I = 1
41500          DO 273 II = 1, NC
41600          JJ = NCS(II)
41700          DO 272 J = 1, JJ
41800          DO 271 IJ = 1, 5
41900          JI = IJ + 5*(J-1)
42000          IF(.NOT.(CSTUB(JI,II).EQ.BLANK)) CONDS(I,J,I) = CSTUB(JI,II)
42100      271 CONTINUE
42200      272 I = I + 1
```

```

42300      273 CONTINUE
42400          NCSR = I - 1
42500      C
42600          I = 1
42700          DO 276 I1 = 1, NA
42800              JJ = NAS(I1)
42900              DO 275 J = 1, JJ
43000                  DO 274 IJ = 1, 5
43100                      JI = IJ + 5*(J - 1)
43200                      IF(.NOT.(ASTUB(JI,IJ).EQ.BLANK)) ACTS(IJ,I) = ASTUB(JI,II)
43300      274 CONTINUE
43400      275 I = I + 1
43500      276 CONTINUE
43600          NASR = I - 1
43700      C
43800      C      WRITE THE DATA ON DISK
43900          REWIND DF
44000          WRITE(DF,1040) (TITLE(J),J=1,10)
44100          WRITE(DF,1050) NR, NC, NA, NCSR, NASR
44200          DO 280 I = 1, NC
44300      280 WRITE(DF,1060) (RULES(I,J),J = 1, NR)
44400          WRITE(DF,1060) (ACIPTR(J),J = 1, NR)
44500          DO 290 I = 1, NCSR
44600      290 WRITE(DF,1040) (CONDSC(J,I),J=1,5)
44700          DO 300 I = 1, NASR
44800      300 WRITE(DF,1040) (ACTS(J,I),J=1,5)
44900          WRITE(DF,1060) (NCS(I), I = 1, NC)
45000          WRITE(DF,1060) (NAS(I), I = 1, NA)
45100          END FILE DF
45200          LUCK DF
45300      C
45400      1000 FORMAT(1H,"BEGIN INPUT INSTRUCTIONS.",/)
45500      1   "   ENTER THE WORD END WHEN FINISHED.",/)
45600      1010 FORMAT(1H,"ENTER THE RULE NUMBER, THE CONDITION ENTRIES,",/
45700      1   " AND THE ACTION ENTRY.",/, " ONE RULE TO A LINE.",/
45800      2   "   ENTER THE WORD LAST WHEN FINISHED.",/)
45900      1020 FORMAT(1H,"ENTER THE CONDITION NUMBER AND THE STRINGS.",/,/
46000      1   " 30 CHARACTERS TO A LINE.",/
46100      2   "   ENTER THE WORD LAST WHEN FINISHED.",/)
46200      1030 FORMAT(1H,"ENTER THE ACTION NUMBER AND THE STRING ONE",/
46300      1   " ACTION TO A LINE.",/, " ENTER THE WORD LAST WHEN FINISHED.",/)
46400      1040 FORMAT(1H,"10A6)
46500      1050 FORMAT(1H,"5I0)
46600      1060 FORMAT(1H,"27I3)
46700          RETURN
46800          END
46900      C
47000      C      - - - - -
47100      C
47200          SUBROUTINE DFDATA
47300      S INCLUDE 'TABCOM'
47400      C
47500      C
47600          LOGICAL TEST
47700      C
47800      C      FILL THE ARRAYS FROM FILE DF
47900          REWIND DF
48000          READ(DF, 1000) (TITLE(I),I=1,10)
48100          READ(DF, 1010) NR, NC, NA, NCSR, NASR
48200          TEST = .TRUE.
48300          IF(NCSR .EQ. 0) TEST = .FALSE.
48400          IF(NCSR .EQ. 0) NCSR = NC

```

```

48500      IF(QASR .EQ. 0) NASR = NA
48600      DO 10 I = 1, NC
48700      10 READ(DF, 1020) (RULES(I,J),J=1, NR)
48800      READ(DF, 1020) (ACIPTR(I), I=1, NR)
48900      DO 20 I = 1, NCSR
49000      20 READ(DF, 1000) (CUNDS(J,I), J=1, 5)
49100      DO 30 I = 1, NASR
49200      30 READ(DF, 1000) (ACTS(J,I), J=1, 5)
49300      IF(.NOT. TEST) GO TO 40
49400      READ(DF, 1020) (NCS(I), I = 1, NC)
49500      READ(DF, 1020) (NAS(I), I = 1, NA)
49600      40 REWIND DF
49700      1000 FORMAT(1H, 10A6)
49800      1010 FORMAT(1H, 2I5)
49900      1020 FORMAT(1H, 2I3)
50000      RETURN
50100      END
50200      C
50300      C -----
50400      C
50500      SUBROUTINE TABTRE
50600      S INCLUDE 'TABCOM'
50700      C
50800      C   SET UP COUNTERS AND FIRST SUBTABLE
50900      C
51000      K = 1
51100      KK = 2
51200      N = NC + NR
51300      NCK(K) = NC
51400      KEY = 1
51500      IF(QUICK) KEY = -1
51600      DO 10 I = 1, N
51700      10 SUB(I,K) = 1
51800      C
51900      C   SELECT THE CONDITION TO BE TESTED FROM SUBTABLE K
52000      C
52100      C   COUNT THE IMMATERIAL ENTRIES IN EACH ROW
52200      20 CALL ZERU(IC, NC)
52300      CALL MATER
52400      AA = NCK(K)
52500      CALL ROW(IC, MR, NMR, 1, AA, CK)
52600      CUND(K) = MR(1)
52700      IF(NMR .EQ. 1) GO TO 30
52800      C
52900      C   COUNT THE EXPLICIT ENTRIES IN EACH ROW
53000      CALL ZERU(EC, NC)
53100      CALL EXPLIT
53200      CALL ROW(EC, ER, NER, -1, NMR, MR)
53300      CUND(K) = ER(1)
53400      IF(NER .EQ. 1) GO TO 30
53500      C
53600      C   COUNT THE DIFFERENCE BETWEEN TRUE AND FALSE ENTRIES
53700      CALL ZERU(DC, NC)
53800      CALL DELTA
53900      CALL ROW(DC, DR, NDR, KEY, NER, ER)
54000      CUND(K) = DR(1)
54100      C
54200      C   COUNT TRUE ENTRIES IN SELECTED CONDITION
54300      C
54400      30 NT = 0
54500      DO 40 I = 1, NR
54600      IF(SUB(NC+I,K) .EQ. 0) GO TO 40

```

```

54700      IF(RULES(COND(K),I) .LT. 0) GO TO 40
54800      NT = NT + 1
54900      TRULES(NT) = I
55000      40 CONTINUE
55100      IF(NT=1) 70, 60, 50
55200      C
55300      C   HERE WE HAVE MULTIPLE RULES. CHECK FOR REDUNDANCY
55400      50 IF(CNCK(K).GT.1) GO TO 80
55500      WRITE(6,1000) (TRULES(I),I=1,NT)
55600      WRITE(6,1010)
55700      SWITCH = .FALSE.
55800      CALL RDLINE
55900      RETURN
56000      C
56100      C   HERE WE HAVE ONE RULE. COUNT THE NUMBER OF OTHER EXPLICIT ENTRIES
56200      C   IN THIS RULE IN THE CURRENT SUBTABLE.
56300      60 CALL COUNT(TRULES(1))
56400      IF(NEE.GT.0) GO TO 60
56500      TSON(K) = -TRULES(1)
56600      GO TO 90
56700      C
56800      C   HERE WE HAVE SATISFIED AN ELSE RULE
56900      70 TSON(K) = 0
57000      GO TO 90
57100      C
57200      C   HERE WE BUILD A NEW SUBTABLE
57300      80 PARENT(KK) = K
57400      TSON(K) = KK
57500      CALL SUBTAB(1)
57600      GO TO 20
57700      C
57800      C   COUNT THE FALSE ENTRIES IN SELECTED CONDITION
57900      C
58000      90 NF = 0
58100      DO 100 I = 1, NR
58200      IF(SUB((NC+I),K) .EQ. 0) GO TO 100
58300      IF(RULES(COND(K),I) .GT. 0) GO TO 100
58400      NF = NF + 1
58500      FRULES(NF) = I
58600      100 CONTINUE
58700      IF(NF=1) 130, 120, 110
58800      C
58900      C   HERE WE HAVE MULTIPLE RULES. CHECK FOR REDUNDANCY
59000      110 IF(CNCK(K).GT.1) GO TO 140
59100      WRITE(6,1000) (FRULES(I),I=1,NF)
59200      WRITE(6,1010)
59300      SWITCH = .FALSE.
59400      CALL RDLINE
59500      RETURN
59600      C
59700      C   HERE WE HAVE ONE RULE. COUNT THE NUMBER OF OTHER EXPLICIT ENTRIES
59800      C   IN THIS RULE IN THE CURRENT SUBTABLE.
59900      120 CALL COUNT(FRULES(1))
60000      IF(NEE.GT.0) GO TO 140
60100      FSON(K) = -FRULES(1)
60200      GO TO 150
60300      C
60400      C   HERE WE HAVE THE ELSE RULE
60500      130 FSON(K) = 0
60600      GO TO 150
60700      C
60800      C   HERE WE BUILD A NEW SUBTABLE

```

```
60900      140 PARENT(KK) = KK
61000      FSON(K) = KK
61100      CALL SUBTAB(-1)
61200      GO TO 20
61300      C
61400      C   LOOK FOR THE PARENT SUBTABLE
61500      C
61600      150 IF(PARENT(K)) 160, 180, 170
61700      C
61800      C   FALSE SON HAS BEEN CHECKED, GO UP ONE LEVEL
61900      160 K = IABS(PARENT(K))
62000      GO TO 150
62100      C
62200      C   FALSE SON HAS NOT BEEN CHECKED
62300      170 K = PARENT(K)
62400      GO TO 90
62500      C
62600      C   PARENT IS THE ROOT, FINISHED WITH THE DECOMPOSITION
62700      180 SWITCH = .TRUE.
62800      C
62900      C   MAKE THE ORDERED LIST OF CONDITIONS SINGLE-VALUED
63000      CALL COMPAC(COND, NC, NCOND)
63100      RETURN
63200      C
63300      1000 FORMAT(1H, "THE FOLLOWING RULES ARE REDUNDANT OR CONTRADICTIONARY:",
63400      1 (515))
63500      1010 FORMAT(1H, "YOUR MUST MODIFY THE DATA.",
63600      1 "   ENTER THE MODIFY OR STOP COMMAND.")
63700      END
63800      C
63900      C   - - - - -
64000      C
64100      SUBROUTINE ZERO(A, B)
64200      INTEGER A(1), B
64300      DO 10 I = 1, B
64400      10 A(I) = 0
64500      RETURN
64600      END
64700      C
64800      C   - - - - -
64900      C
65000      SUBROUTINE METER
65100      $ INCLUDE 'TABCOM'
65200      C
65300      C   COUNT THE IMMATERIAL ENTRIES IN THE APPLICABLE RULES
65400      C   FOR EACH CONDITION IN THIS SUBTABLE.
65500      I1 = 0
65600      CALL ZERO(CK, NC)
65700      DO 20 I = 1, NC
65800      IF(SUB(I,K) .EQ. 0) GO TO 20
65900      I1 = I1 + 1
66000      CK(I1) = I
66100      DO 10 J = 1, NR
66200      IF(SUB((NC+J), K) .EQ. 0) GO TO 10
66300      IF( RULES(I,J) .EQ. 0 ) IC(I1) = IC(I1) + 1
66400      10 CONTINUE
66500      20 CONTINUE
66600      RETURN
66700      END
66800      C
66900      C   - - - - -
67000      C
```

```

67100      SUBROUTINE ROW(A, B, Y, Z, AA, AAA)
67200      $ INCLUDE 'TABCOM'
67300      C
67400      INTEGER A(1), B(1), Y, Z, TEST, AA, AAA(1)
67500      C
67600      C      INITIALIZE SORTING VARIABLES
67700      CALL ZERU(B,NC)
67800      Y = 1
67900      B(1) = AAA(1)
68000      TEST = A(1)
68100      C
68200      C      SORT THE LIST TO FIND THE ROWS WITH THE SMALLEST
68300      C      (OR LARGEST) NUMBERS IN THE LIST A
68400      DO 30 J = 2, AA
68500      IF(Z*( A(J) - TEST )) 10, 20, 30
68600      10 Y = 1
68700      B(1) = AAA(J)
68800      TEST = A(J)
68900      GO TO 30
69000      20 Y = Y + 1
69100      B(Y) = AAA(J)
69200      30 CONTINUE
69300      RETURN
69400      END
69500      C
69600      C -----
69700      C
69800      SUBROUTINE EXPLII
69900      $ INCLUDE 'TABCOM'
70000      C
70100      C      COUNT THE EXPLICIT ENTRIES IN THE APPLICABLE RULES
70200      C      FOR EACH OF THE MATERIAL CONDITIONS.
70300      DO 20 I1 = 1, NMR
70400      I = MR(I1)
70500      DO 10 J = 1, NR
70600      IF(SUB(NC+J,K) .EQ. 0) GO TO 10
70700      IF(IABS(RULES(I,J)) .EQ. 1) EC(I1) = EC(I1) + 1
70800      10 CONTINUE
70900      20 CONTINUE
71000      RETURN
71100      END
71200      C
71300      C -----
71400      C
71500      SUBROUTINE DELTA
71600      $ INCLUDE 'TABCOM'
71700      C
71800      C      COUNT THE DIFFERENCE BETWEEN THE YES AND NO ENTRIES
71900      C      IN THE APPLICABLE RULES FOR EACH OF THE EXPLICIT CONDITIONS
72000      DO 20 I1 = 1, NER
72100      I = ER(I1)
72200      DO 10 J = 1, NR
72300      IF(SUB(NC+J,K) .EQ. 0) GO TO 10
72400      IF(IABS(RULES(I,J)) .LE. 1) DC(I1) = DC(I1) + RULES(I,J)
72500      10 CONTINUE
72600      DC(I1) = IABS(DC(I1))
72700      20 CONTINUE
72800      C
72900      RETURN
73000      END
73100      C
73200      C -----

```

```

73300 C
73400 SUBROUTINE SUBTAB(LOGIC)
73500 $ INCLUDE 'TABCOM'
73600 C
73700 DO 10 I = 1, NC
73800 10 SUB(I, KK) = SUB(I, K)
73900 SUB(COND(K), KK) = 0
74000 IF(LOGIC.LT.0) GO TO 30
74100 DO 20 I = 1, NT
74200 20 SUB(NC+TRULES(I), KK) = 1
74300 GO TO 50
74400 30 DO 40 I = 1, NF
74500 40 SUB(NC+FRULES(I), KK) = 1
74600 50 NCK(KK) = NCK(K) - 1
74700 K = KK
74800 KK = KK + 1
74900 RETURN
75000 END
75100 C
75200 C -----
75300 C
75400 SUBROUTINE COUNT(J)
75500 $ INCLUDE 'TABCOM'
75600 C
75700 NEE = 0
75800 DO 10 I = 1, NC
75900 IF(I.EQ.COND(K)) GO TO 10
76000 IF(SUB(I, K).EQ.0) GO TO 10
76100 IF(ABS(RULES(I, J)).EQ.1) NEE = NEE + 1
76200 10 CONTINUE
76300 RETURN
76400 END
76500 C
76600 C -----
76700 C
76800 SUBROUTINE COMPAC(I, J, L)
76900 INTEGER I(1), L(1), HIT(60)
77000 DO 5 M = 1, 60
77100 5 HIT(M) = 0
77200 II = 0
77300 DO 10 M = 1, 60
77400 IF(I(M).EQ.J) GO TO 20
77500 IF(HIT(I(M)).NE.0) GO TO 10
77600 II = II + 1
77700 HIT(I(M)) = 1
77800 L(II) = I(M)
77900 10 CONTINUE
78000 20 IF(II.NE.J) WRITE(0,1000)
78100 RETURN
78200 1000 FORMAT(1H, 'NETWORK IS INCOMPLETE---EXAMINE FOR ERRORS.')
78300 END
78400 C
78500 C -----
78600 C
78700 SUBROUTINE TOPU
78800 $ INCLUDE 'TABCOM'
78900 C
79000 C SET THE INITIAL CONDITIONS TO TRAVERSE THE BINARY TREE IN PREORDER
79100 C FINDING THE DISTANCE OF THE NUDES FROM EACH END AS WE GO.
79200 C
79300 J = 2
79400 K = 1

```

```
79500      L = 1
79600      M = 1
79700      ORDER(1) = COND(1)
79800      LEVEL(1) = 0
79900      PTRBK(1) = 0
80000      BKSTK(1) = 1
80100      C
80200      C   TEST THE TRUE SON TO SEE IF IT IS A RULE OR A CONDITON
80300      10 IF(TSON(K).LE.0) GO TO 20
80400      C   HERE IT IS A CONDITION
80500      ORDER(J) = COND(TSON(K))
80600      LEVEL(J) = L
80700      PTRBK(J) = J - 1
80800      M = M + 1
80900      BKSTK(M) = J
81000      J = J + 1
81100      K = TSON(K)
81200      L = L + 1
81300      GO TO 10
81400      C   HERE IT IS A RULE
81500      20 ORDER(J) = TSON(K)
81600      LEVEL(J) = L
81700      PTRBK(J) = J - 1
81800      J = J + 1
81900      C
82000      C   TEST THE FALSE SON TO SEE IF IT IS CONDITION OR A RULE
82100      30 IF(FSON(K).LE.0) GO TO 40
82200      C   HERE IT IS A CONDITION
82300      ORDER(J) = COND(FSON(K))
82400      LEVEL(J) = L
82500      PTRBK(J) = BKSTK(M)
82600      BKSTK(M) = J
82700      J = J + 1
82800      K = FSON(K)
82900      L = L + 1
83000      GO TO 10
83100      C   HERE IT IS A RULE
83200      40 ORDER(J) = FSON(K)
83300      LEVEL(J) = L
83400      PTRBK(J) = BKSTK(M)
83500      M = M + 1
83600      JJ = J
83700      J = J + 1
83800      LL = 1
83900      C
84000      C   PROCEED BACK UP TO THE PARENT CONDITION
84100      50 L = L - 1
84200      PK = PARENT(K)
84300      K = IABS(PK)
84400      C   SET THE LENGTH AT THE PARENT EQUAL TO THE MAXIMUM OF THE TWO PATHS
84500      IF(LENGTH(PTRBK(JJ)).LT.LL) LENGTH(PTRBK(JJ)) = LL
84600      LL = LENGTH(PTRBK(JJ)) + 1
84700      JJ = PTRBK(JJ)
84800      C   FIND OUT WHETHER WE ARE APPROACHING FROM TRUE OR FALSE BRANCH
84900      IF(PK) 50, 60, 55
85000      C   SET THE LENGTH AT THE PARENT
85100      55 LENGTH(PTRBK(JJ)) = LL
85200      GO TO 30
85300      C
85400      C   HERE WE ARE AT THE ROOT NODE, STORE THE
85500      C   NUMBER OF NODES IN THE NETWORK.
85600      60 NN = J - 1
```



```

85700 C
85800 C MAKE THE PREORDER LIST OF THE RULES
85900 K = 0
86000 DU 70 I = 1, NR
86100 IF(ORDER(I) .GE. 0) GO TO 70
86200 K = K + 1
86300 NRULE(K) = IAS(ORDER(I))
86400 70 CONTINUE
86500 C
86600 C CHECK THE NUMBER OF THE RULES
86700 IF(K .NE. NR) CALL COMPAC(NRULE, NR, NRULE)
86800 RETURN
86900 END
87000 C
87100 C - - - - -
87200 C
87300 SUBROUTINE OUT1
87400 S INCLUDE 'TABCOM'
87500 C
87600 C WRITE THE PAGE HEADING AND THE ORIGINAL TABLE
87700 C
87800 IF(.NOT. TABTWO) WRITE(FLOUT,1000)
87900 ICL = 1
88000 IRUL = 1
88100 JRUL = NR
88200 IF((FLOUT .NE. 6) .OR. (NR .LE. 12)) GO TO 10
88300 JRUL = 12
88400 10 IF(.NOT. TABTWO) WRITE(FLOUT,1010) TITLE, (I,I=IRUL,JRUL)
88500 IF(TABTWO) WRITE(FLOUT,1010) TITLE, (NRULE(I),I=IRUL,JRUL)
88600 DO 30 II = 1, NC
88700 I = II
88800 IF(TABTWO) I = NCOND(II)
88900 IF(TABTWO) CALL SUBSCR(NCS, I, ICL)
89000 WRITE(FLOUT,1020) I, (CONDS(J,ICL),J=1,5), ASTER,
89100 1 (SIGN(RULES(I,J)+3),J=IRUL,JRUL)
89200 ICL = ICL + 1
89300 IF(NCS(I) .LT. 2) GO TO 30
89400 JJ = NCS(I)
89500 DU 20 K = 2, JJ
89600 WRITE(FLOUT,1040) (CONDS(J,ICL),J=1,5), ASTER
89700 20 ICL = ICL + 1
89800 30 CONTINUE
89900 J = 36 + 3*(JRUL + 1 - IRUL)
90000 WRITE(FLOUT,1030) (DIVIDE(I),I=1,J)
90100 IAL = 1
90200 DU 50 I = 1, NA
90300 DU 40 JJ = IRUL, JRUL
90400 ACTION(JJ) = BLANK
90500 J = JJ
90600 IF(TABTWO) J = NRULE(JJ)
90700 IF(ACTPTR(J).EQ.1) ACTION(JJ) = XX
90800 40 CONTINUE
90900 WRITE(FLOUT,1020) I, (ACTS(K,IAL),K=1,5), ASTER,
91000 1 (ACTION(J),J=IRUL,JRUL)
91100 IAL = IAL + 1
91200 IF(NAS(I) .EQ. 1) GO TO 60
91300 JJ = NAS(I)
91400 DU 50 K = 2, JJ
91500 WRITE(FLOUT,1040) (ACTS(J,IAL),J=1,5), ASTER
91600 50 IAL = IAL + 1
91700 60 CONTINUE
91800 IF(NR .LE. JRUL) RETURN

```

```

91900      IRUL = IRUL + 12
92000      JRUL = JRUL + 12
92100      IF(JRUL .GT. NR) JRUL = NR
92200      GO TO 10
92300      RETURN
92400      C
92500      1000 FORMAT(1H1, "ORIGINAL DECISION TABLE")
92600      1010 FORMAT(1H0, '//, 10x, 10A6, '// 30x, 12I3)
92700      1020 FORMAT(1H , 12, x, 5A6), x, 12(A1, 2X), A1)
92800      1030 FORMAT(1H , 72A1)
92900      1040 FORMAT(1H , 3x, 5A6, x, A1)
93000      END

```

```

93100      C
93200      C -----
93300      C

```

```

93400      SUBROUTINE SUBSCR(1, J, K)
93500      DIMENSION I(1)
93600      K = 1
93700      IF(J .EQ. 1) RETURN
93800      M = J - 1
93900      DO 10 L = 1, M
94000      10 K = K + I(L)
94100      RETURN
94200      END

```

```

94300      C
94400      C -----
94500      C

```

```

94600      SUBROUTINE OUT2
94700      $ INCLUDE 'TABCOM'
94800      C
94900      C WRITE THE DECISION NETWORK
95000      C
95100      CALL ARRAY
95200      IF(FLOUT .EQ. 6) GO TO 30
95300      DO 20 J = 1, JMAX
95400      20 WRITE(FLOUT,1010) (NET(I,J),I=1,132)
95500      RETURN
95600      30 DO 40 J = 1, JMAX
95700      40 WRITE(FLOUT,1030) (NET(I,J),I=1,72)
95800      IF(JMAX.LE.72) RETURN
95900      DO 50 J = 1, JMAX
96000      50 WRITE(FLOUT,1030) (NET(I,J),I=73,132)
96100      RETURN
96200      1010 FORMAT(1H , 132A1)
96300      1030 FORMAT(1H , 72A1)
96400      END

```

```

96500      C
96600      C -----
96700      C

```

```

96800      SUBROUTINE ARRAY
96900      $ INCLUDE 'TABCOM'
97000      C
97100      C CONSTRUCT THE OUTPUT FOR THE DECISION NETWORK
97200      INTEGER SKC(15)
97300      DO 5 I = 1, 120
97400      DO 5 J = 1, 132
97500      5 NET(J,I) = BLANK
97600      DO 5 I = 1, 15
97700      CVB(I) = 0
97800      8 LVB(I) = 0
97900      JMAX = 0
98000      IMAX = 0

```

```
98100      J = 1
98200      K = 0
98300      KC = 1
98400      M = 0
98500      MK = 0
98600      10 K = K + 1
98700      IF(K.GT.MN) RETURN
98800      I = 10*LEVEL(K)
98900      C
99000      C CHECK TO SEE IF THE NODE IS A CONDITION OR A RULE
99100      IF(ORDER(K)) 60, 50, 1
99200      C
99300      C THE NODE IS A CONDITION, WRITE IT IN THE ARRAY
99400      1 NET((I+1),J) = C
99500      CALL NUMB(I,J)
99600      C
99700      C FIND THE SIGN OF THE NEXT BRANCH AND THE NUMBER OF THE
99800      C NEXT SUBTABLE
99900      C
100000     IF(TSON(KC)) 14, 14, 11
100100     11 IF(FSON(KC)) 13, 13, 12
100200     12 IF(ORDER(K+1) .EQ. COND(TSON(KC))) GO TO 17
100300     GO TO 19
100400     13 IF(ORDER(K+1) .EQ. COND(TSON(KC))) GO TO 18
100500     GO TO 40
100600     14 IF(FSON(KC)) 16, 16, 15
100700     15 IF(ORDER(K+1) .EQ. TSON(KC)) GO TO 22
100800     GO TO 23
100900     16 IF(ORDER(K+1) .EQ. TSON(KC)) GO TO 25
101000     GO TO 26
101100     C
101200     17 TRUE BRANCH, NEXT TRUE SUBTABLE
101300     MK = MK + 1
101400     SKC(MK) = FSON(KC)
101500     18 KC = TSON(KC)
101600     BRANCH = .TRUE.
101700     GO TO 28
101800     C
101900     19 FALSE BRANCH, NEXT SUBTABLE ALONG IT
102000     MK = MK + 1
102100     SKC(MK) = TSON(KC)
102200     KC = FSON(KC)
102300     GO TO 21
102400     20 KC = TSON(KC)
102500     21 BRANCH = .FALSE.
102600     GO TO 28
102700     C
102800     22 NEXT SUBTABLE IS ON FALSE BRANCH
102900     BRANCH = .TRUE.
103000     GO TO 24
103100     23 BRANCH = .FALSE.
103200     24 KC = FSON(KC)
103300     GO TO 25
103400     C
103500     25 NEXT SUBTABLE IS BACK IN STACK
103600     BRANCH = .TRUE.
103700     GO TO 27
103800     26 BRANCH = .FALSE.
103900     27 IF(MK .EQ. 0) GO TO 28
104000     KC = SKC(MK)
104100     MK = MK - 1
104200     C
104300     28 PROCEED WITH THE NETWORK
104400     IF(BRANCH) GO TO 29
104500     ROW = MINUS
104600     COLUMN = PLUS
104700     CVB(LEVEL(K) + 1) = -1
```

```
104300      GO TO 30
104400      29 ROW = PLUS
104500      COLUMN = MINUS
104600      CVB(LEVEL(K) + 1) = 1
104700      C
104800      C   EXTEND THE HORIZONTAL BRANCH
104900      30 IS = I + 5
105000      I9 = I + 9
105100      DO 35 II = IS, I9, 2
105200      35 NET(II,J) = ROW
105300      C
105400      C   EXTEND A VERTICAL BRANCH DOWN FROM THE CONDITION
105500      J1 = J + 1
105600      J3 = J + 3
105700      DO 40 JJ = J1, J3
105800      40 NET((I+2),JJ) = COLUMN
105900      M = M + 1
106000      LVB(M) = I
106100      GO TO 10
106200      C
106300      C   THE NODE IS AN ELSE RULE, WRITE IT IN THE ARRAY
106400      50 NET((I-1),J) = E
106500      NET(I,J) = EL
106600      NET((I+1),J) = S
106700      NET((I+2),J) = E
106800      IF((I+2).GT.IMAX) IMAX=I+2
106900      GO TO 70
107000      C
107100      C   THE NODE IS A RULE, WRITE IT IN THE ARRAY
107200      60 NET((I+1),J) = R
107300      CALL NUMB(I,J)
107400      IF((I+2).GT.IMAX) IMAX = I + 2
107500      C
107600      C   EXTEND THE LAST VERTICAL BRANCH ACROSS TO THE NEXT NODE
107700      70 IF(M.LT.1) RETURN
107800      J = J + 3
107900      IF(JMAX.LT.J) JMAX = J
108000      I = LVB(M)
108100      IS = I + 5
108200      I9 = I + 9
108300      POINT = MINUS
108400      IF(CVB((I/10) + 1).EQ.-1) POINT = PLUS
108500      CVB((I/10) + 1) = 0
108600      M = M - 1
108700      DO 80 II = IS, I9, 2
108800      80 NET(II,J) = POINT
108900      C
109000      C   EXTEND THE INCOMPLETE VERTICAL BRANCHES
109100      IF(M.EQ.0) GO TO 10
109200      J1 = J + 1
109300      J3 = J + 3
109400      JM = LVB(M) + 2
109500      DO 130 II = 2, JM, 10
109600      IF(CVB((II-2)/10) + 1) 90, 130, 100
109700      90 POINT = PLUS
109800      GO TO 110
109900      100 POINT = MINUS
110000      110 DO 120 JJ = J1, J3
110100      NET(II,JJ) = POINT
110200      120 CONTINUE
110300      130 CONTINUE
110400      GO TO 10
```

```

110500          END
110600      C -----
110700      C
110800      C
110900          SUBROUTINE NUMB(I,J)
111000      $ INCLUDE 'TABCOM'
111100      C
111200          NUMBER = IABS(ORDER(K))
111300          IF(NUMBER.GT.9) GO TO 10
111400          NET((I+2),J) = SYMBOL(NUMBER + 1)
111500          IF((I+2).GT.IMAX) IMAX=I+2
111600          RETURN
111700      C
111800      10 N10 = NUMBER / 10
111900          N1 = NUMBER - 10*N10
112000          NET((I+2),J) = SYMBOL(N10+1)
112100          NET((I+3),J) = SYMBOL(N1+1)
112200          IF((I+3).GT.IMAX) IMAX=I+3
112300          RETURN
112400          END
112500      C -----
112600      C
112700      C
112800          SUBROUTINE SORT
112900      $ INCLUDE 'TABCOM'
113000      C
113100      C      SET THE INITIAL CONDITIONS. THE VECTORS PTRBK, LVB, AND
113200      C      BKSTK WHICH WERE USED IN THE LUPO ROUTINE WILL BE
113300      C      USED HERE FOR TEMPORARY STORAGE.
113400      C      CALL ZERU(PTRBK,NN)
113500      C      CALL ZERU(BKSTK,NN)
113600      C      CALL ZERU(LVB,10)
113700      C      J = 1
113800      C      K = 1
113900      C      M = 0
114000      C
114100      C      PUT THE FIRST CONDITION NODE IN THE SORTED LIST
114200      C      PTRBK(K) = ORDER(J)
114300      C      BKSTK(K) = LEVEL(J)
114400      C      K = K + 1
114500      C      FIND THE SONS OF THE CURRENT NODE
114600      C      10 J1 = J + 1
114700      C      J2 = J + 2
114800      C      L = LEVEL(J1)
114900      C
115000      C      J1 IS THE TRUE SON. FIND THE FALSE SON OF THIS CONDITION
115100      C      DO 20 I = J2, NN
115200      C      IF(LEVEL(I) .EQ. L) GO TO 30
115300      C      20 CONTINUE
115400      C
115500      C      FIND THE SHORTEST BRANCH
115600      C      30 IF(LENGTH(J1) .GT. LENGTH(I)) GO TO 40
115700      C      N1 = J1
115800      C      N2 = I
115900      C      GO TO 45
116000      C      40 N1 = I
116100      C      N2 = J1
116200      C
116300      C      PUT THE NODE ON THE SHORTEST BRANCH IN THE ORDERED LIST
116400      C      45 PTRBK(K) = ORDER(N1)
116500      C      BKSTK(K) = LEVEL(N1)
116600      C      K = K + 1

```

```
116700 C
116800 C CHECK FOR NODES BETWEEN I AND J2
116900 IF(I .EQ. J2) GO TO 50
117000 IF(ORDER(N1)) 50, 50, 40
117100 48 M = M + 1
117200 LVB(M) = N2
117300 J = N1
117400 IF(I .GE. NN) GO TO 55
117500 GO TO 10
117600 50 PTRBK(K) = ORDER(N2)
117700 BKSTK(K) = LEVEL(N2)
117800 K = K + 1
117900 C
118000 C SEE IF N2 IS A CONDITION
118100 IF(ORDER(N2) .LE. 0) GO TO 55
118200 C N2 IS A CONDITION
118300 J = N2
118400 GO TO 10
118500 C
118600 C CHECK TO SEE IF WE ARE FINISHED
118700 55 IF(K .GT. NN) GO TO 70
118800 C
118900 C CHECK TO SEE IF WE HAVE NODES WAITING IN THE STACK
119000 IF(M .EQ. 0) GO TO 60
119100 J = LVB(M)
119200 PTRBK(K) = ORDER(J)
119300 BKSTK(K) = LEVEL(J)
119400 K = K + 1
119500 M = M - 1
119600 GO TO 10
119700 60 J = N2
119800 GO TO 10
119900 C
120000 C HERE WE ARE DONE. PUT THE ORDERED NETWORK INTO ORDER.
120100 C REORDER THE LEVEL VECTOR, AND SET NCOND AND NRULES.
120200 70 J = 0
120300 K = 0
120400 DO 100 I = 1, NN
120500 ORDER(I) = PTRBK(I)
120600 LEVEL(I) = BKSTK(I)
120700 IF(ORDER(I)) 80, 100, 90
120800 C HERE IT IS A RULE
120900 80 J = J + 1
121000 NRULE(J) = IABS(ORDER(I))
121100 GO TO 100
121200 C HERE IT IS A CONDITION
121300 90 K = K + 1
121400 NCOND(K) = ORDER(I)
121500 100 CONTINUE
121600 C
121700 C CHECK THE NUMBER OF RULES
121800 IF(J .NE. NR) CALL COMPAC(NRULE, NR, NRULE)
121900 C
122000 C CHECK THE NUMBER OF CONDITIONS
122100 IF(K .NE. NC) CALL COMPAC(NCOND, NC, NCOND)
122200 C
122300 C THE LISTS ORDER, NCOND, AND NRULE ARE NOW SORTED
122400 C BY BRANCH LENGTH AND READY TO BE USED FOR OUTPUT.
122500 C
122600 RETURN
122700 END
122800 C
```

```
122900 C -----
123000 C
123100 SUBROUTINE ERROR(I)
123200 IF(I .GT. 10) GO TO 10
123300 WRITE(6,1000)
123400 RETURN
123500 10 WRITE(6,1010)
123600 WRITE(1,1010)
123700 STOP
123800 C
123900 1000 FORMAT(1H , "INCORRECT INPUT---REENTER ON A NEW LINE")
124000 1010 FORMAT(1H , "INCORRECT INPUT---PROGRAM EXECUTION STOPPED.")
124100 RETURN
124200 END
124300 C
124400 C -----
124500 C
124600 BLOCK DATA
124700 $ INCLUDE 'TABCOM'
124800 C
124900 DATA BLANK, ASTER, XX, C, E, R, PLUS, MINUS, EL, S/6H ,
125000 1 6H* , 6HX , 6HC , 6HE , 6HR ,
125100 2 6H+ , 6H- , 6HL , 6HS , / ,
125200 3 (SYMBOL(I),I=1,10), (SIGN(I),I=1,5)/6H0 ,
125300 4 6H1 , 6H2 , 6H3 , 6H4 , 6H5 ,
125400 5 6H6 , 6H7 , 6H8 , 6H9 , 6H- ,
125500 6 6HF , 6H. , 6HT , 6H+ /
125600 END
```

Chapter Two

INFORMATION NETWORK PROGRAM

2.1 Algorithms

The purpose of the program is to assemble and store network data and to enable the user to examine any ingredience or dependence network that he desires. The only data required to operate the program is a list of node numbers and their ingredient nodes. The alphameric data stored as labels and titles for the nodes simply enhance the readability of the output.

The dependents of each node are calculated from the list of ingredients by passing through the list twice. Each time a node appears in the list of ingredients, it has one dependent. The first pass through the list of ingredients is used to build a vector that has the number of dependents of each node recorded. From this vector a new vector is constructed that will serve as a pointer to the rows in the list of dependents where the dependents of each node will be stored. A second pass is then made through the list of ingredients to fill the proper values into the list of dependents. Subroutine DEPEN accomplishes this task.

The input level, output level, and total float of each node are calculated in a manner very similar to the calculation of the early start, late finish, and total float in the construction activity scheduling algorithm. (ref. 2.1) The first step is to arrange the nodes in an ordered list, so that no node is placed before any of its ingredients. This allows the straightforward calculation of the input level for all the nodes in the

network. The input level is taken as zero for the starting node and then incremented by one for each branch that is passed through. Where a node has several ingredients, the input level is taken as one plus the maximum input level of any of its ingredients. In this way, the input level is the number of branches between the node and the input nodes along the longest path coming into the node. The output level is calculated in a similar manner by traveling through the ordered list in the opposite direction, and assigning level zero to those nodes with no dependents (the "output" nodes). A float is calculated for each node by subtracting the sum of its input and output levels from the length of the network. The length of the network is the number of branches along the longest path from input to output in the network. It is equal to the largest input level or output level calculated for any node. Subroutine LEVEL calculates all of these values. (see Fig. 2.6).

The dependents, input level, output level, and total float are not stored permanently in the data file, but are recreated from the stored data each time the network is called by the program. The ingredients are the only topologic data permanently stored.

The ingredience and dependence networks are formulated in the manner called preorder by Knuth (ref. 1.4). The algorithm for either is the same, except one passes along the ingredients while the other goes to the dependents. In this discussion, the term sons will be used in a generic sense. The algorithm can be summarized as four basic steps:

1. Start at the root node; put it in a stack and in the output list.
2. Does the last node in the stack have any sons? If yes, go to 3.

If no, go to 4.

3. Obtain the next son of the node, put it in the stack and in the output list. Remove it from its fathers list of sons. Go to 2.
4. Remove the node from the stack and go to 2. Unless there are no nodes left in the stack, then stop.

Subroutine GLOBAL travels through the network as described (see Fig. 2.8). The output list is prepared line by line on a data file, then recalled a line at a time when its printed out.

The sorting processes that the program can accomplish simply modifies the order in which the sons of any particular node are listed. It is possible to list the sons so that those with either the smallest or largest level or float will appear first. Subroutine SORT is where the reordering is done (Fig. 2.7.). Note that if no sorting is asked for, the sons will be listed in an order that is the reverse of the way that they were entered in the input.

2.2 Logic Diagrams

The following diagrams describe the overall structure of the program and the major subroutines. For a more detailed study of the program, consult the listing of the FORTRAN code.

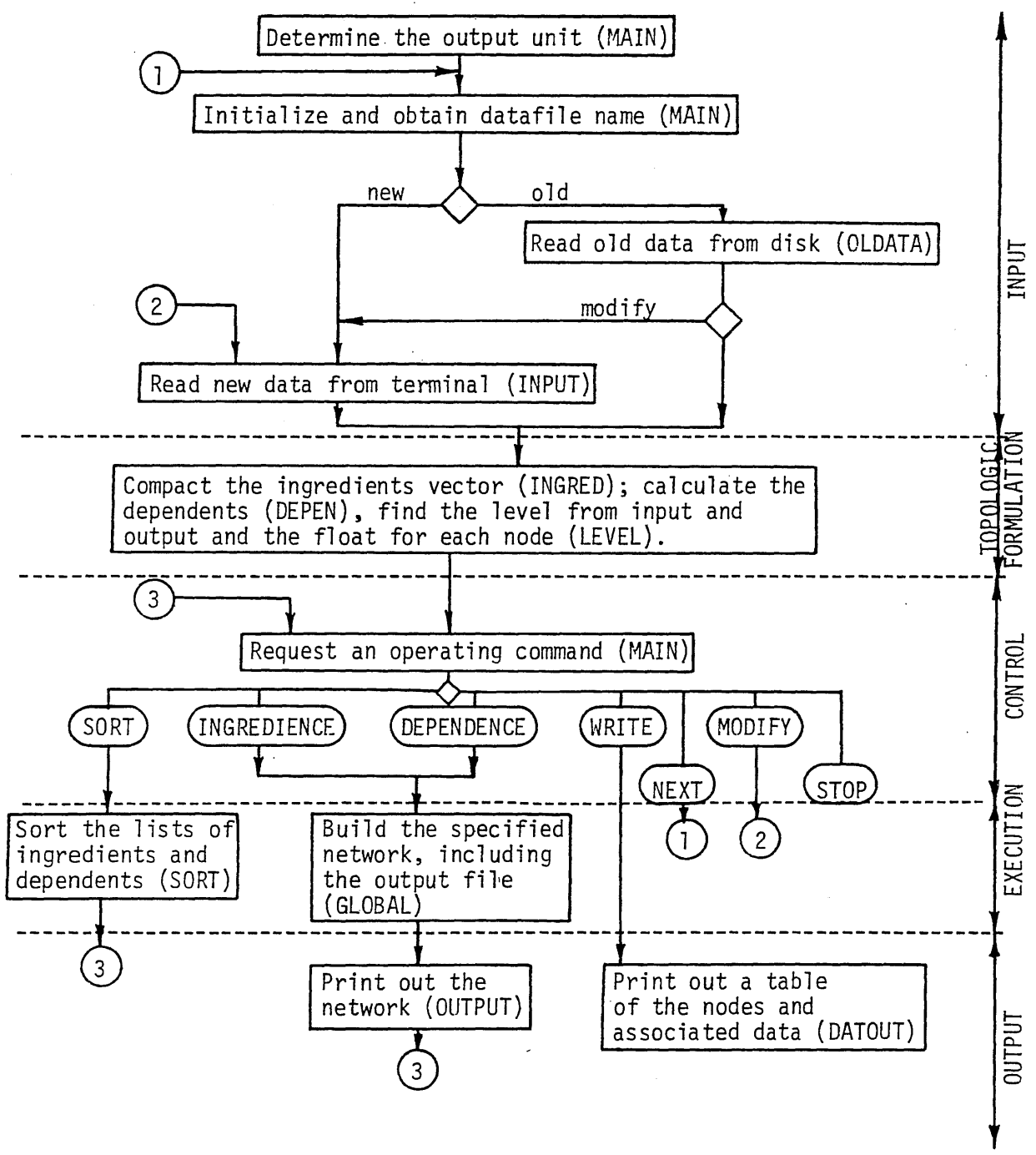
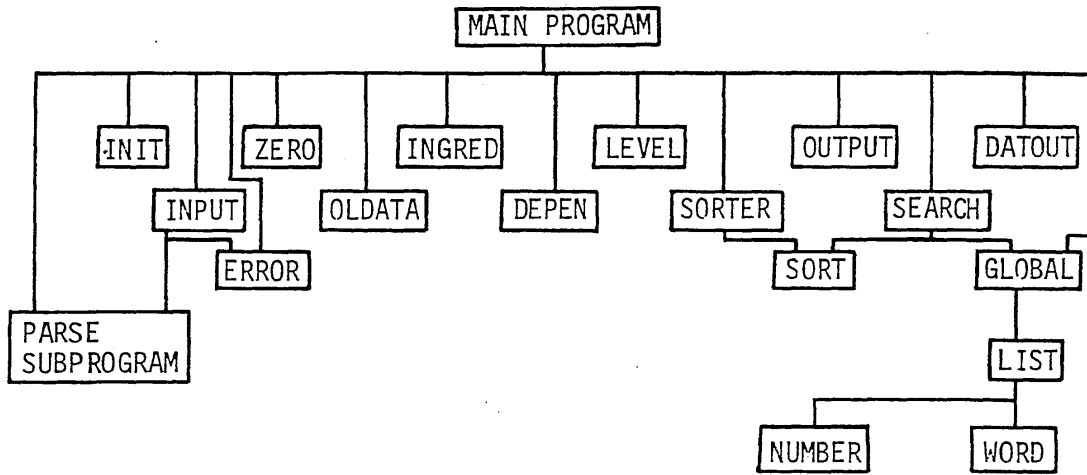


Figure 2.1 Block Diagram for the Information Network Program



NOTES:

1. PARSE contains several functions and subroutines, including WDINIT, RDLIN, END, MATCH, FIXED, STRING, and SETOUT.
2. INPUT and MAIN include the file PARCOM, used for the COMMON declaration needed for PARSE

Figure 2.2 Subroutine Linkage, (NETWORK)

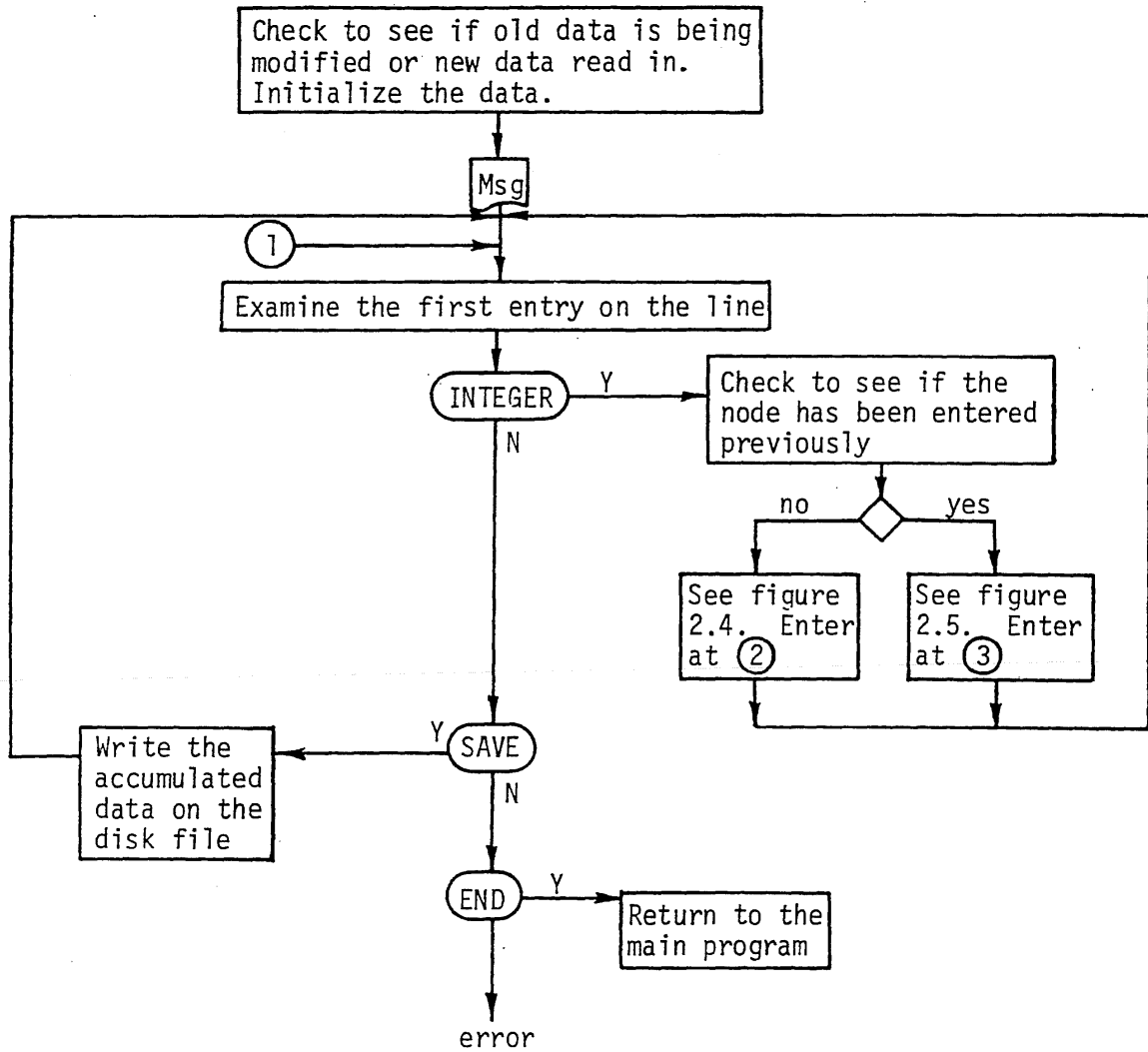


Figure 2.3 Subroutine INPUT (NETWORK)

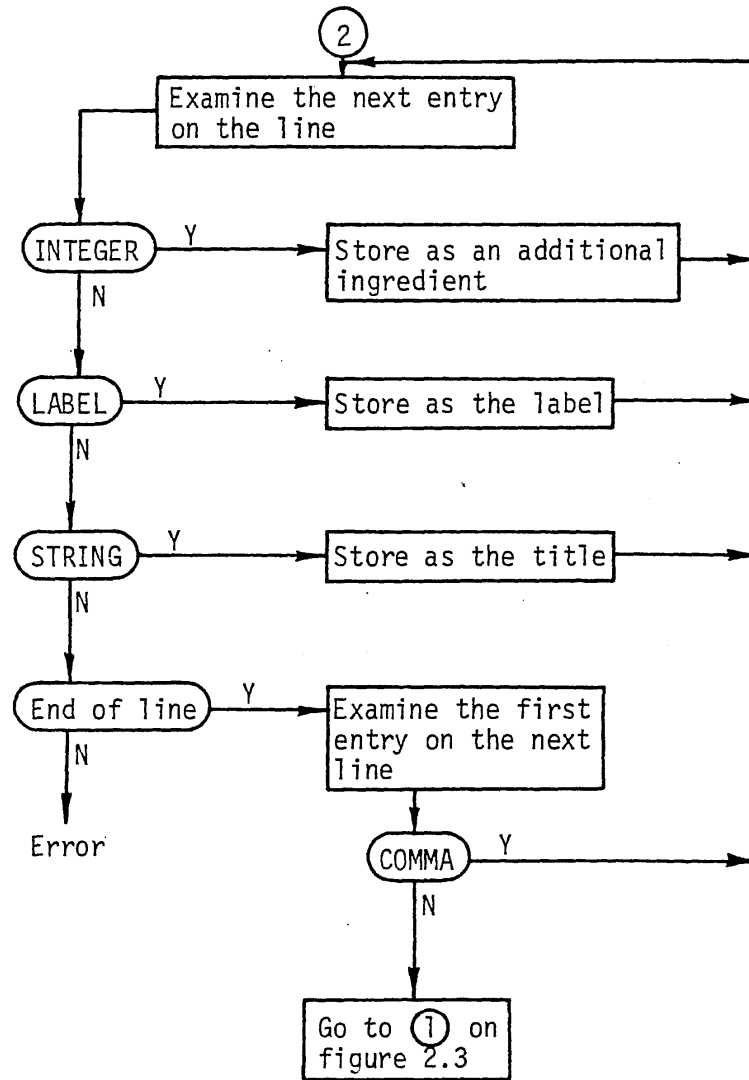


Figure 2.4 Input of Data for a New Node (NETWORK)

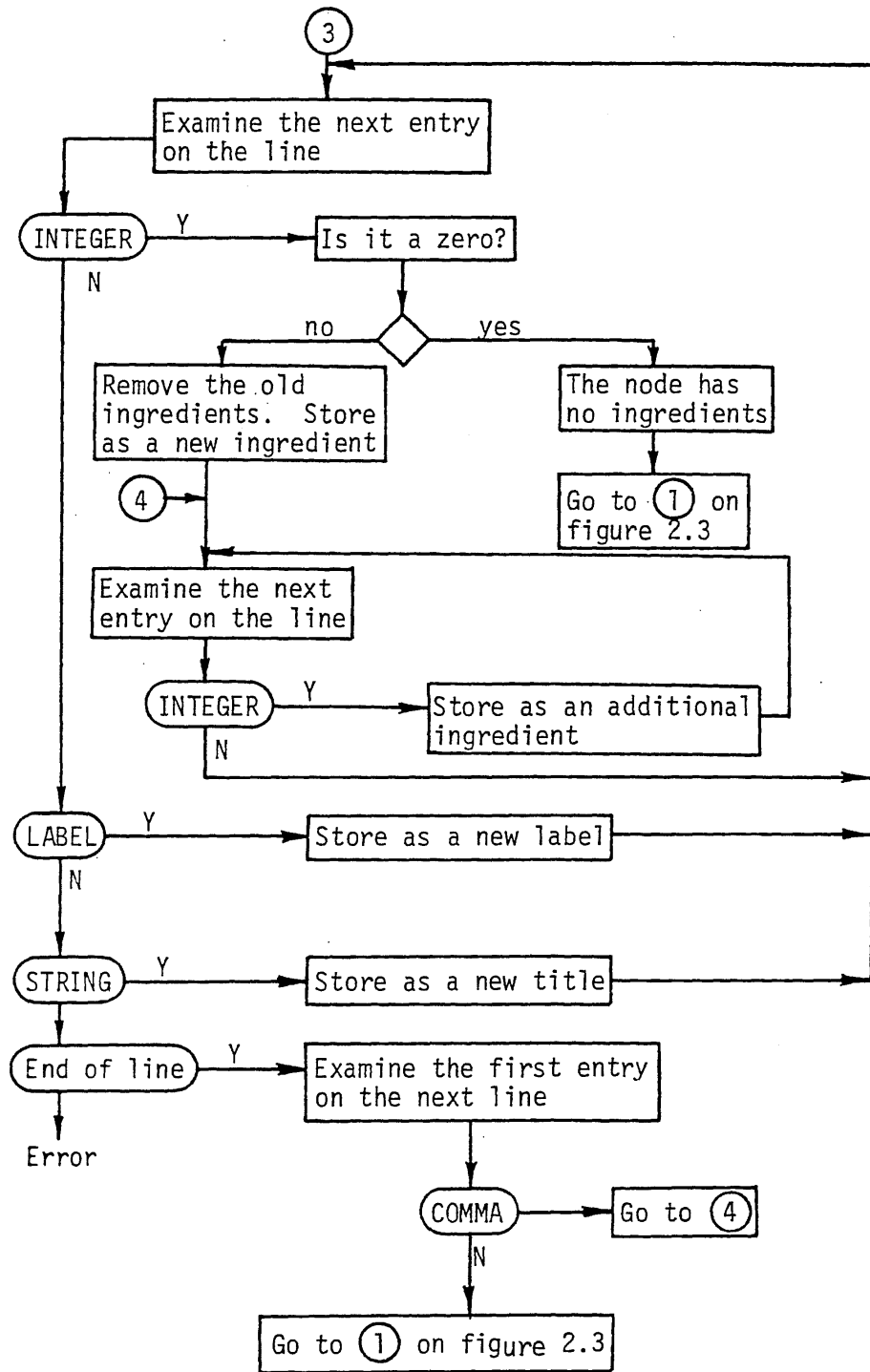


Figure 2.5 Input of New Data for a Node (NETWORK)

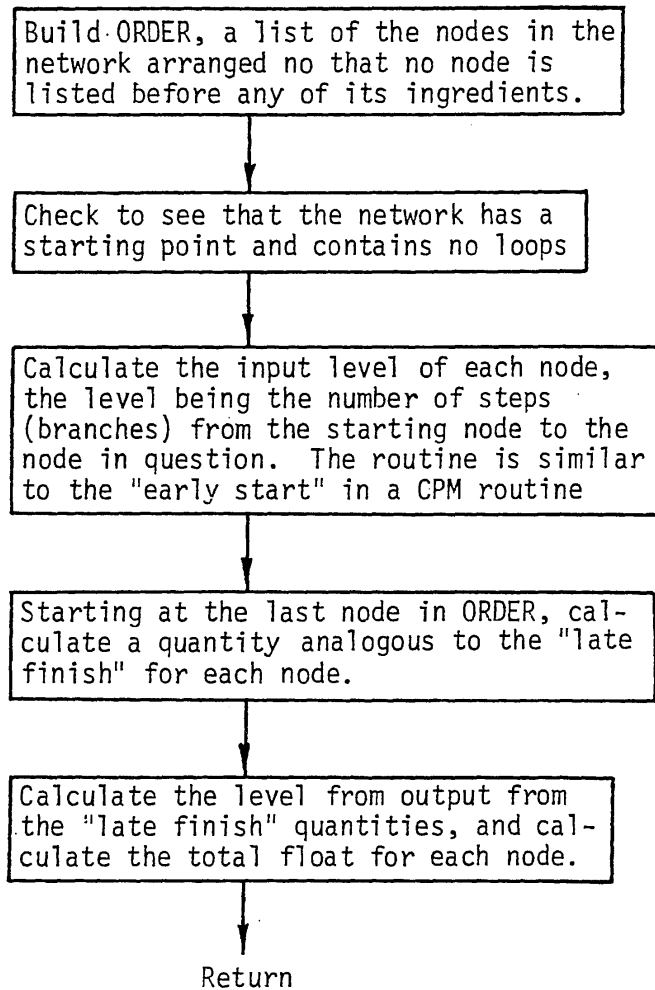


Figure 2.6 Subroutine LEVEL (NETWORK)

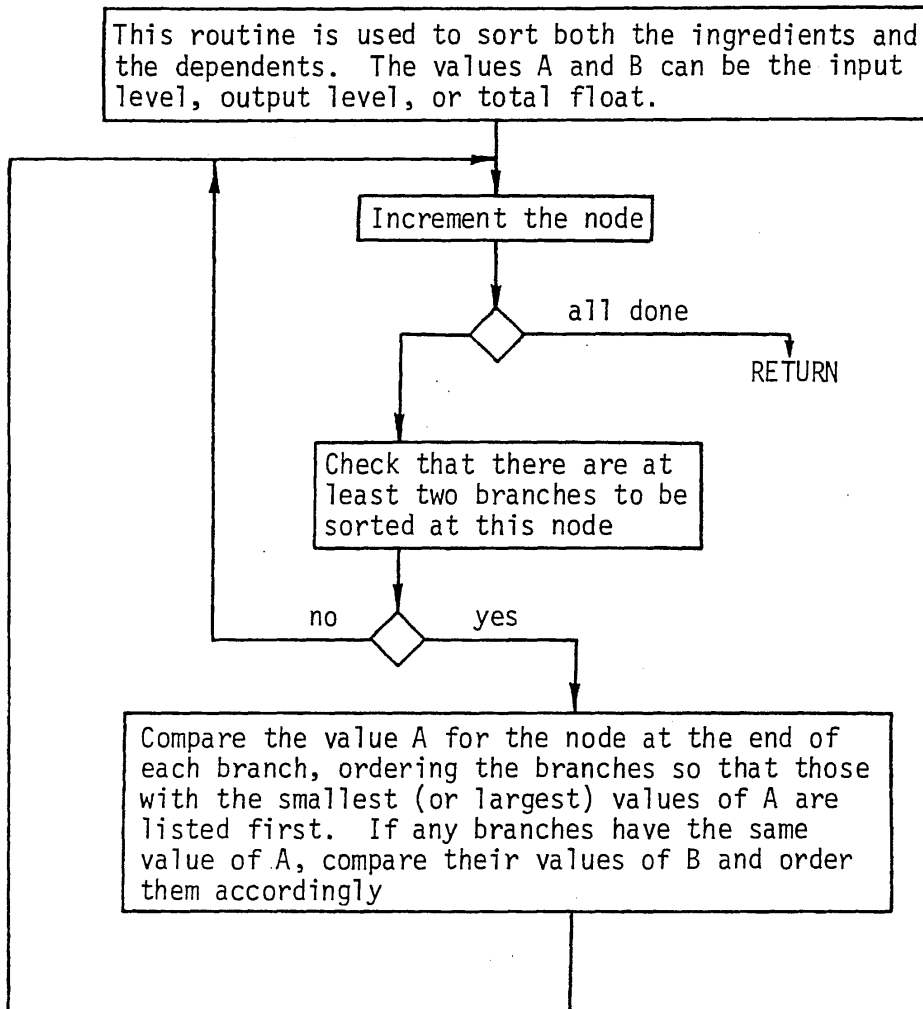


Figure 2.7 Subroutine SORT (NETWORK)

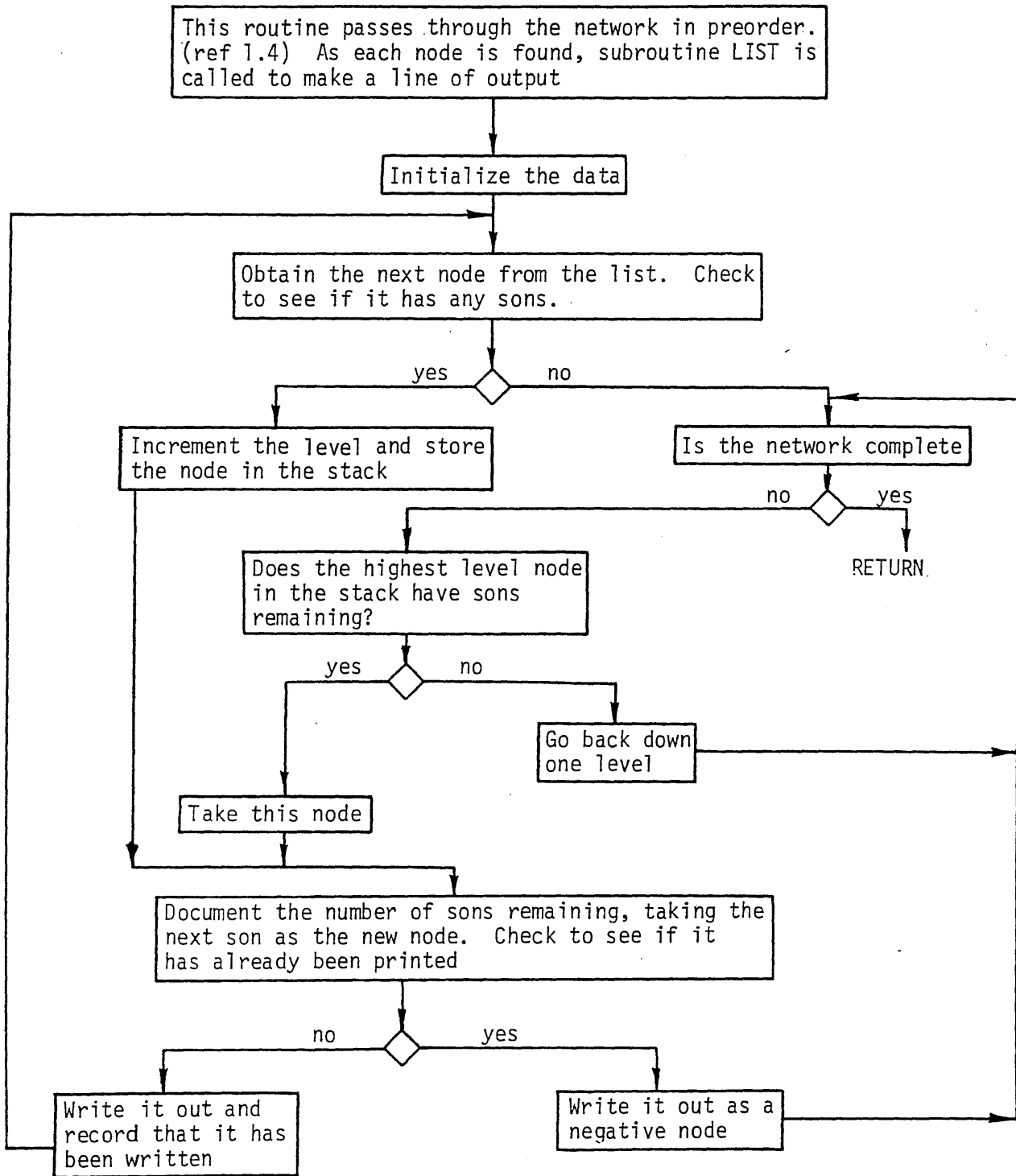


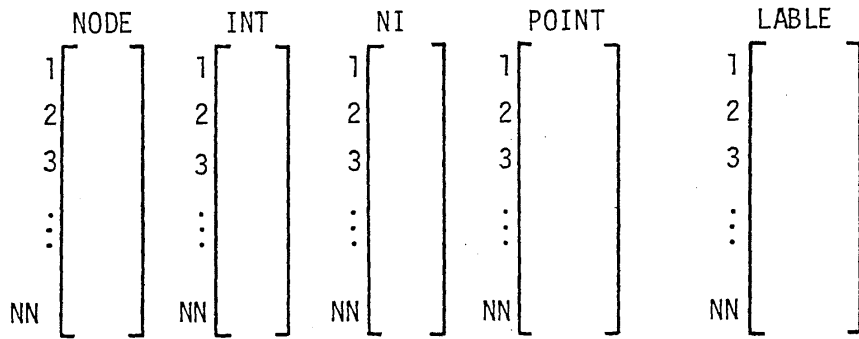
Figure 2.8 Subroutine GLOBAL (NETWORK)

2.3 Data Structure

The figures on the following pages show the principal items of data used in the program. A complete list of all the data items used in the program is in the glossary, section 2.4.

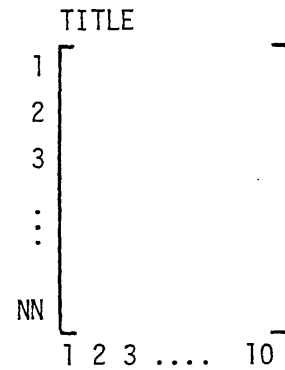
All of the data except the labels and descriptive titles are integer variables. Except for those items in PARCOM, no use is made of a common data structure. All data are passed into subroutines by the use of arguments in the calling statements.

NN = Total number of nodes
 TNI = total number of branches
 L = Length of linked ingredients list



The row number corresponds to the internal number for all arrays above except INT, in which the row number is the external identifying number. The contents are as follows:

- NODE - external number
- INT - internal number
- NI - number of ingredients
- POINT - row number in ingr. that contains the first ingredient
- LABEL - single word label (six characters)
- TITLE - ten word descriptive title



INGR and LINK contains the ingredients in the following manner: Each ingredient is entered into INGR. The row number of the previous ingredient is entered in LINK. If there is no previous ingredient for the node, LINK contains a zero. When revised ingredients are entered, they enter INGR at the bottom, and the old ingredients are lost.

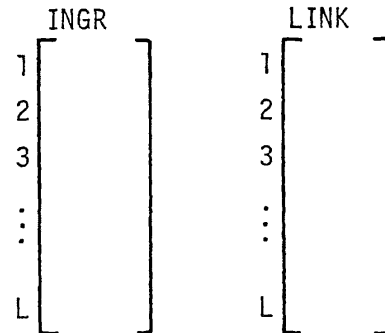
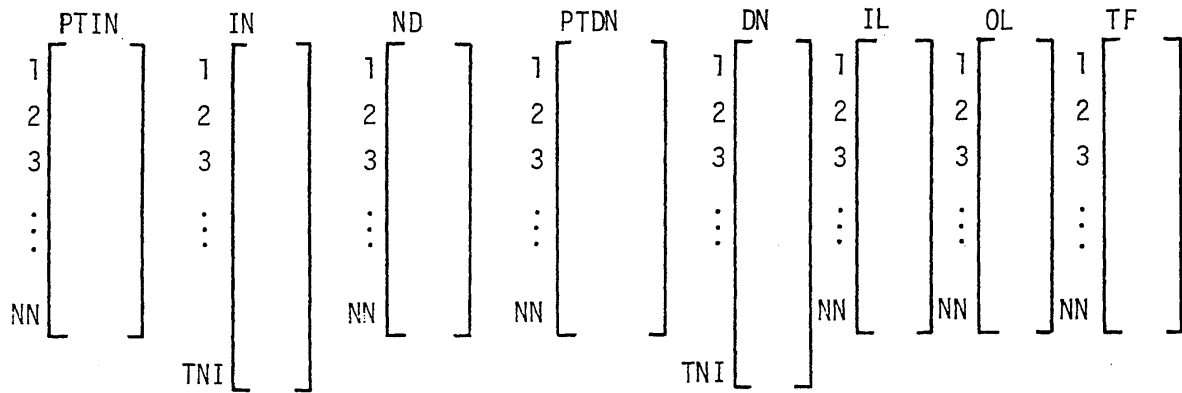


Figure 2.9 Permanent Data (NETWORK)

Contents of the line	Format
NN TNI L LNN	4I6
NODE INT NI POINT LABEL, row 1	4I6, A6
NODE INT NI POINT LABEL, row 2	
:	
NODE INT NI POINT LABEL, row LNN	
INGR LINK, row 1	2I6
INGR LINK, row 2	
:	
INGR LINK, row L	
TITLE row 1	10A6
TITLE row 2	
:	
TITLE row LNN	

Figure 2.10 Permanent Data File (NETWORK)



For all arrays which are NN in length, the row number corresponds to the internal node number. The contents are as follows:

- PTIN - row number of IN where the first ingredient is located.
- IN - the ingredients of all nodes
- ND - the number dependents of each node
- PTDN - row number of DN where the first dependent is located.
- DN - the dependents of all nodes
- IL - the extreme level from input for each node
- OL - the extreme level from output for each node
- TF - the difference in length between the longest path through the network and the longest path through the node.

Figure 2.11 Temporary Data (NETWORK)

100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
2600
2700
2800
2900
3000
3100
3200
3300
3400
3500
3600
3700
3800
3900
4000
4100
4200
4300
4400
4500
4600
4700
4800
4900
5000
5100
5200
5300
5400
5500
5600
5700
5800
5900

2.4 GLOSSARY

SYMBOL : DESCRIPTION

1400 A() : DUMMY NAME FOR TL, OL, OR TF

1500 :

1600 ALPHA1 : VARIABLES USED TO CARRY ALPHABETIC INFORMATION FOR

1700 ALPHA2, : USE IN DESCRIBING THE MANNER OF SORTING

1800 ALPHA3, :

1900 ALPHA4 :

2000 :

2100 ASTER : CARRIES THE ASTERISK SYMBOL

2200 :

2300 B() : DUMMY NAME FOR IL, OL, OR TF

2400 :

2500 BLANK : CARRIES A BLANK SYMBOL. IN SOME SUBROUTINES IT IS A SIX

2600 : CHARACTER WORD, WHILE IN OTHERS IT IS ONLY ONE CHARACTER

2700 :

2800 COLON : THE COLON SYMBOL

2900 :

3000 DASH : A SIX CHARACTER WORD; ALL CARRYING THE DASH SYMBOL

3100 :

3200 DATOUT : A SUBROUTINE

3300 :

3400 DEPEND : A SUBROUTINE

3500 :

3600 DF : THE NUMBER OF THE DATAFILE BEING USED

3700 :

3800 DNC() : VECTOR WHICH LISTS THE DEPENDENTS OF ALL THE NODES

3900 :

4000 EIGHT : THE SYMBOL 8 IN ALPHANERIC FORM

4100 :

4200 END : A LOGICAL FUNCTION IN THE PARSE ROUTINES THAT IS TRUE

4300 : WHEN THE SCANNER IS AT THE END OF A LINE

4400 :

4500 ENTITY() : THE VECTOR WHICH CONTAINS THE MATERIAL SCANNED BY PARSE

4600 :

4700 ERROR : A SUBROUTINE

4800 :

4900 FLO : THE WORD FLOAT (THE SIXTH CHARACTER IS A BLANK)

5000 :

5100 FIVE : THE SYMBOL 5 IN ALPHANERIC FORM

5200 :

5300 FIXED : A LOGICAL FUNCTION IN THE PARSE ROUTINES THAT IS TRUE

5400 : IF THE ITEM SCANNED IS AN INTEGER

5500 :

5600 FOUR : THE SYMBOL 4 IN ALPHANERIC FORM

5700 :

5800 GLOB() : A VECTOR CONTAINING THE NODES LISTED IN PREORDER

5900 :

```

6000 GLOBAL      : A SUBROUTINE
6100             :
6200 HIT()       : A VECTOR THAT IS USED TO RECORD THE OCCURANCE OF A NODE
6300             : IN THE PREORDER LIST
6400             :
6500 I            : A COUNTING INDEX; AS USED IN "INPUT", IT IS THE NUMBER OF
6600             : NODES WHICH HAVE BEEN ENTERED INTO THE SYSTEM
6700             :
6800 II, IJ       : COUNTING INDICES
6900             :
7000 III()        : VECTOR WHICH CONTAINS THE INPUT LEVEL; THE LONGEST
7100             : PATH FROM THE NODE TO THE INPUT NODES
7200             :
7300 INC()         : VECTOR THAT LISTS THE INGREDIENTS OF ALL THE NODES
7400             :
7500 INGIS        : A SUBROUTINE
7600             :
7700 INGR()        : THE LINKED LIST OF INGREDIENTS AS ENTERED
7800             :
7900 INGRFD       : A SUBROUTINE
8000             :
8100 INIT          : A SUBROUTINE
8200             :
8300 INPUT         : A SUBROUTINE
8400             :
8500 INT()         : VECTOR THAT CONTAINS THE INTERNAL NODE NUMBER. THE
8600             : ROW CORRESPONDS TO THE EXTERNAL NUMBER
8700             :
8800 IVALUE        : VARIABLE WHICH CARRIES AN INTEGER NUMBER BACK FROM PARSE
8900             :
9000 IP1, IP2,    : VARIOUS COUNTING INDICES AND TEMPORARY INTEGERS
9100 I1, I2,      :
9200 I4, I9,      :
9300             :
9400 J            : AS USED IN THE INPUT ROUTINE IT IS THE NODE THAT HAS
9500             : INGREDIENTS BEING ENTERED. ELSEWHERE IT IS A
9600             : COUNTING INDEX
9700             :
9800 JI           : THE ROW IN ORDER WITH THE LAST ENTRY
9900             :
10000 JK        : THE LAST ROW IN ORDER THAT HAS HAD IT DEPENDENTS PROCESSED
10100             :
10200 JJ         : NODE WHICH IS HAVING ITS DEPENDENTS PROCESSED FOR ORDER
10300             :
10400 K          : A COUNTING INDEX
10500             :
10600 KEY        : AN INTEGER USED TO TRACE THE PATH INTO SUBROUTINE GLOBAL
10700             :
10800 KI, K1,     : COUNTING INDICES
10900 K2         :
11000             :
11100 KOUNT       : RECORD THE NUMBER OF LINES OF OUTPUT IN THE GRAPHICAL
11200             : TREE PRODUCED IN GLOBAL
11300             :
11400 KOUT        : FILE NUMBER FOR PRINTED OUTPUT. IT ALWAYS IS FILE NO. 1
11500             : IN THE PRESENT PROGRAM
11600             :
11700 L          : USED TO RECORD THE LENGTH OF THE LINKED LIST OF INGREDIENTS.
11800             : IT ALSO IS USED TO RECORD THE CORRENT LEVEL INTO THE
11900             : NETWORK IN THE ROUTINES GLOBAL AND OUTPUT
12000             :
12100 LABEL()     : VECTOR CONTAINING THE ALPHA LABELS (UP TO SIX

```



```

12200      : CHARACTERS) FOR EACH3 NODE
12300      :
12400  LARGE      : THE WORD LARGE (THE SIXTH CHARACTER IS BLANK)
12500      :
12600  LENGTH     : USED IN SUBROUTINE LEVEL TO RECORD THE INPUT LEVEL OF
12700      : NODE NF. WHEN COMPLETED, IT IS THE LENGTH OF THE
12800      : LONGEST PATH THROUGH THE NETWORK
12900      :
13000  LEV()      : DUMMY NAME FOR IL OR OL
13100      :
13200  LEVEL      : A SUBROUTINE
13300      :
13400  L1, L1,     : VARIOUS COUNTING INDICES AND TEMPORARY NODE NUMBERS
13500  L2, LL      :
13600      :
13700  LINE()     : ALPHAMERIC LIST USED TO STORE ONE LINE OF OUTPUT
13800      :
13900  LINEI      : A PARTICULAR SPACE IN LINE()
14000      :
14100  LINK()     : VECTOR WHICH LINKS THE INGREDIENTS OF EACH NODE
14200      : TOGETHER IN THE LIST INGR
14300      :
14400  LIST       : A SUBROUTINE
14500      :
14600  LN         : THE LEVEL OF THE NODE BEING PRINTED
14700      :
14800  LV         : THE WORD LEVEL (THE SIXTH CHARACTER IS BLANK)
14900      :
15000  M         : COUNTS THE NUMBER OF INGREDIENTS AS THEY ARE ENTERED
15100      :
15200  MARK       : INTEGER USED TO TRACE THE PATH INTO THE OUTPUT ROUTINE
15300      :
15400  MATCH      : A LOGICAL FUNCTION IN PARSE THAT IS TRUE IF THE INPUT
15500      : AGREES WITH THE ARGUMENT
15600      :
15700  MINUS      : THE MINUS SYMBOL
15800      :
15900  MODE        : AN INTEGER RETURNED BY PARSE THAT INDICATES THE KIND
16000      : OF INPUT BEING SCANNED
16100      :
16200  MULT1,     : INTEGERS USED TO DIFFERENTIATE SORTING FOR LARGE
16300  MULT2      : DIFFERENCES AND SMALL DIFFERENCES
16400      :
16500  N           : A PARTICULAR NODE NUMBER
16600      :
16700  NCHAR      : THE NUMBER OF CHARACTERS IN THE WORD SCANNED BY PARSE
16800      :
16900  ND()       : VECTOR CONTAINING THE NUMBER OF DEPENDENTS OF
17000      : EACH NODE
17100      :
17200  NEXT       : A LOGICAL FUNCTION IN PARSE
17300      :
17400  NF         : THE FARTHEST NODE ALONG THE NETWORK WHICH HAS
17500      : BEEN PROCESSED
17600      :
17700  NT()       : VECTOR CONTAINING THE NUMBER OF LOCAL INGREDIENTS
17800      : OF EACH NODE
17900      :
18000  NINE       : THE SYMBOL 9 IN ALPHAMERIC FORM
18100      :
18200  NN         : TOTAL NUMBER OF NODES
18300      :

```

18400 NNR() : DUPLICATE OF A DUMMY FOR THE LISTS NI() OR ND()
18500 :
18600 NODE() : VECTOR CONTAINING THE EXTERNAL NUMBERS FOR EACH NODE.
18700 : IT IS ALSO USED TO DENOTE A SPECIFIC ONE OF THESE
18800 : IN SUBROUTINE LIST
18900 :
19000 NOUGHT : THE SYMBOL 0 IN ALPHAMERIC FORM
19100 :
19200 NP() : DUMMY NAME FOR NI() OR ND()
19300 :
19400 NP : COUNTER OF THE PAGE NUMBER FOR OUTPUT
19500 :
19600 NR() : DUMMY NAME FOR NI() OR ND()
19700 :
19800 NUM : USED FOR STORAGE OF THE NUMBER OF THE ARTIFICIAL ROOT
19900 :
20000 NUMR : STORES ONE DIGIT OF A NODE NUMBER
20100 :
20200 NUMBER : A SUBROUTINE
20300 :
20400 NWD : A VALUE RETURNED BY PARSE
20500 :
20600 NWDS : THE NUMBER OF SIX CHARACTER WORDS OCCUPIED BY A TITLE
20700 :
20800 NI : THE UNITS DIGIT OF A NODE NUMBER
20900 :
21000 NIO : THE TENS DIGIT OF A NODE NUMBER
21100 :
21200 NIOO : THE HUNDREDS DIGIT OF A NODE NUMBER
21300 :
21400 OL() : OUTPUT LEVEL; THE LONGEST PATH FROM THE NODE TO OUTPUT
21500 :
21600 OI DATA : A SUBROUTINE
21700 :
21800 ONE : THE SYMBOL 1 IN ALPHAMERIC FORM
21900 :
22000 ORDER() : VECTOR WHICH LISTS THE NODES STARTING WITH THE INPUT
22100 : NODES AND ENDING WITH THE OUTPUT NODES
22200 :
22300 OUTPUT : A SUBROUTINE
22400 :
22500 PARSE : A FAMILY OF SUBROUTINES THAT IS AN INTERPRETIVE
22600 : INPUT SCANNER
22700 :
22800 POINT() : VECTOR THAT POINTS TO THE LIST INGR()
22900 :
23000 PRT : VARIABLE WHICH TURNS THE ECHO PRINT OF THE INPUT
23100 : SCANNER ON OR OFF
23200 :
23300 PTDN() : VECTOR THAT POINTS TO THE LIST OF DEPENDENTS, DN()
23400 :
23500 PTIN() : VECTOR THAT POINTS TO THE LIST OF INGREDIENTS, IN()
23600 :
23700 PTRN() : DUMMY NAME FOR PTDN() OR PTIN()
23800 :
23900 PDI LINE : A SUBROUTINE IN PARSE THAT BRINGS A LINE OF INPUT IN
24000 :
24100 RN() : A DUMMY NAME FOR IN() OR DN()
24200 :
24300 SCAN : A SUBROUTINE IN PARSE
24400 :
24500 SEARCH : A SUBROUTINE

24600 :
24700 SEVEN : THE SYMBOL 7 IN ALPHAMERIC FORM
24800 :
24900 SIX : THE SYMBOL 6 IN ALPHAMERIC FORM
25000 :
25100 SMALL : THE WORD SMALL (THE SIXTH CHARACTER IS BLANK)
2 :
25300 SN() : IN() OR DN() MODIFIED TO CONTAIN THE INGREDIENTS
25400 : OR DEPENDENTS OF THE ARTIFICIAL ROOT NODE
25500 :
25600 SORT : A SUBROUTINE
25700 :
25800 SORTER : A SUBROUTINE
25900 :
26000 STACK() : ARRAY THAT STORES THE NODE THAT IS THE FATHER OF THE
26100 : CURRENT BRANCH FOR EACH LEVEL AND THE NUMBER OF SONS
26200 : THAT IT HAS REMAINING
26300 :
26400 STRING : A LOGICAL FUNCTION IN PARSE THAT IS TRUE WHEN THE
26500 : INPUT IS A STRING
26600 :
26700 SWITCH : A LOGICAL VARIABLE USED TO INDICATE THE PATH INTO
26800 : THE INPUT ROUTINE
26900 :
27000 TR : THE TOTAL NUMBER OF BRANCHES IN THE NETWORK AFTER
27100 : ADDING THOSE TO THE ARTIFICIAL ROOT NODE
27200 :
27300 TEMP : VARIABLE USED AS TEMPORARY STORAGE WHILE SORTING
27400 :
27500 TF() : THE TOTAL FLOAT, THE DIFFERENCE BETWEEN THE LONGEST
27600 : PATH THROUGH THE NETWORK AND THE LONGEST SUCH PATH
27700 : THROUGH A GIVEN NODE
27800 :
27900 THREE : THE SYMBOL 3 IN ALPHAMERIC FORM
28000 :
28100 TITLE() : ARRAY USED TO STORE THE ALPHAMERIC TITLES OF THE NODES
28200 :
28300 TNT : THE TOTAL NUMBER OF BRANCHES IN THE NETWORK
28400 :
28500 TSORT : A LOGICAL VARIABLE THAT IS TRUE IF THE NETWORK HAS
28600 : BEEN SORTED
28700 :
28800 TWO : THE SYMBOL 2 IN ALPHAMERIC FORM
28900 :
29000 VALUE : VARIABLE THAT CARRIES A REAL NUMBER BACK FROM PARSE
29100 :
29200 WDNIT : A SUBROUTINE IN PARSE USED TO SET INITIAL CONDITIONS
29300 :
29400 WORD : A SUBROUTINE
29500 :
29600 X : ARGUMENT OF SEVERAL LOGICAL FUNCTIONS IN PARSE
29700 :
29800 ZERO : A SUBROUTINE

10
20
30
40
50
60
70
80
90

2.5 PROGRAM LISTING

```

100      $ RESET FREE
200      FILE 5=INPUT,UNIT=REMOTE,RECORD=14
300      FILE 6=OUTPUT,UNIT=REMOTE,RECORD=14
400      FILE 1=OUTPUT,UNIT=PRINTER,RECORD=23
500      FILE 2=FREE LIST,UNIT=DISKPACK,RECORD=40,BLOCKING=30
600      FILE 3=DUMMY,UNIT=DISKPACK,RECORD=14,BLOCKING=30
700      FILE 4=DUMMY,UNIT=DISKPACK,RECORD=14,BLOCKING=30,AREA=200*10,SAVE=30
800      $ INCLUDE 'PARSE'
900      C
1000     C      DIMENSION ALL OF THE ARRAYS USED IN THE PROGRAM
1100     C      INTEGER NODE(500), INT(500), INGR(1000), POINT(500), LINK(1000),
1200     1      NI(500), PTIN(500), IN(1000), NO(500), PTDN(500),
1300     2      UN(1000), IL(500), OL(500), TF(500), TNI
1400     C      BOTH THE FOLLOWING ARRAYS CONTAIN ALPHAMERIC WORDS
1500     REAL LABEL(500), TITLE(10,500), ALPHA1, ALPHA2, ALPHA3, ALPHA4,
1600     1      SMALL, LARGE, LV, FLD
1700     COMMON TITLE
1800     C      THE FOLLOWING ARE USED AS LOGIC TRACES
1900     LOGICAL SWITCH, TSURT, UNPACK, FORM
2000     C      THE FOLLOWING WORDS ARE USED IN OUTPUT
2100     DATA SMALL, LARGE, LV, FLD/"SMALL ", "LARGE ",
2200     1      "LEVEL ", "FLDAT "/
2300     C
2400     C      DECLARATIONS FOR USE OF THE INPUT SCANNER
2500     LOGICAL END, FIXED, MATCH, STRING, NEXT
2600     COMMON/SCANNER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
2700     EQUIVALENCE (IVALUE,VALUE)
2800     C      DELETE THE ECHO PRINT OF THE SCANNER
2900     DATA PRI/"OFF"/
3000     C      SET THE LENGTH OF A LINE OF INPUT EQUAL TO 72 SPACES
3100     C      AND THE NUMBER OF BLANKS REQUIRED TO SIGNIFY THE END
3200     C      OF A RECORD EQUAL TO 10
3300     CALL WDINIT(10,72,PRI)
3400     C
3500     C      READ THE OUTPUT UNIT IN
3600     KUOT = 5
3700     WRITE(KUOT,1005)
3800     CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
3900     IF(.NOT. MATCH("P",1)) GO TO 10
4000     KUOT = 1
4100     CALL SETOUT(KUOT)
4200     C
4300     C      SET THE ECHO
4400     WRITE(6,1015)
4500     CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
4600     IF(MATCH("YES",3)) CALL WDINIT(10,72,"ON  ")
4700     C
4800     C      INITIALIZE THE ARRAYS
4900     10 CALL INI(NODE, INT, INGR, POINT, LINK, NI, PTIN, IN, NO,
5000     1      PTDN, UN, IL, OL, TF, LABEL)

```

```
5100 C
5200 C READ THE DATA FILE NAME IN
5300 K = 0
5400 20 WRITE(6,1000)
5500 CALL RDLINE
5600 ENTITY(NRD+1) = 0H.
5700 CLOSE(3)
5800 CHANGE(3,TITLE=ENTITY)
5900 CHANGE(4,TITLE=ENTITY)
6000 C
6100 C CHECK FOR EXISTING DATAFILE
6200 INQUIRE(3, RESIDENT=ONPACK)
6300 IF(.NOT. ONPACK) GO TO 50
6400 C
6500 C EXISTING FILE, DOUBLE CHECK THE NAME
6600 WRITE(6,1010)
6700 CALL RDLINE
6800 IF(MATCH("YES",3)) GO TO 05
6900 GO TO 20
7000 C
7100 C READ NEW NETWORK DATA IN FROM THE TERMINAL
7200 50 SWITCH = .TRUE.
7300 CALL INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL, NI,
7400 1 POINT, INGR, LINK, LNN)
7500 GO TO 70
7600 C
7700 C READ THE OLD NETWORK DATA FROM THE DATAFILE
7800 65 CALL GLDATA(NN, INI, L, NODE, INT, LABEL, NI, POINT,
7900 1 INGR, LINK, LNN)
8000 C
8100 C CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
8200 WRITE(6,1020)
8300 CALL RDLINE
8400 IF(.NOT.MATCH("YES",3)) GO TO 70
8500 SWITCH = .FALSE.
8600 CALL INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL, NI,
8700 1 POINT, INGR, LINK, LNN)
8800 C
8900 C CONSTRUCT THE SIMPLIFIED INGREDIENTS LIST
9000 70 CALL INGRED(NN, INI, NI, POINT, INGR, LINK, PTIN, IN)
9100 C
9200 C CONSTRUCT THE SIMPLIFIED DEPENDENTS LIST
9300 CALL DEPEND(NN, INI, NI, PTIN, IN, ND, PTDN, DN)
9400 C
9500 C CALCULATE THE TOPOLOGIC LEVELS OF THE NETWORK
9600 CALL LEVEL(NN, INI, NI, PTIN, ND, PTDN, DN, IL, OL, TF, LENGTH)
9700 C
9800 C SET THE TRACE INDICATING THE NO SORT HAS BEEN DONE
9900 TSORT = .FALSE.
10000 C
10100 C BEGIN SCANNING FOR OPERATING COMMANDS
10200 C
10300 K = 0
10400 100 WRITE(6,1030)
10500 CALL RDLINE
10600 110 IF(END(X)) GO TO 100
10700 IF(MATCH("SORT",3)) GO TO 120
10800 IF(MATCH("INGRED",3)) GO TO 100
10900 IF(MATCH("DEPEND",3)) GO TO 190
11000 IF(MATCH("WRITE",3)) GO TO 220
11100 IF(MATCH("MODIFY",3)) GO TO 230
11200 IF(MATCH("STOP",3)) STOP
```

```

11300      IF(MATCH("NEXT",3)) GO TO 10
11400      K = K + 1
11500      IF(K.EQ.10) CALL ERROR(1)
11600      GO TO 100
11700      C
11800      C      SORTING DONE HERE. FIND THE PARAMETERS
11900      C
12000      120 TSDRT = .TRUE.
12100      C
12200      C      FIND THE VALUE FOR FIRST PRIORITY SORTING
12300      WRITE(6,1040)
12400      CALL RDLINE
12500      M1 = 1
12600      ALPHA2 = FLU
12700      ALPHA4 = LV
12800      IF(MATCH("FLOAT",3)) GO TO 130
12900      M1 = 2
13000      ALPHA2 = LV
13100      ALPHA4 = FLU
13200      IF(MATCH("LEVEL",3)) GO TO 130
13300      K = K + 1
13400      IF(K.EQ.10) CALL ERROR(1)
13500      GO TO 120
13600      C
13700      C      FIND THE MODE FOR FIRST PRIORITY SORTING
13800      130 WRITE(5,1050)
13900      CALL RDLINE
14000      M2 = 1
14100      ALPHA1 = SMALL
14200      IF(MATCH("SMALL",3)) GO TO 140
14300      M2 = 2
14400      ALPHA1 = LARGE
14500      IF(MATCH("LARGE",3)) GO TO 140
14600      K = K + 1
14700      IF(K.EQ.10) CALL ERROR(1)
14800      GO TO 130
14900      C
15000      C      FIND THE MODE FOR SECOND PRIORITY SORTING
15100      140 WRITE(6,1060)
15200      CALL RDLINE
15300      M3 = 1
15400      ALPHA3 = SMALL
15500      IF(MATCH("SMALL",3)) GO TO 150
15600      M3 = 2
15700      ALPHA3 = LARGE
15800      IF(MATCH("LARGE",3)) GO TO 150
15900      K = K + 1
16000      IF(K.EQ.10) CALL ERROR(1)
16100      GO TO 140
16200      C
16300      C      CALL THE SORTING ROUTINE
16400      150 CALL SORTER(M1, M2, M3, NN, IN1, NI, PI(N, IN, ND, PTD),
16500      1      DN, UL, IL, TF)
16600      K = 0
16700      GO TO 100
16800      C
16900      C      GLOBAL INGREDIENCE DESIRED HERE. FIND THE DESIRED ROOT
17000      C
17100      160 FURM = .FALSE.
17200      IF(MATCH("TITLE",3)) FURM = .TRUE.
17300      WRITE(6,1070)
17400      CALL RDLINE

```

```
17500      IF(MATCH("COMPLETE",3)) GO TO 170
17600      IF(FIXED(X)) GO TO 180
17700      K = K + 1
17800      IF(K.EQ.10) CALL ERROR(1)
17900      GO TO 160
18000      C
18100      C      DO FOR THE COMPLETE NETWORK
18200      170 CALL SEARCH(NN, IN1, ND, NI, PTIN, IN, OL, KOUNT, NODE,
18300      1      LABEL, TSORT, M1, M2, M3, TF, FORM)
18400      CALL OUTPUT(3, KOUI, O, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
18500      1 ALPHA4, LENGTH, FORM)
18600      K = 0
18700      GO TO 100
18800      C
18900      C      DO FOR ONE NODE
19000      180 N = INT(IVALUE)
19100      CALL GLOBAL(N, NN, IN1, NI, PTIN, IN, OL, 1, KOUNT,
19200      1      NODE, LABEL, FORM)
19300      CALL OUTPUT(1, KOUI, IVALUE, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3
19400      1 ALPHA4, LENGTH, FORM)
19500      K = 0
19600      GO TO 100
19700      C
19800      C      GLOBAL DEPENDENCE DESIRED HERE. FIND THE DESIRED ROOT
19900      C
20000      190 FORM = .FALSE.
20100      IF(MATCH("TITLE",3)) FORM = .TRUE.
20200      WRITE(6,1070)
20300      CALL ROLINE
20400      IF(MATCH("COMPLETE",3)) GO TO 200
20500      IF(FIXED(X)) GO TO 210
20600      K = K + 1
20700      IF(K.EQ.10) CALL ERROR(1)
20800      GO TO 190
20900      C
21000      C      DO FOR THE COMPLETE NETWORK
21100      200 CALL SEARCH(NN, IN1, NI, ND, PTIN, DN, IL, KOUNT, NODE,
21200      1      LABEL, TSORT, M1, M2, M3, TF, FORM)
21300      CALL OUTPUT(4, KOUI, O, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
21400      1 ALPHA4, LENGTH, FORM)
21500      K = 0
21600      GO TO 100
21700      C
21800      C      DO FOR ONE NODE
21900      210 N = INT(IVALUE)
22000      CALL GLOBAL(N, NN, IN1, ND, PTIN, DN, IL, 1, KOUNT,
22100      1      NODE, LABEL, FORM)
22200      CALL OUTPUT(2, KOUI, IVALUE, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
22300      1 ALPHA4, LENGTH, FORM)
22400      K = 0
22500      GO TO 100
22600      C
22700      C      WRITE OUT THE COMPLETE DATA
22800      C
22900      220 CALL DATOUT(NN, IN1, NODE, IN1, LABEL, NI, PTIN,
23000      1      IN, ND, PTIN, DN, IL, OL, TF)
23100      WRITE(KOUI,5000)
23200      K = 0
23300      GO TO 100
23400      C
23500      C      MODIFY THE BASIC INPUT DATA
23600      C
```

```
23700      230 SWITCH = .FALSE.
23800      CALL INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL, NI,
23900      1          POINT, INGR, LINK, LNN)
24000      K = 0
24100      C
24200      CALL ZERO(PTIN, IN, ND, PTON, ON, IL, OL, TF)
24300      GO TO 70
24400      C
24500      1000 FORMAT(1H, "ENTER THE DATA FILE NAME", /)
24600      1005 FORMAT(1H, "ENTER P FOR OUTPUT ON THE ON-SITE PRINTER",
24700      1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL", /)
24800      1010 FORMAT(1H, "FILE EXISTS WITH THIS NAME. DO YOU WANT TO USE IT?", /)
24900      1015 FORMAT(1H, "DO YOU WANT TO HAVE THE INPUT ECHOED ON INPUT", /)
25000      1020 FORMAT(1H, "DO YOU WANT TO MODIFY THE EXISTING DATA?", /)
25100      1030 FORMAT(1H, "ENTER A PROGRAM COMMAND", /)
25200      1040 FORMAT(1H, "ENTER THE VALUE FOR FIRST PRIORITY SORTING", /)
25300      1050 FORMAT(1H, "ENTER THE MODE FOR FIRST PRIORITY SORTING", /)
25400      1060 FORMAT(1H, "ENTER THE MODE FOR SECOND PRIORITY SORTING", /)
25500      1070 FORMAT(1H, "ENTER THE ROOT NODE NUMBER",
25600      1          "----OR THE WORD 'COMPLETE'", /)
25700      5000 FORMAT(1H1)
25800      END
25900      C
26000      C - - - - -
26100      C
26200      SUBROUTINE INIT(NODE, INT, INGR, POINT, LINK, NI, PTIN, IN,
26300      1          ND, PTON, ON, IL, OL, TF, LABEL)
26400      INTEGER NODE(1), INT(1), INGR(1), POINT(1), LINK(1), NI(1),
26500      1          PTIN(1), IN(1), ND(1), PTON(1), ON(1), IL(1), OL(1), TF(1)
26600      REAL LABEL(1), TITLE(10,500)
26700      COMMON TITLE
26800      DATA BLANK/6H /
26900      DO 10 I = 1, 500
27000      10 NODE(I) = 0
27100      DO 20 I = 1, 500
27200      20 INT(I) = 0
27300      DO 30 I = 1, 500
27400      30 POINT(I) = 0
27500      DO 40 I = 1, 500
27600      40 LINK(I) = 0
27700      DO 50 I = 1, 500
27800      50 PTIN(I) = 0
27900      DO 60 I = 1, 500
28000      60 ND(I) = 0
28100      DO 70 I = 1, 500
28200      70 PTON(I) = 0
28300      DO 80 I = 1, 500
28400      80 IL(I) = 0
28500      DO 90 I = 1, 500
28600      90 OL(I) = 0
28700      DO 100 I = 1, 500
28800      100 TF(I) = 0
28900      DO 110 I = 1, 500
29000      DO 110 J = 1, 10
29100      110 TITLE(J, I) = BLANK
29200      DO 120 I = 1, 1000
29300      120 INGR(I) = 0
29400      DO 130 I = 1, 1000
29500      130 LINK(I) = 0
29600      DO 140 I = 1, 1000
29700      140 IN(I) = 0
29800      DO 150 I = 1, 1000
```



```
29900      150 DN(I) = 0
30000          RETURN
30100          END
30200      C
30300      C -----
30400      C
30500          SUBROUTINE INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL,
30600      1          NI, POINT, INGR, LINK, LNN)
30700      C
30800      C ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
30900      C INTEGER INI, NODE(1), INT(1), NI(1), POINT(1), INGR(1),
31000      1 LINK(1)
31100      C REAL LABEL(1), TITLE(10,500), BLANK
31200      C COMMON TITLE
31300      C DATA BLANK/6H /
31400      C LOGICAL SWITCH, EXIT
31500      C
31600      C DECLARATIONS FOR THE USE OF THE INPUT SCANNER
31700      C LOGICAL END, FIXED, MATCH, STRING, NEXT
31800      C COMMON/SCANNER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
31900      C EQUIVALENCE (IVALUE,VALUE)
32000      C
32100      C CHECK THE SWITCH FOR NEW DATA OR MODIFICATION OF OLD DATA
32200      C IF(SWITCH) GO TO 10
32300      C
32400      C OLD DATA TO BE MODIFIED
32500      C I = NN
32600      C GO TO 15
32700      C
32800      C NEW DATA, SET INITIAL CONDITIONS
32900      10 L = 0
33000      C I = 0
33100      C LNN = 0
33200      C
33300      C SCAN FOR THE NODE NUMBER
33400      15 K = 0
33500      20 WRITE(6,1020)
33600      C CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
33700      30 IF(FIXED(X)) GO TO 40
33800      C EXIT = .FALSE.
33900      C IF(MATCH("SAVE",3)) GO TO 150
34000      C EXIT = .TRUE.
34100      C IF(MATCH("END",3)) GO TO 150
34200      C INPROPER INPUT DATA
34300      C K = K + 1
34400      C IF(K.EQ.10) CALL ERROR(1)
34500      C WRITE(6,1000)
34600      C CALL ROUTINE
34700      C GO TO 50
34800      C
34900      C HAVE THE NODE NUMBER, CHECK FOR PREVIOUS ENTRY
35000      40 N = IVALUE
35100      C IF(INT(N).NE.0) GO TO 90
35200      C
35300      C NODE HAS NOT BEEN ENTERED BEFORE
35400      C I = I + 1
35500      C IF(LNN .LT. N) LNN = N
35600      C INT(N) = I
35700      C NODE(I) = N
35800      C J = I
35900      C M = 0
36000      C
```

```
36100 C LOOK FOR LABEL, TITLE, AND INGREDIENTS
36200 50 IF(FIXED(X)) GO TO 60
36300 IF(MODE.EQ.3) GO TO 65
36400 IF(MODE.EQ.7) GO TO 70
36500 IF(END(X)) GO TO 55
36600 C IMPROPER INPUT DATA
36700 WRITE(6,1000)
36800 CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
36900 K = K + 1
37000 IF(K.EQ.10) CALL ERROR(1)
37100 GO TO 50
37200 C
37300 C TEST FOR CONTINUATION OF DATA
37400 55 IF(MATCH(" ",1)) GO TO 50
37500 GO TO 30
37600 C
37700 C HERE WE HAVE AN INGREDIENT, PUT IT IN THE LISTS
37800 60 CALL INGLIS(I, J, L, M, IVALUE, INT, NODE, NI, POINT, INGR, LINK)
37900 GO TO 50
38000 C
38100 C HERE WE HAVE A LABEL
38200 65 LABEL(J) = ENTITY(1)
38300 CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
38400 GO TO 50
38500 C
38600 C HERE WE HAVE A DESCRIPTION, PUT IT INTO THE TITLE ARRAY
38700 70 NWDS = (NCHAR + 5)/6
38800 DO 80 IJ = 1, NWDS
38900 80 TITLE(IJ,J) = ENTITY(IJ)
39000 CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
39100 GO TO 50
39200 C
39300 C NODE HAS BEEN ENTERED BEFORE
39400 90 J = INT(N)
39500 C
39600 C SCAN FOR NEW INGREDIENTS, LABELS, OR TITLES
39700 100 IF(FIXED(X)) GO TO 110
39800 IF(MODE.EQ.3) GO TO 125
39900 IF(MODE.EQ.7) GO TO 130
40000 IF(END(X)) GO TO 105
40100 C IMPROPER INPUT DATA
40200 WRITE(6,1000)
40300 CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
40400 K = K + 1
40500 IF(K.EQ.10) CALL ERROR(1)
40600 GO TO 100
40700 C
40800 C CHECK FOR CONTINUATION OF DATA
40900 105 IF(MATCH(" ",1)) GO TO 120
41000 GO TO 30
41100 C
41200 C HERE WE HAVE A REVISED LIST OF INGREDIENTS, CHECK THE FIRST ONE
41300 110 IF(IVALUE.NE.0) GO TO 115
41400 C CHANGE---THE NODE HAS NO INGREDIENTS
41500 NI(J) = 0
41600 POINT(J) = 0
41700 GO TO 100
41800 C ADD TO THE LIST OF INGREDIENTS
41900 115 M = 0
42000 120 CALL INGLIS(I, J, L, M, IVALUE, INT, NODE, NI, POINT, INGR, LINK)
42100 C CHECK FOR MORE INGREDIENTS
42200 IF(.NOT.FIXED(X)) GO TO 100
```

```
42300      GO TO 120
42400      C
42500      C   HERE WE HAVE A NEW LABEL
42600      125 LABEL(J) = ENTITY(I)
42700      CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
42800      GO TO 130
42900      C
43000      C   HERE WE HAVE A NEW DESCRIPTION
43100      130 DO 135 IJ = 1, 10
43200      135 TITLE(IJ,J) = BLANK
43300      NWDS = (NCHAR + 5)/6
43400      DO 140 IJ = 1, NWDS
43500      140 TITLE(IJ,J) = ENTITY(IJ)
43600      CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
43700      GO TO 150
43800      C
43900      C   END OF INPUT DATA, STORE ON DISC
44000      C
44100      150 REWIND 4
44200      NN = 1
44300      TNI = 0
44400      DO 160 II = 1, LNN
44500      160 TNI = TNI + NI(II)
44600      WRITE(4,1010) NN, TNI, L, LNN
44700      DO 170 I = 1, LNN
44800      170 WRITE(4,1010) NODE(I), INT(I), NI(I), POINT(I), LABEL(I)
44900      DO 180 I = 1, L
45000      180 WRITE(4,1010) INGR(I), LINK(I)
45100      DO 190 I = 1, LNN
45200      190 WRITE(4,1030) (TITLE(J,I),J=1,10)
45300      LOCK 4
45400      I = NN
45500      IF(.NOT. EXIT) GO TO 30
45600      C
45700      1000 FORMAT(1H,"INPUT ERROR---RE-ENTER ON A NEW LINE"/)
45800      1010 FORMAT(1H," 416, A0)
45900      1020 FORMAT(1H,"ENTER THE NODE NUMBERS AND ASSOCIATED DATA, ONE NODE
46000      1TU A LINE."/," IF IT IS NECESSARY TO USE TWO LINES FOR THE"
46100      2" INGREDIENTS,"/," ENTER A COMMA AT THE BEGINNING OF THE "
46200      3"SECOND LINE,"/," ENTER 'END' TO SIGNIFY THE END OF THE DATA,"/)
46300      1030 FORMAT(1H,"10A6)
46400      RETURN
46500      END
46600      C
46700      C   - - - - -
46800      C
46900      C   SUBROUTINE INGLIS(I, J, L, M, N, INT, NODE, NI, POINT, INGR, LINK)
47000      C
47100      C   ALL OF THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
47200      C   INTEGER INT(1), NODE(1), NI(1), POINT(1), INGR(1), LINK(1)
47300      C
47400      C   INCREMENT THE COUNTERS
47500      M = M + 1
47600      L = L + 1
47700      NI(J) = M
47800      POINT(J) = L
47900      C
48000      C   CHECK TO SEE IF THIS IS THE FIRST INGREDIENT
48100      IF(M.EQ.1) LINK(L) = 0
48200      IF(M.NE.1) LINK(L) = L - 1
48300      C
48400      C   CHECK TO SEE IF INGREDIENT NODE HAS BEEN ENTERED BEFORE
```

```

48500      IF(INT(N).EQ.0) GO TO 10
48600      C
48700      C   INGREDIENT NODE HAS BEEN ENTERED BEFORE
48800      INGR(L) = INT(N)
48900      GO TO 4)
49000      C
49100      C   INGREDIENT NODE HAS NOT BEEN ENTERED BEFORE
49200      10 I = I + 1
49300      INT(N) = I
49400      NODE(I) = N
49500      INGR(L) = I
49600      20 RETURN
49700      END
49800      C
49900      C   -----
50000      C
50100      C   SUBROUTINE O-DATA(NN, TNI, L, NODE, INT, LABEL, NI,
50200      1      POINT, INGR, LINK, LNN)
50300      C
50400      C   ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
50500      INTEGER TNI, NODE(1), INT(1), NI(1), POINT(1),
50600      1      INGR(1), LINK(1)
50700      REAL LABEL(1), TITLE(10,500)
50800      COMMON TITLE
50900      C
51000      C   FILL THE ARRAYS FROM FILE 3
51100      REWIND 3
51200      READ(3,1000) NN, TNI, L, LNN
51300      IF(LNN .LT. NN) LNN= NN
51400      DO 10 I = 1, LNN
51500      10 READ(3,1010) NODE(I), INT(1), NI(1), POINT(1), LABEL(I)
51600      DO 20 I = 1, L
51700      20 READ(3,1020) INGR(I), LINK(I)
51800      DO 30 I = 1, LNN
51900      30 READ(3,1030) (TITLE(J,I),J=1,10)
52000      REWIND 3
52100      1000 FORMAT(1H , 4I6)
52200      1010 FORMAT(1H , 4I6, A0)
52300      1020 FORMAT(1H , 4I6)
52400      1030 FORMAT(1H , 10A6)
52500      RETURN
52600      END
52700      C
52800      C   -----
52900      C
53000      C   SUBROUTINE ZERO(PTIN, IN, ND, PTDN, DN, IL, JL, TF)
53100      INTEGER PTIN(1), IN(1), ND(1), PTDN(1), DN(1),
53200      1      IL(1), JL(1), IF(1)
53300      DO 10 I = 1, 500
53400      PTIN(I) = 0
53500      ND(I) = 0
53600      PTDN(I) = 0
53700      IL(I) = 0
53800      JL(I) = 0
53900      TF(I) = 0
54000      10 CONTINUE
54100      DO 20 I = 1, 1000
54200      IN(I) = 0
54300      DN(I) = 0
54400      20 CONTINUE
54500      RETURN
54600      END

```

```
54700 C
54800 C -----
54900 C
55000 C   SUBROUTINE INGRES(NN, INI, NI, POINT, INGR, LINK, PTIN, IN)
55100 C
55200 C   ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
55300 C   INTEGER INI, NI(1), POINT(1), INGR(1), LINK(1), PTIN(1), IN(1)
55400 C
55500 C   CONSTRUCT THE POINTER PTIN FROM NI
55600 C   PTIN(1) = 1
55700 C   DO 10 I = 2, NN
55800 C   10 PTIN(I) = PTIN(I-1) + NI(I-1)
55900 C
56000 C   FILL THE SIMPLE INGREDIENT LIST IN
56100 C   K = 1
56200 C   DO 50 I = 1, NN
56300 C   IF (NI(I) = 1) 50, 20, 30
56400 C   20 IN(K) = INGR(POINT(I))
56500 C   K = K + 1
56600 C   GO TO 50
56700 C   30 M = POINT(I)
56800 C   DO 40 J = 1, NI(I)
56900 C   IN(K) = INGR(M)
57000 C   K = K + 1
57100 C   M = LINK(M)
57200 C   40 CONTINUE
57300 C   50 CONTINUE
57400 C   RETURN
57500 C   END
57600 C
57700 C -----
57800 C
57900 C   SUBROUTINE DEPEND(NN, INI, NI, PTIN, IN, ND, PTDN, DN)
58000 C   INTEGER INI, NI(1), PTIN(1), IN(1), ND(1), PTDN(1), DN(1)
58100 C
58200 C   CALCULATE THE NUMBER OF DEPENDENTS
58300 C   DO 10 I = 1, INI
58400 C   10 ND(IN(I)) = ND(IN(I)) + 1
58500 C
58600 C   CONSTRUCT THE POINTER FOR THE DEPENDENT LIST
58700 C   PTDN(1) = 1
58800 C   DO 30 I = 2, NN
58900 C   30 PTDN(I) = PTDN(I-1) + ND(I-1)
59000 C
59100 C   CONSTRUCT THE DEPENDENT LIST
59200 C   DO 50 I = 1, NN
59300 C   IF (NI(I) .EQ. 0) GO TO 50
59400 C   IP1 = PTIN(I)
59500 C   IP2 = IP1 + NI(I) - 1
59600 C   DO 40 J = IP1, IP2
59700 C   DN(PTDN(IN(J))) = 1
59800 C   40 PTDN(IN(J)) = PTDN(IN(J)) + 1
59900 C   50 CONTINUE
60000 C
60100 C   CORRECT THE POINTER FOR THE DEPENDENT LIST
60200 C   DO 60 I = 1, NN
60300 C   60 PTDN(I) = PTDN(I) - ND(I)
60400 C   RETURN
60500 C   END
60600 C
60700 C -----
60800 C
```

```

60900      SUBROUTINE LEVEL(NN, TNI, NI, PTIN, ND, PTDN, DN, IL, OL, TF,
61000      1          LENGTH)
61100      C
61200      C      ALL THE FOLLOWING ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
61300      C      INTEGER TNI, NI(1), PTIN(1), ND(1), PTDN(1), DN(1), IL(1),
61400      1          UL(1), TF(1)
61500      C
61600      C      ORDER IS A LOCAL VARIABLE
61700      C      INTEGER ORDER(500)
61800      C
61900      C      INITIALIZE THE COUNTERS AND ORDER
62000      C      JL = 0
62100      C      JF = 0
62200      C      DO 10 I = 1, 500
62300      10  ORDER(I) = 0
62400      C
62500      C      BUILD AN ORDERED LIST OF THE NETWORK BY LOOKING FOR
62600      C      STARTING NODES (THOSE WITH NO INGREDIENTS)
62700      C      DO 40 I = 1, NN
62800      C      IF(NI(I).NE.0) GO TO 40
62900      C      HERE WE HAVE A NODE WITH NO REMAINING INGREDIENTS
63000      C      PUT IT IN THE LIST
63100      C      NI(I) = -1
63200      C      JL = JL + 1
63300      C      ORDER(JL) = I
63400      C
63500      C      PROCESS THE DEPENDENTS OF THIS NODE BY REDUCING THEIR
63600      C      REMAINING INGREDIENTS BY ONE
63700      20  JF = JF + 1
63800      C      JJ = ORDER(JF)
63900      C      IF(ND(JJ).EQ.0) GO TO 35
64000      C      L1 = PTDN(JJ)
64100      C      L2 = L1 - 1 + ND(JJ)
64200      C      DO 30 LI = L1, L2
64300      C      LL = DN(LI)
64400      C      NI(LL) = NI(LL) - 1
64500      C
64600      C      CHECK FOR MORE NODES WITH NO REMAINING INGREDIENTS
64700      C      AMONG THESE DEPENDENTS
64800      C      IF(NI(LL).NE.0) GO TO 30
64900      C      HERE WE HAVE A NODE WITH NO REMAINING INGREDIENTS
65000      C      PUT IT IN THE LIST
65100      C      NI(LL) = -1
65200      C      JL = JL + 1
65300      C      ORDER(JL) = LL
65400      30  CONTINUE
65500      C      CHECK TO SEE IF ALL THE NODES NOW IN "ORDER" HAVE
65600      C      HAD THEIR DEPENDENTS PROCESSED
65700      C      35  IF(JF.LT.JL) GO TO 20
65800      C      40  CONTINUE
65900      C
66000      C      CHECK NETWORK FOR ERRORS
66100      C      IF(JL.EQ.0) CALL ERROR(2)
66200      C      IF(JL.NE.NN) CALL ERROR(3)
66300      C
66400      C      CORRECT THE VALUES OF NI
66500      C      J = NN - 1
66600      C      DO 50 I = 1, J
66700      50  NI(I) = PTIN(I + 1) - PTIN(I)
66800      C      NI(NN) = TNI - PTIN(NN) + 1
66900      C
67000      C      CALCULATE THE EXTREME DISTANCE FROM INPUT FOR EACH NODE---

```

```

67100 C THIS IS SIMILAR TO THE EARLY START IN A CPM ROUTINE
67200 NF = ORDER(I)
67300 LENGTH = 0
67400 DO 30 I = 1, NN
67500 J = ORDER(I)
67600 IF(ND(J).EQ.0) GO TO 70
67700 L1 = PTDN(J)
67800 L2 = L1 + ND(J) - 1
67900 DO 50 LI = L1, L2
68000 M = DN(LI)
68100 IF(IL(M).LT.(IL(J) + 1)) IL(M) = IL(J) + 1
68200 60 CONTINUE
68300 70 IF(LENGTH.GE.(IL(J) + 1)) GO TO 80
68400 NF = J
68500 LENGTH = IL(J) + 1
68600 80 CONTINUE
68700 C
68800 C CALCULATE THE EXTREME DISTANCE FROM OUTPUT FOR EACH NODE---
68900 C THIS IS A MODIFICATION OF THE LATE FINISH IN A CPM ROUTINE
69000 K = NN
69100 90 J = ORDER(K)
69200 OL(J) = LENGTH
69300 IF(ND(J).EQ.0) GO TO 110
69400 L1 = PTDN(J)
69500 L2 = L1 + ND(J) - 1
69600 DO 100 LI = L1, L2
69700 M = DN(LI)
69800 IF((OL(M) - 1).LT.OL(J)) OL(J) = OL(M) - 1
69900 100 CONTINUE
70000 110 K = K - 1
70100 IF(K.NE.0) GO TO 90
70200 C
70300 C CALCULATE THE OUTPUT LEVEL AND THE TOTAL FLOAT
70400 DO 120 J = 1, NN
70500 OL(J) = LENGTH - OL(J)
70600 120 TF(J) = LENGTH - OL(J) - IL(J) - 1
70700 RETURN
70800 END
70900 C
71000 C -----
71100 C
71200 C SUBROUTINE SORTER(M1, M2, M3, NN, INI, NI, PTIN, IN, NO, PTDN, DN,
71300 C OL, IL, TF)
71400 C
71500 C ALL OF THE ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
71600 C INTEGER INI, NI(1), PTIN(1), IN(1), NO(1), PTDN(1),
71700 C I DN(1), OL(1), IL(1), TF(1)
71800 C
71900 C DETERMINE THE ORDER OF SORTING
72000 C GO TO (10, 20) M1
72100 C
72200 C SORTING BY FLOAT, THEN BY LEVEL
72300 C SORT THE INGREDIENTS
72400 C 10 CALL SORT(NN, INI, NI, PTIN, IN, TF, OL, M2, M3)
72500 C SORT THE DEPENDENTS
72600 C CALL SORT(NN, INI, NO, PTDN, DN, TF, IL, M2, M3)
72700 C RETURN
72800 C
72900 C SORTING BY LEVEL, THEN BY FLOAT
73000 C SORT THE INGREDIENTS
73100 C 20 CALL SORT(NN, INI, NI, PTIN, IN, OL, TF, M2, M3)
73200 C SORT THE DEPENDENTS

```

```

73300      CALL SURT(NN, TNI, ND, PTRN, UN, IL, TF, M2, M3)
73400      RETURN
73500      END
73600      C
73700      C - - - - -
73800      C
73900      SUBROUTINE SURT(NN, TNI, NR, PTRN, RN, A, B, M2, M3)
74000      C
74100      C ALL OF THE ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
74200      C INTEGER TNI, TEMP, NR(1), PTRN(1), RN(1), A(1), B(1)
74300      C
74400      C SET THE MULTIPLIERS FOR NEAREST AND FARTHEST MODES OF SORTING
74500      C MULT1 = 1
74600      C MULT2 = 1
74700      C IF(42.EQ.2) MULT1 = -1
74800      C IF(43.EQ.2) MULT2 = -1
74900      C
75000      C ORDER THE BRANCHES AT EACH NODE BY A AND B
75100      C DO 30 I = 1, NN
75200      C
75300      C CHECK FOR MULTIPLE BRANCHES
75400      C IF(NR(I).LT.2) GO TO 30
75500      C N = NR(I) - 1
75600      C DO 30 JJ = 1, N
75700      C J = JJ - 1
75800      C DO 30 K = JJ, N
75900      C
76000      C SORT FIRST BY A
76100      C IF(MULT1*(A(RN(PTRN(I)+J)) - A(RN(PTRN(I)+K)))) 30, 20, 10
76200      C
76300      C SWITCH BRANCHES J AND K
76400      C 10 TEMP = RN(PTRN(I)+J)
76500      C RN(PTRN(I)+J) = RN(PTRN(I)+K)
76600      C RN(PTRN(I)+K) = TEMP
76700      C GO TO 30
76800      C
76900      C SORT ANY TIES BY B
77000      C 20 IF(MULT2*(B(RN(PTRN(I)+J)) - B(RN(PTRN(I)+K)))) 30, 30, 10
77100      C 30 CONTINUE
77200      C RETURN
77300      C END
77400      C
77500      C - - - - -
77600      C
77700      C SUBROUTINE SEARCH(NN, TNI, NP, NR, PTRN, RN, LEV, KOJNT,
77800      C 1 NODE, LABEL, TSORT, M1, M2, M3, TF, FORM)
77900      C
78000      C THE FOLLOWING ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
78100      C INTEGER TNI, NP(1), NR(1), PTRN(1), RN(1), LEV(1),
78200      C 1 NODE(1), TF(1)
78300      C REAL LABEL(1)
78400      C LOGICAL TSORT, FORM
78500      C
78600      C SN DOES NOT APPEAR IN THE MAIN PROGRAM AND IS DIMENSIONED HERE
78700      C INTEGER SN(500), TB
78800      C
78900      C CONNECT ALL THE STARTING NODES TO THE ARTIFICIAL
79000      C NODE N AND MODIFY NR, PTRN, AND RN ACCORDINGLY
79100      C N = NN + 1
79200      C NR(N) = 0
79300      C I = 0
79400      C DO 10 J = 1, NN

```



```

79500      IF(NP(J).NE.0) GO TO 10
79600      I = I + 1
79700      SN(I) = J
79800      NR(N) = NR(N) + 1
79900      10 CONTINUE
80000      TB = TNI + I
80100      PTRN(N) = TNI + 1
80200      DO 20 J = PTRN(N), TB
80300      I = J - TNI
80400      20 RN(J) = SN(I)
80500      C
80600      C      CHECK TO SEE IF THE NEW BRANCHES SHOULD BE SORTED
80700      IF(.NOT.ISORT) GO TO 50
80800      C
80900      C      SORT IS DESIRED, DETERMINE THE ORDER
81000      GO TO (30,40) M1
81100      C      SORT BY FLOAT, THEN BY LEVEL
81200      30 CALL SORT(1, I, NR(N), PTRN(N), RN, TF, (LEV, M2, M3)
81300      GO TO 50
81400      C      SORT BY LEVEL, THEN BY FLOAT
81500      40 CALL SORT(1, I, NR(N), PTRN(N), RN, LEV, TF, M2, M3)
81600      C
81700      C      FIND THE GLOBAL INGREDIENCE (DEPENDENCE) OF NODE N
81800      50 NUM = N
81900      CALL GLOBAL(N, N, I, NR, PTRN, RN, LEV, 0, KOUNT, NODE, LABEL,
82000      1          FORM)
82100      C
82200      C      CORRECT NR, PTRN, AND RN TO THEIR ORIGINAL VALUES
82300      NR(NUM) = 0
82400      DO 60 J = PTRN(NUM), TB
82500      60 RN(J) = 0
82600      PTRN(NUM) = 0
82700      RETURN
82800      END
82900      C
83000      C      -----
83100      C
83200      C      SUBROUTINE GLOBAL(N, NN, TNI, NR, PTRN, RN, LEV, KEY, KOUNT,
83300      1          NODE, LABEL, FORM)
83400      C
83500      C      THESE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
83600      INTEGER TNI, NODE(1), NR(1), PTRN(1), RN(1), LEV(1)
83700      REAL LABEL(1)
83800      LOGICAL FORM
83900      C
84000      C      GLOB, HIT AND STACK ARE VARIABLES WHICH DO NOT APPEAR
84100      C      IN THE MAIN PROGRAM AND ARE DIMENSIONED HERE
84200      INTEGER GLOB(1000), HIT(500), STACK(20,2)
84300      C
84400      C      NR AND PTRN ARE BOTH DESTROYED IN THE ROUTINE
84500      C      SO A DUPLICATE OF NR IS PRODUCED HERE
84600      INTEGER NNR(500)
84700      DO 10 I = 1, NN
84800      10 NNR(I) = NR(I)
84900      C
85000      C      INITIALIZE GLOB, HIT, AND STACK
85100      DO 20 I = 1, 1000
85200      20 GLOB(I) = 0
85300      DO 30 I = 1, NN
85400      30 HIT(I) = 0
85500      DO 40 I = 1, 20
85600      40 STACK(I,1) = 0

```

```

85700      40 STACK(I,2) = 0
85800      C
85900      C      BEGIN PROCESSING THE BRANCHES OF THE NETWORK IN PREORDER
86000      M = 1
86100      L = 0
86200      REWIND 2
86300      KUUNT = 0
86400      C
86500      C      INSERT THE ROOT NODE INTO THE OUTPUT LIST
86600      C      UNLESS IT IS IMAGINARY
86700      IF(KEY.EQ.1) CALL LIST(N, L, STACK, LEV, NNR(N), NODE(N),
86800      1 LABEL(N), FORM)
86900      IF(KEY.EQ.1) KUUNT = 1
87000      50 GLOB(M) = N
87100      IF(NR(N).EQ.0) GO TO 60
87200      L = L + 1
87300      STACK(L,1) = N
87400      60 NR(N) = NR(N) + 1
87500      STACK(L,2) = NR(N)
87600      M = M + 1
87700      J = RN(PTRN(N))
87800      PTRN(N) = PTRN(N) + 1
87900      IF(HIT(J).NE.0) GO TO 70
88000      HIT(J) = 1
88100      N = J
88200      CALL LIST(N, L, STACK, LEV, NNR(N), NODE(N), LABEL(N), FORM)
88300      KUUNT = KUUNT + 1
88400      GO TO 50
88500      70 GLOB(M) = J
88600      CALL LIST(GLOB(M), L, STACK, LEV, NNR(J), NODE(J), LABEL(J), FORM)
88700      KUUNT = KUUNT + 1
88800      80 IF(L.EQ.0) GO TO 100
88900      IF(STACK(L,2).EQ.0) GO TO 90
89000      N = STACK(L,1)
89100      GO TO 60
89200      90 L = L - 1
89300      GO TO 50
89400      C
89500      C      CORRECT THE VALUES OF NR AND PTRN
89600      100 PTRN(1) = 1
89700      NR(1) = NNR(1)
89800      DO 110 I = 2, NN
89900      NR(I) = NNR(I)
90000      110 PTRN(I) = PTRN(I - 1) + NNR(I - 1)
90100      END FILE 2
90200      RETURN
90300      END
90400      C
90500      C      - - - - -
90600      C
90700      SUBROUTINE LIST(N, L, STACK, LEV, NNR, NODE, LABEL, FORM)
90800      C
90900      C      STACK AND LEV ARE DIMENSIONED IN THE CALLING PROGRAM
91000      C      INTEGER STACK(20,2), LEV(1)
91100      C
91200      C      LINE IS THE VARIABLE WHICH CONTAINS A LINE OF OUTPUT
91300      C      REAL LINE(234), MINJS, LABEL
91400      C      DATA BLANK, COLON, DOT, MINUS, ASTER/1H .1H.:1H.:1H-.1H*/
91500      C      LOGICAL FORM
91600      C
91700      C      INITIALIZE THE LINE TO ALL BLANKS
91800      C      DO 10 I = 1, 234

```

```

91900      10 LINE(I) = BLANK
92000      IF(L.EQ.0) GO TO 25
92100      C
92200      C      PUT COLONS BELOW ALL "FATHERS" OF THE NODE
92300      DO 20 I = 1, L
92400      J = LEV(STACK(I,1))
92500      IF(STACK(I,2).EQ.0 .AND. I.NE.L) GO TO 20
92600      LINE(10*J + 4) = COLON
92700      20 CONTINUE
92800      C
92900      C      PUT DOTS FROM THE LAST COLON TO THE NODE
93000      25 LN = LEV(ABS(N))
93100      I1 = 10*J + 5
93200      I2 = 10*LN + 3
93300      IF(LN.EQ.0) GO TO 35
93400      DO 30 I = I1, I2
93500      30 LINE(I) = DOT
93600      C
93700      C      BREAK THE EXTERNAL NUMBER OF THE NODE INTO DIGITS AND ENTER
93800      C      THEM IN THE APPROPRIATE SPACES IN THE LINE
93900      35 N100 = ABS(NODE)/100
94000      N10 = ABS(NODE)/10 - 10*N100
94100      N1 = ABS(NODE) - 10*N10 - 100*N100
94200      IF(N1.EQ.0) N1=10
94300      IF(N10.NE.0) GO TO 40
94400      IF(N100.NE.0) GO TO 50
94500      GO TO 60
94600      40 CALL NUMBER(LINE(I2-1),N100)
94700      IF(N10.EQ.0) N10 = 10
94800      50 CALL NUMBER(LINE(I2),N10)
94900      60 CALL NUMBER(LINE(I2+1),N1)
95000      C
95100      C      ENTER A NEGATIVE SIGN IF THE NODE HAS BEEN PRINTED BEFORE
95200      IF(N.LT.0 .AND. N100.NE.0 .AND. N10.NE.0) LINE(I2-2) = MINUS
95300      IF(N.LT.0 .AND. N100.EQ.0 .AND. N10.NE.0) LINE(I2-1) = MINUS
95400      IF(N.LT.0 .AND. N100.EQ.0 .AND. N10.EQ.0) LINE(I2) = MINUS
95500      IF(N.LT.0 .AND. NNR.NE.0) LINE(I2+2) = ASTER
95600      C
95700      C      PUT THE LABEL OR TITLE OF THE NODE IN THE LINE
95800      IF(.NOT. FORM) CALL WORD(LABEL, LINE, I2)
95900      IF(FORM) CALL WORD1(LINE, I2, ABS(N))
96000      C
96100      C      WRITE THE LINE ON THE DISK
96200      WRITE(2,70) LINE
96300      70 FORMAT(1H,234A1)
96400      RETURN
96500      END
96600      C
96700      C - - - - -
96800      C
96900      SUBROUTINE NUMBER(LINEI, NUMB)
97000      C
97100      C      THIS ROUTINE IS USED TO ENTER THE PROPER DIGIT IN THE
97200      C      PROPER SPACE IN THE LINE
97300      REAL LINEI, NUMB
97400      DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,NOUGHT/
97500      1,"1","2","3","4","5","6","7","8","9","0"/
97600      GO TO(1,2,3,4,5,6,7,8,9,10) NUMB
97700      1 LINEI = ONE
97800      RETURN
97900      2 LINEI = TWO
98000      RETURN

```

```
98100      3 LINEI = THREE
98200      RETURN
98300      4 LINEI = FOUR
98400      RETURN
98500      5 LINEI = FIVE
98600      RETURN
98700      6 LINEI = SIX
98800      RETURN
98900      7 LINEI = SEVEN
99000      RETURN
99100      8 LINEI = EIGHT
99200      RETURN
99300      9 LINEI = NINE
99400      RETURN
99500     10 LINEI = NUGHT
99600      RETURN
99700      END
99800      C
99900      C - - - - -
00000      C
00100      SUBROUTINE WRD(LABEL, LINE, IZ)
00200      C
00300      C ALL ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
00400      REAL LABEL, LINE(1)
00500      C
00600      C SET THE COLUMNS IN WHICH THE LABEL IS TO BE PRINTED
00700      I4 = IZ + 4
00800      I9 = IZ + 9
00900      C
01000     C CONVERT THE LABEL FROM A WORD TO SIX CHARACTERS
01100      WRITE(2,100) LABEL
01200      BACKSPACE 2
01300      READ(2,110) (LINE(I), I=I4, I9)
01400      BACKSPACE 2
01500      C
01600      100 FORMAT(A6)
01700      110 FORMAT(6A1)
01800      RETURN
01900      END
02000      C
02100      C - - - - -
02200      C
02300      SUBROUTINE WRD1(LINE, IZ, N)
02400      C
02500      C ALL ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
02600      REAL TITLE(10,500), LINE(1)
02700      COMMON TITLE
02800      C
02900      C SET THE COLUMNS IN WHICH THE TITLE IS TO BE PRINTED
03000      I4 = IZ + 4
03100      IX = IZ + 63
03200      C
03300     C CONVERT FROM WORDS TO CHARACTERS
03400      WRITE(2,100) (TITLE(I,N), I=1,10)
03500      BACKSPACE 2
03600      READ(2,110) (LINE(I), I=I4, IX)
03700      BACKSPACE 2
03800      C
03900      100 FORMAT(10A6)
04000      110 FORMAT(60A1)
04100      RETURN
04200      END
```

```
04300 C
04400 C
04500 C
04600 SUBROUTINE OUTPUT(MARK, KOUT, N, KOUNT, TSORT, ALPHA1, ALPHA2,
04700 1 ALPHA3, ALPHA4, LENGTH, FORM)
04800 REAL LINE(39), ALPHA1, ALPHA2, ALPHA3, ALPHA4
04900 LOGICAL ISORT, FORM
05000 C
05100 C SET THE LIMITS FOR THE WIDTH OF OUTPUT
05200 I1 = 0
05300 I2 = LENGTH - 1
05400 I3 = 1
05500 NL = 12
05600 NW = 22
05700 IF(KOUT .EQ. 6) NL = 7
05800 IF(KOUT .EQ. 6) NW = 13
05900 I4 = NW
06000 IX = 13
06100 IF(FORM) IX = 67
06200 NP = 1 + (10*(I2) + IX)/(6*NW)
06300 IF(NP .EQ. 1) GO TO 5
06400 C
06500 C RESET THE WIDTH LIMITS FOR A PARTITIONED OUTPUT
06600 I2 = NL
06700 J = 0
06800 2 I1 = I1 + J*NL
06900 I2 = I2 + J*NL
07000 I3 = I3 + J*NW
07100 I4 = I4 + J*NW
07200 C BEGIN THE OUTPUT
07300 5 REWIND 2
07400 C
07500 C WRITE THE PROPER PAGE HEADING
07600 10 IF(TSORT) GO TO 12
07700 WRITE(KOUT, 700)
07800 GO TO 15
07900 12 WRITE(KOUT, 990) ALPHA1, ALPHA2, ALPHA3, ALPHA4
08000 16 GO TO (20, 30, 40, 50) MARK
08100 20 WRITE(KOUT, 1000) N
08200 GO TO 60
08300 30 WRITE(KOUT, 1010) N
08400 GO TO 40
08500 40 WRITE(KOUT, 1020)
08600 GO TO 50
08700 50 WRITE(KOUT, 1030)
08800 GO TO 70
08900 C
09000 C WRITE THE SECOND AND THIRD LINES OF THE PAGE HEADING
09100 60 WRITE(KOUT, 1040)
09200 GO TO 80
09300 70 WRITE(KOUT, 1050)
09400 80 WRITE(KOUT, 1060) (I, I=I1, I2)
09500 C
09600 C READ IN A LINE OF OUTPUT FROM THE DISK AND PRINT IT OUT
09700 90 READ(2, 1070) (LINE(I), I=1, I4)
09800 WRITE(KOUT, 1090)
09900 WRITE(KOUT, 1030) (LINE(I), I=I3, I4)
10000 C
10100 C DECREMENT THE TOTAL NUMBER OF LINES AND CHECK
10200 KOUNT = KOUNT - 1
10300 IF(KOUNT.GT.0) GO TO 90
10400 C
```

```
10500 C CHECK FOR MORE PAGES
10600 J = J + 1
10700 NP = NP - 1
10800 IF(NP) 100, 100, 2
10900 C
11000 100 RETURN
11100 C
11200 980 FORMAT(1H1 "UNSORTED")
11300 990 FORMAT(1H1 "SORTED FIRST BY ", 2A6, "AND THEN BY ", 2A6)
11400 1000 FORMAT(1H0,"GLOBAL INGREDIENCE OF NODE",I3)
11500 1010 FORMAT(1H0,"GLOBAL DEPENDENCE OF NODE",I3)
11600 1020 FORMAT(1H0,"GLOBAL INGREDIENCE OF COMPLETE NETWORK")
11700 1030 FORMAT(1H0,"GLOBAL DEPENDENCE OF COMPLETE NETWORK")
11800 1040 FORMAT(1H0,"EXTREME LEVEL FROM OUTPUT")
11900 1050 FORMAT(1H0,"EXTREME LEVEL FROM INPUT")
12000 1060 FORMAT(1H0, I4, 10(5X,I2)/)
12100 1070 FORMAT(39A6)
12200 1080 FORMAT(1H+,39A6)
12300 1090 FORMAT(5H )
12400 END
12500 C
12600 C - - - - -
12700 C
12800 SUBROUTINE DATOUT(NN, TNI, NODE, INT, LABEL, NI,
12900 1 PTIN, IN, ND, PTDN, DN, IL, UL, TF)
13000 C
13100 C ALL OF THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM EXCEPT DASH
13200 C INTEGER TNI, NODE(1), INT(1), NI(1), PTIN(1), IN(1), ND(1),
13300 1 PTDN(1), DN(1), IL(1), UL(1), TF(1)
13400 REAL LABEL(1), TITLE(10,500), DASH(22)
13500 COMMON TITLE
13600 DATA DASH(1)/6H-----/
13700 DO 10 J = 2, 22
13800 10 DASH(J) = DASH(1)
13900 C
14000 C WRITE THE PAGE HEADING
14100 I = 0
14200 20 WRITE(1,1000) DASH
14300 C
14400 C INCREMENT THE NODE AND WRITE OUT A LINE OF DATA
14500 30 I = I + 1
14600 IF(I.GT.NN) RETURN
14700 N = INT(1)
14800 WRITE(1,1010) I, LABEL(N), (TITLE(J,N),J=1,10),
14900 1 IL(N), UL(N), TF(N)
15000 C
15100 C FIND THE NUMBER OF INGREDIENTS AND DEPENDENTS
15200 K = NI(N)
15300 L = ND(N)
15400 C
15500 C DETERMINE THE NUMBER OF LINES REQUIRED
15600 KL = (K+3)/4
15700 LL = (L+3)/4
15800 IF((KL.GT.1).OR.(LL.GT.1)) GO TO 50
15900 C
16000 C ONLY ONE LINE REQUIRED
16100 IF(K.EQ.0) GO TO 35
16200 K1 = PTIN(N)
16300 K2 = K1 + K - 1
16400 WRITE(1,1020) (NODE(IN(J)),J=K1,K2)
16500 35 IF(L.EQ.0) GO TO 30
16600 L1 = PTDN(N)
```

```
16700      L2 = L1 + L - 1
16800      WRITE(1,1030) (NODE(DN(J)),J=L1,L2)
16900      GO TO 30
17000  C
17100  C      MORE THAN ONE LINE REQUIRED, PROCESS ONE LINE AT A TIME
17200      50 M = 0
17300      60 K1 = PTIN(N) + 4*M
17400      IF(KL=1) 100, 80, 70
17500      70 K2 = K1 + 3
17600      GO TO 90
17700      80 K2 = PTIN(N) + K - 1
17800  C      WRITE THE INGREDIENTS
17900      90 WRITE(1,1020) (NODE(IN(J)),J=K1,K2)
18000      100 L1 = PTDN(N) + 4*M
18100      IF(LL=1) 140, 120, 110
18200      110 L2 = L1 + 3
18300      GO TO 130
18400      120 L2 = PTDN(N) + L - 1
18500  C      WRITE THE DEPENDENTS
18600      130 WRITE(1,1030) (NODE(DN(J)),J=L1,L2)
18700  C
18800  C      CHECK TO SEE IF MORE LINES ARE REQUIRED
18900      140 IF((KL.LE.1).AND.(LL.LE.1)) GO TO 30
19000  C
19100  C      INCREMENT THE COUNTERS
19200      M = M + 1
19300      KL = KL - 1
19400      LL = LL - 1
19500      WRITE(1,1040)
19600      GO TO 60
19700  C
19800      1000 FORMAT(1H1,"DATA DATA",I115,"INPUT OUTPUT TOTAL",/
19900      1      " NO. LABEL", 19X,"DATA DESCRIPTION",
20000      2      T/8,"INGREDIENTS", 9X,"DEPENDENTS", 7X,
20100      3      "LEVEL LEVEL FLOAT",/ 22A6)
20200      1010 FORMAT(1H ,I3, 3X, A6, 3X, 10A0, 41X, 2(I2, 4X), I2)
20300      1020 FORMAT(1H+, (5X, 414)
20400      1030 FORMAT(1H+, 95X, 414)
20500      1040 FORMAT(1H )
20600      END
20700  C
20800  C - - - - -
20900  C
21000      SUBROUTINE ERROR(N)
21100      GO TO (1, 2, 3) N
21200  C
21300      1 WRITE(6,101)
21400      STOP
21500  C
21600      2 WRITE(6,102)
21700      STOP
21800  C
21900      3 WRITE(6,103)
22000      STOP
22100  C
22200      101 FORMAT(1H ,"PROGRAM STOPPED---INCORRECT INPUT")
22300      102 FORMAT(1H ,"PROGRAM STOPPED---NO STARTING NODE IN NETWORK")
22400      103 FORMAT(1H ,"PROGRAM STOPPED---NETWORK CONTAINS A LOOP")
22500      RETURN
22600      END
```

Chapter Three

OUTLINE PROGRAM

3.1 Algorithm

The purpose of the algorithm is to map the information network onto the network of arguments. The argument network represents the outline or table of contents of the specification and its hierarchical structure is basically unrelated to that of the information network. Only a portion of the nodes in the information network are used in the algorithm and they are known as provisions. They are generally at or near the output level of the information network. The outlining algorithm does not account for any logical relations between the provisions. The user specifies the provisions, a set of arguments associated with each provision, and the hierarchical structure of the arguments.

There are two principal concepts governing the algorithm. First, the user specifies the order in which the argument trees will be expressed in the outline. Second, the algorithm is provision directed. That is, when there are provisions related to an argument and not yet outlined because they are also related to other arguments, additional argument trees are appended to the network of arguments to allow entry of these provisions. Broadly, the flow of the algorithm can be represented in the following steps:

1. Obtain the next heading from the argument tree.
2. Determine if any provisions are related to the argument. If so, determine if they may be entered into the outline. (They may

if all of the other arguments associated with them have already appeared in the current portion of the argument network).

3. If there are provisions that may not yet be entered, and if the tree containing the argument associated first with the provision is the primary tree, then append other trees of arguments so that those arguments that are required will become available. Go to 1.
4. If there are no provisions associated with the argument determine if the tree to which the argument being considered belongs is the primary tree. If not, suppress the heading. Go to 1.

More detail about the algorithm is shown in the logic diagrams for subroutines OUTLN and ADVANC in the next section.

3.2 Logic Diagrams

The following diagrams describe the structure of the program and the major subroutines.

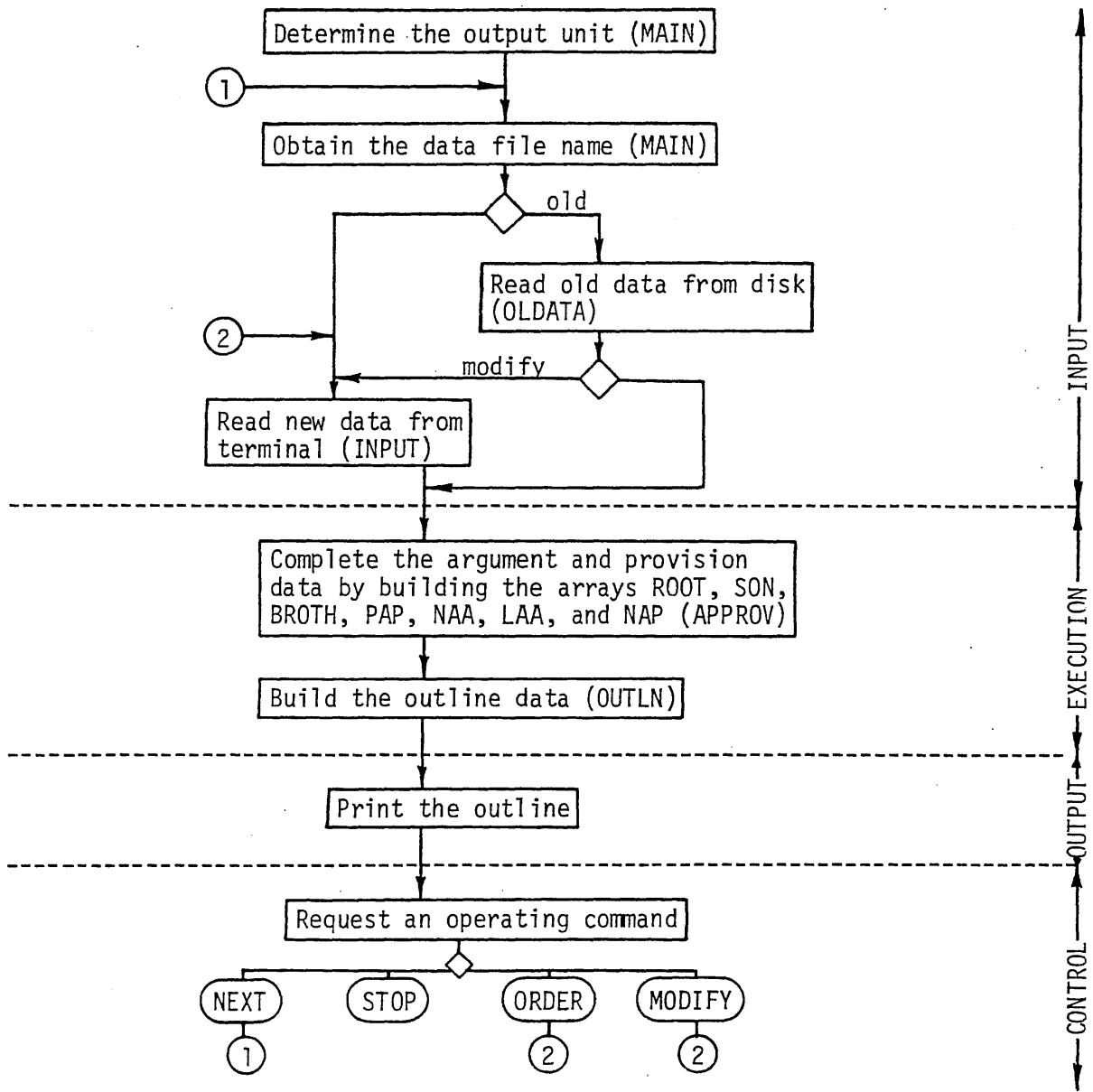
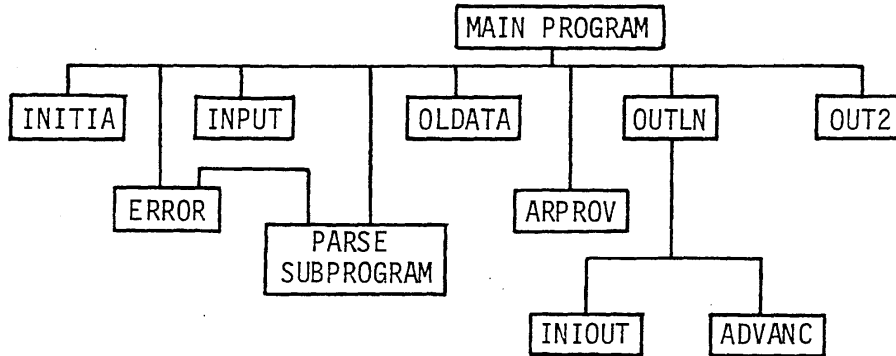


Figure 3.1 Block Diagram for the Outline Program



NOTES:

1. PARSE contains several function and subroutines, including PARSE, WDINIT, SCAN, SETOUT, RDLIN, END, MATCH, STRING, and FIXED.
2. INPUT and MAIN include the file PARCOM, used for the common declarations needed for PARSE.

Figure 3.2 Subroutine Linkage, (OUTLINE)

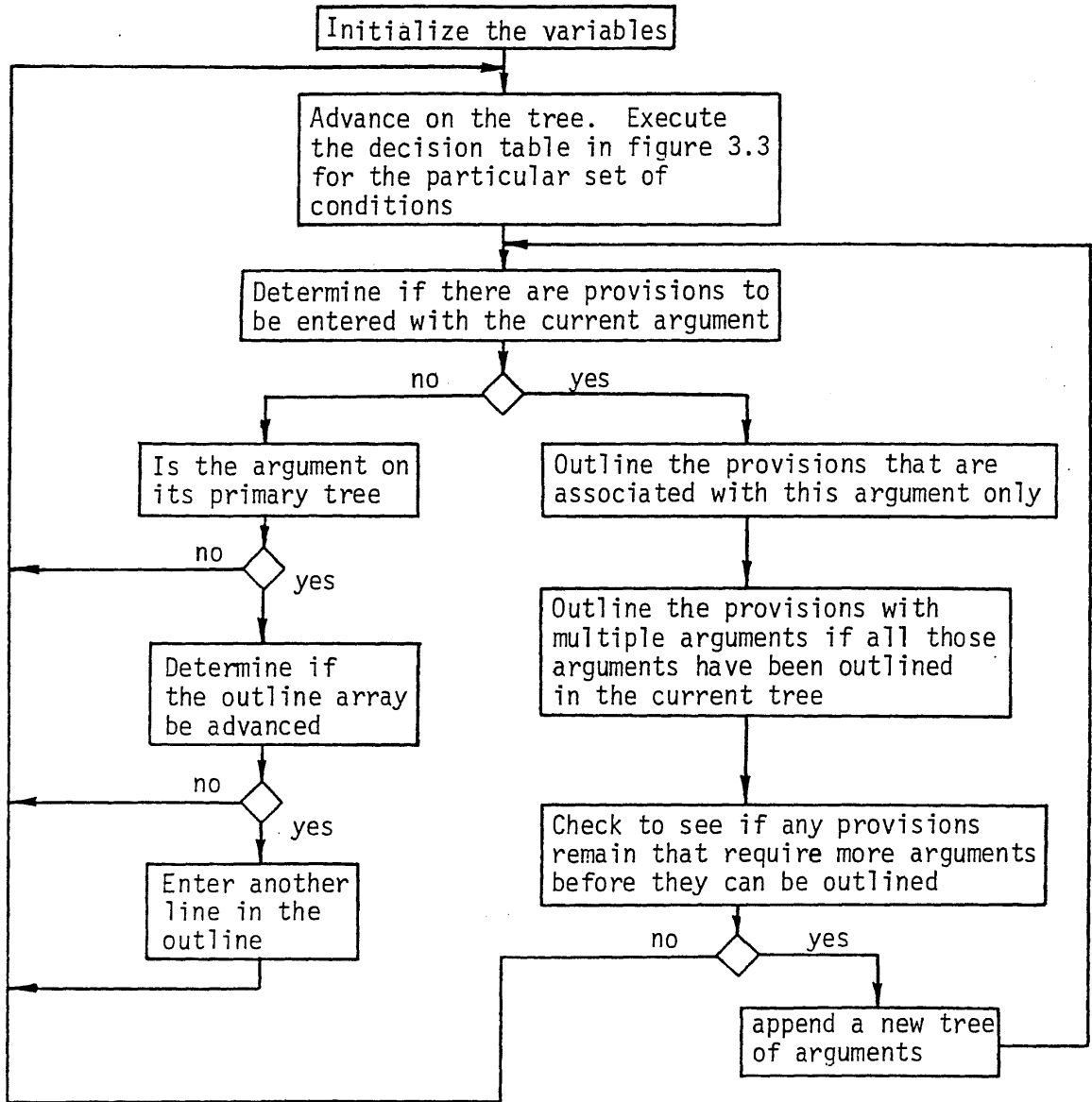


Figure 3.3 Subroutine OUTLN (OUTLINE)

DECISION TABLE FOR OUTLINING ALGORITHM

	1	2	3	4	5	6	7	8	9	10	11
1 All provisions outlined	*	T	F	F	F	F	F	F	F	F	F
2 Initial root, L0 = 0	*	.	T	F	F	F	F	F	F	F	F
3 Initial =	*	.	.	T	T	T	T	F	F	F	F
4 Argument has son	*	.	.	T	F	F	F
5 Argument has brother	*	.	.	.	T	T	F	T	F	F	F
6 Suppressed =	*	.	.	.	F	T
7 Argument is tree root	*	F	T	T	T
8 Argument at bottom stack	*	F	T	T
9 Additional argument tree	*	T	F

1 Exit from outlining	*	X									
2 Put son in stack	*		X								
3 Put brother in stack	*			X	X		X				
4 Clear stack to parent	*					X		X			
5 Remove argument from stack	*								X		
6 Clear stack, get next tree root	*									X	
7 Initial =	*		Y	Y	Y	Y	N	Y	N	Y	Y
8 Advance =	*		Y	Y	Y	Y		Y		N	Y
9 LI = L0	*		X	X	X	X		X		X	X
10 L0 = L0+1 (Incr. indent.)	*		X	X		X					
11 L0 = L0-1 (Dec. indent.)	*						X		X		
12 Exit from table	*		X	X	X	X		X		X	X
13 Re-enter table	*						X		X		
14 Outlining fails provision w/o Arg.	*										X

DERIVED DECISION NETWORK

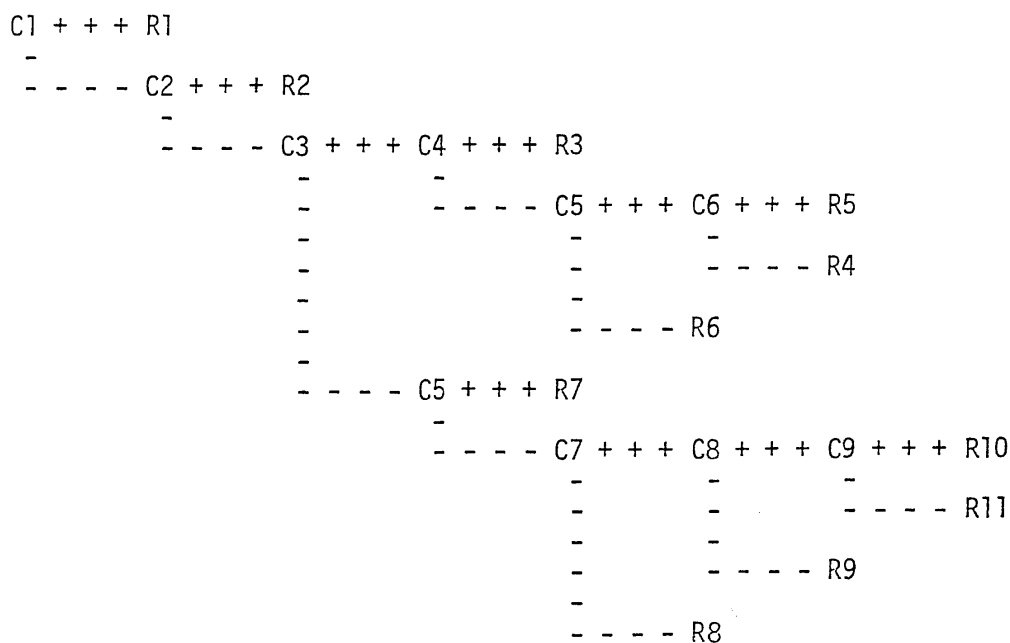
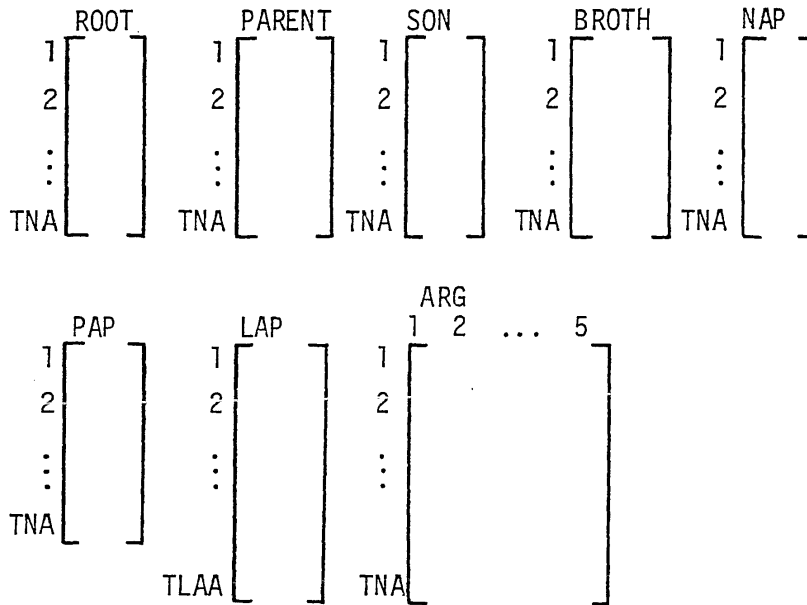


Figure 3.4 Subroutine ADVANC (OUTLINE)

3.3 Data Structure

The figures on the following pages show the principal items of data used in the program. A complete list of all the data names is in the glossary, section 3.4.

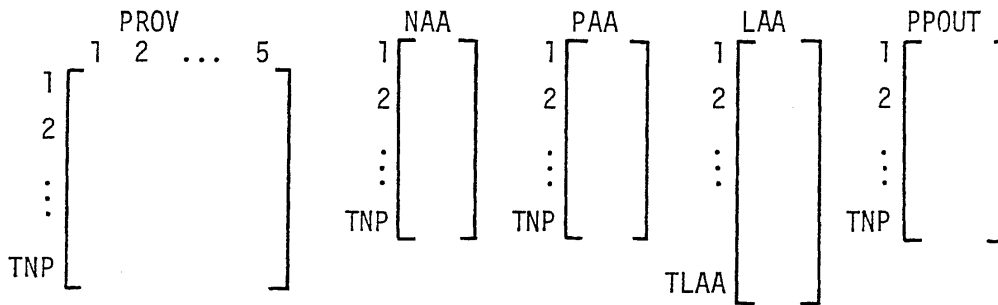


TNA is the total number of arguments
TLAA is the total number of associated arguments

For all of the lists that are TNS is length, the row number is the argument number. The contents of the lists are:

- ROOT - the argument at the root of the argument tree
- PARENT - the parent of the argument
- SON - the first son of the argument
- BROTH - the next brother of the argument
- NAP - the number of associated provisions
- PAP - the row number in LAP where the first associated provision is located
- LAP - the list of associated provisions
- ARG - the alphanumeric description of the argument

Figure 3.5 Argument Data (OUTLINE)

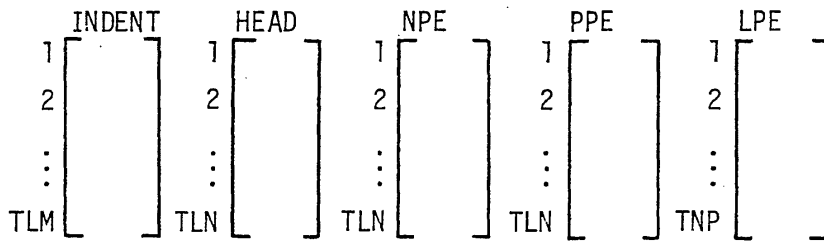


TNP is the total number of provisions
TLAA is the total number of associated arguments

For all of the lists that are TNP in length, the row number is the provision number. The contents of the lists are:

- PROV - the alphanumeric description of the provision
- NAA - the number of associated arguments
- PAA - the row number in LAA where the first associated argument is located
- LAA - the list of associated arguments
- PPOUT - the line number of the outline where the provision is located

Figure 3.6 Provison Data (OUTLINE)



TLN is the total number of lines (headings) in the outline.

For all of the lists except LPE, the row number is the line number. The contents are:

- INDENT - the level of indentation
- HEAD - the number of the argument used as a heading
- NPE - the number of provisions entered at this heading
- PPE - the row number in LPE where the first provision for this line is found
- LPE - the list of provisions entered in the outline

Figure 3.7 Outline Data (OUTLINE)

Contents of the line	Format
TNP TNA	2I2
PROV NUMA LA for prov. 1	5A6, 6I3
PROV NUMA LA for prov. 2	
⋮	
PROV NUMA LA for prov. TNP	
ARG PARENT for arg. 1	5A6, I3
ARG PARENT for arg. 2	
⋮	
ARG PARENT for arg. TNA	

NOTES:

1. NUMA is the number of associated arguments for each provision (same as NAA).
2. LA is the list of arguments associated with the provision (five at most).

Figure 3.8 Permanent Data File (OUTLINE)

100
200
300
400
500
600
700
800
900

3.4 GLOSSARY

1000
1100
1200
1300
1400
1500

PLEASE NOTE THAT THE NAMES ASSOCIATED WITH PARSE ARE NOT IN THIS LIST.
REFER TO THE GLOSSARIES FOR NETWORK OR TABLE FOR THEIR DEFINITIONS.

1600

SYMBOL : DESCRIPTION

1700

1800

A : ARGUMENT NUMBER

1900

2000

ADV : LOGICAL VARIABLE THAT IS FALSE IF THE ARGUMENT IS NOT
ENTERED IN THE OUTLINE

2100

2200

2300

ADVANC : A SUBROUTINE

2400

2500

AN : ARGUMENT NUMBER

2600

2700

ARPROV : A SUBROUTINE

2800

2900

ARG : ARRAY CONTAINING THE DESCRIPTIVE TITLE OF THE ARGUMENTS

3000

3100

BL : A BLANK SYMBOL (4 CHARACTERS)

3200

3300

BLANK : A BLANK SYMBOL (6 CHARACTERS)

3400

3500

PROTH : VECTOR CONTAINING THE FIRST BROTHER OF EACH ARGUMENT

3600

3700

DF : NUMBER OF THE DATA FILE

3800

3900

DOT : THE DOT SYMBOL

4000

4100

END IN : LOGICAL VARIABLE USED FOR DETECTING THE END OF PROVISION
DATA OF ARGUMENT DATA IN THE INPUT ROUTINE

4200

4300

4400

LEAD : VECTOR CONTAINING THE ARGUMENT NUMBER FOR EACH HEADING
IN THE OUTLINE

4500

4600

4700

INC : NUMBER OF ARGUMENTS THAT ARE NOT IN THE STACK

4800

4900

INDENT : VECTOR CONTAINING THE INDENTATION OF EACH LINE IN THE
OUTLINE

5000

5100

INITOUT : A SUBROUTINE

5200

5300

5400

INIT : LOGICAL VARIABLE THAT IS FALSE IF SUBROUTINE ADVANC IS
BEING PROCESSED A SECOND TIME WITHOUT RETURN

5500

5600

INITIA : A SUBROUTINE

5700

5800

INPUT : A SUBROUTINE

5900

6000		:	
6100	IA	:	DUMMY NAME FOR LAA
6200		:	
6300	LAA	:	VECTOR CONTAINING THE LIST OF ARGUMENTS ASSOCIATED WITH
6400		:	THE PROVISIONS
6500		:	
6600	LAP	:	VECTOR CONTAINING THE LIST OF PROVISIONS ASSOCIATED WITH
6700		:	THE ARGUMENTS
6800		:	
6900	IT	:	INDENTATION OF THE HEADING
7000		:	
7100	LIST	:	VECTOR CONTAINING THE LIST OF PROVISIONS ELIGIBLE TO
7200		:	BE OUTLINED AT THIS HEADING
7300		:	
7400	LN	:	LINE NUMBER OF THE OUTLINE
7500		:	
7600	LD	:	INDENTATION LEVEL OF THE NEXT HEADING
7700		:	
7800	LPF	:	VECTOR CONTAINING THE LIST OF PROVISION NUMBERS IN THE
7900		:	THE ORDER THEY APPEAR IN THE OUTLINE
8000		:	
8100	MISS	:	THE NUMBER OF AN ASSOCIATED ARGUMENT THAT IS NOT IN THE
8200		:	CURRENT ARGUMENT NETWORK
8300		:	
8400	MODIFY	:	LOGICAL VARIABLE THAT IS TRUE WHEN MODIFICATIONS TO THE
8500		:	DATA ARE BEING MADE
8600		:	
8700	MAA	:	VECTOR CONTAINING THE NUMBER OF ARGUMENTS ASSOCIATED
8800		:	WITH EACH PROVISION
8900		:	
9000	MAP	:	VECTOR CONTAINING THE NUMBER OF PROVISIONS THAT EACH
9100		:	ARGUMENT IS ASSOCIATED WITH
9200		:	
9300	NP	:	NUMBER OF PROVISIONS ENTERED AT THE CURRENT HEADING
9400		:	
9500	NPF	:	VECTOR CONTAINING THE NUMBER OF PROVISIONS ENTERED AT
9600		:	LINE (HEADING) OF THE OUTLINE
9700		:	
9800	NPP	:	NUMBER OF PROVISIONS REMAINING TO BE OUTLINED
9900		:	
10000	NRU	:	RULE NUMBER IN THE DECISION TABLE FOR SUBROUTINE ADVANC
10100		:	
10200	NST	:	NUMBER OF THE STEP IN THE FLOW DIAGRAM FOR SUBROUTINE OUTLN
10300		:	
10400	NMA	:	VECTOR CONTAINING THE NUMBER OF ASSOCIATED ARGUMENTS
10500		:	FOR EACH PROVISION
10600		:	
10700	NMA	:	NUMBER OF ASSOCIATED ARGUMENTS
10800		:	
10900	OLDAT	:	LOGICAL VARIABLE THAT IS TRUE FOR OLD DATA
11000		:	
11100	OLDATA	:	A SUBROUTINE
11200		:	
11300	ORD	:	LOGICAL VARIABLE ASSOCIATED WITH THE COMMAND ORDER
11400		:	
11500	ORDER	:	VECTOR CONTAINING THE ROOTS OF THE ARGUMENT TREES IN
11600		:	THE ORDER THEY ARE TO BE USED
11700		:	
11800	OUTCOM	:	FILE CONTAINING THE COMMON DECLARATIONS FOR THE PROGRAM
11900		:	
12000	OUTLN	:	A SUBROUTINE
12100		:	

-99-

12200	OUT2	:	A SUBROUTINE
12300		:	
12400	P	:	PROVISION NUMBER
12500		:	
12600	PAA	:	VECTOR CONTAINING THE ROW NUMBER OF LAA WHERE THE FIRST
12700		:	ARGUMENT ASSOCIATED WITH EACH PROVISION IS STORED
12800		:	
12900	PAP	:	VECTOR CONTAINING THE ROW NUMBER OF LAP WHERE THE FIRST
13000		:	PROVISION EACH ARGUMENT IS ASSOCIATED WITH IS STORED
13100		:	
13200	PARENT	:	VECTOR CONTAINING THE PARENT OF EACH ARGUMENT
13300		:	
13400	PN	:	PROVISION NUMBER
13500		:	
13600	POINT	:	POINTER TO THE PROVISION BEING ENTERED IN THE OUTLINE
13700		:	
13800	PPF	:	VECTOR CONTAINING THE ROW NUMBER IN LPE WHERE THE FIRST
13900		:	PROVISION FOR EACH LINE (HEADING) IS STORED
14000		:	
14100	PROUT	:	VECTOR CONTAINING THE LINE NUMBER WHERE EACH PROVISION
14200		:	IS TO BE ENTERED IN THE OUTLINE
14300		:	
14400	PROV	:	ARRAY CONTAINING THE DESCRIPTIVE TITLE OF THE PROVISIONS
14500		:	
14600	ROOT	:	VECTOR CONTAINING THE ROOT OF THE ARGUMENT TREE FOR
14700		:	EACH ARGUMENT
14800		:	
14900	SON	:	VECTOR CONTAINING THE SON OF EACH ARGUMENT
15000		:	
15100	STACK	:	VECTOR CONTAINING A STACK OF ARGUMENT NUMBERS THAT HAVE
15200		:	PROVISIONS YET TO BE OUTLINED IN THE CURRENT TREE
15300		:	
15400	SUPP	:	LOGICAL VARIABLE THAT IS TRUE IF THE ARGUMENT IS NOT
15500		:	ENTERED IN THE OUTLINE
15600		:	
15700	TLAA	:	TOTAL NUMBER OF ASSOCIATIONS BETWEEN PROVISIONS AND
15800		:	ARGUMENTS---THE LENGTH OF LAA AND LAP
15900		:	
16000	TLN	:	TOTAL NUMBER OF LINES IN THE OUTLINE
16100		:	
16200	TNA	:	TOTAL NUMBER OF ARGUMENTS
16300		:	
16400	TNP	:	TOTAL NUMBER OF PROVISIONS
16500		:	
16600	Y	:	SUBSCRIPT OF ORDER
16700		:	
16800	XMAX	:	TOTAL NUMBER OF ROOTS (NUMBER OF TREES OF ARGUMENTS)
16900		:	
17000	Z	:	NUMBER OF THE FILE USED FOR OUTPUT

10
20
30
40
50
60
70
80
90
100
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
2600
2700
2800
2900
3000
3100
3200
3300
3400
3500
3600
3700
3800
3900
4000
4100
5000
5100
5110
5120
5130
5140
5150
5160
5170
5180
5200
5205
5210

3.5 PROGRAM LISTING

```

& RFSEI FREE
FILE 5=INCOM,UNIT=REMOTE,RECORD=14
FILE 6=OUTCOM,UNIT=REMOTE,RECORD=14
FILE 1=OUTPUT,UNIT=PRINTER,RECORD=23
FILE 3=OUTFILE1,UNIT=DISKPACK,RECORD=14,BLOCKING=30,SAVE=1
& INCLUDE 'PARSE'
& INCLUDE 'OUTCOM'
C
C THIS FILE DIMENSIONS ALL THE ARRAYS USED IN THE
C PROGRAM AND ASSIGNS THEM TO COMMON
C
C INTEGER NAA(30),FAA(30),LAA(45),ROOT(30),PARENT(30),
C *SUN(30),BROTH(30),NAP(30),PAP(30),LAP(45),ORDER(10),
C *INDENT(30),HEAD(30),NPE(30),PPE(30),LPE(30),PPOUT(30),
C *STACK(10),LIST(10),LA(30,5),A(30),P(30),NUA(30)
C
C REAL PRUV(30,5),ARG(30,5)
C
C LOGICAL ENLIN,ORJ,INIT,SUPP,ADV,OLDAT,MODIFY,UNPACK
C
C COMMON/INT/DF,P,TNP,TNA,TLAA,AN,XMAX,LN,NPR,LI,LO,
C *TLN,NAA,FAA,LAA,ROOT,PARENT,SUN,BROTH,NAP,PAP,
C *LAP,ORDER,INDENT,HEAD,NPE,PPE,LPE,PPOUT,STACK,LIST,
C *MISS,INC,NP,PDINI,X,K,LA,A,P,NUA,Z
C
C COMMON/REAL/PRUV,ARG,BLANK,BL,OUT
C COMMON/LUG/ENLIN,ORJ,INIT,SUPP,ADV,OLDAT,MODIFY,UNPACK
C
C END OF OUTCOM
& INCLUDE 'PARCOM'
C
C THIS FILE LISTS THE DECLARATIONS FOR USE OF THE INPUT SCANNER
C
C LOGICAL END, FIXED, MATCH, STRING, NEXT
C COMMON/SCANNER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
C EQUIVALENCE (IVALUE,VALUE)
C
C END OF PARCOM
C
C DELETE THE ECHO PRINT OF THE SCANNER
DATA PRI/"OFF"/
C
C SET THE LENGTH OF A LINE OF INPUT EQUAL TO 72 SPACES
C AND THE NUMBER OF BLANKS REQUIRED TO SIGNIFY THE END
C OF A RECORD EQUAL TO 10
C CALL W0INIT(10,72,PRI)
C
C READ THE OUTPUT UNIT IN
Z = 6
    
```

```
5215 WRITE(Z,1005)
5220 CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
5225 IF(MATCH("P",1)) Z = 1
5230 CALL SETUUT(Z)
5235 NEXT= .TRUE.
5300 C READ THE DATA FILE NAME IN
5400 I = 0
5500 10 WRITE(6,1200)
5510 CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
5520 IF(MATCH("YES",3)) CALL W0INIT(10,72,"UN ")
5522 C
5524 C INITIALIZE THE ARRAYS
5526 30 CALL INITIA
5530 40 WRITE(6,1000)
5540 CALL RDLIN
5550 ENTITY(NWD+1)=6H.
5560 CLOSE (3)
5570 CHANGE(3,TITLE=ENTITY)
5580 DF=3
6700 C
6800 C CHECK FOR EXISTING DATAFILE
6900 I = 0
7000 INQUIRE(3,RESIDENT=UNPACK)
7050 IF(.NOT. UNPACK) GO TO 50
7100 C
7150 C EXISTING FILE, DOUBLE CHECK THE NAME
7200 WRITE(6,1010)
7250 CALL RDLIN
7300 IF(MATCH("YES",3)) GO TO 63
7600 GO TO 40
7700 C
7800 C READ NEW DATA IN FROM THE TERMINAL
7900 50 CALL INPUT
8000 GO TO 70
8100 C
8200 C READ THE OLD DATA FROM THE DATAFILE
8340 63 CALL OLDATA
8400 C
8500 C CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
8600 WRITE(6,1020)
8700 CALL RDLIN
8800 IF(.NOT.MATCH("YES",3)) GO TO 100
8810 MODIFY=.TRUE.
8900 65 CALL INPUT
9000 70 CALL APPROV
9100 CALL OUTLN
9200 C PRINT OUT THE OUTLINE
9300 C CALL OUT1
9350 CALL OUT2
9400 100 WRITE(6,1030)
9500 CALL RDLIN
9600 110 IF(ENO(X)) GO TO 100
9700 IF(MATCH("NEXT",3)) GO TO 10
9800 IF(MATCH("STOP",3)) STOP
9900 IF(MATCH("ORDER",3)) GO TO 150
9910 WRITE(6,1100)
10000 I = I + 1
10100 IF(I.EQ.10) CALL ERROR(1)
10200 GO TO 100
10240 150 CALL INITIA
10270 CALL OLDATA
10300 ORD=.TRUE.
```

```

10400          GO TO 65
10500      C
10600      1000 FORMAT(1H,"ENTER THE DATA FILE NAME")
10650      1005 FORMAT(1H,"ENTER P FOR OUTPUT ON THE ON-SITE PRINTER",
10660          1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL")
10700      1010 FORMAT(1H,"DATA FILE WITH THIS NAME EXISTS. ",
10750          * "DO YOU WANT TO USE IT?",/)
10800      1020 FORMAT(1H,"DO YOU WANT TO MODIFY THE EXISTING DATA?")
10900      1030 FORMAT(1H,"ENTER A PROGRAM CUMMAND")
10910      1100 FORMAT(" INPUT ERROR --- RE-ENTER ON A NEW LINE")
10920      1200 FORMAT(" DO YOU WANT THE PRINTOUT OF THE INPUT")
11000      END

```

```

11100      C
11200      C -----

```

```

11300      C
11400          SUBROUTINE INITIA
11500      $INCLUDE 'OUTCOM'
11600          DO 40 I=1,30
11700          DO 10 J=1,5
11800          ARG(I,J)=BLANK
11900          10 PROV(I,J)=BLANK
12000          NAA(I)=0
12100          PAA(I)=0
12200          RUOT(I)=0
12300          PARENT(I)=0
12400          SUN(I)=0
12500          BROTH(I)=0
12600          NAP(I)=0
12700          PAP(I)=0
12800          INDENT(I)=0
12900          HEAD(I)=0
13000          NPE(I)=0
13100          PPE(I)=0
13200          LPE(I)=0
13300          40 PROJ(I)=0
13400          DO 50 I=1,10
13500          STACK(I)=0
13600          50 LIST(I)=0
13700          OKD=.FALSE.
13800          ENLIN=.FALSE.
13810          MODIFY=.FALSE.
13820          TNA=0
13830          TNP=0
13900          RETURN
14000      END

```

```

14100      C
14200      C -----

```

```

14300      C
14400          SUBROUTINE INPUT
14410      CERUG MONITOR(1) ORD,P,PN,TNP,NWD1,PROV,II,LA,MUA,ENLIN,AN
14420      CERUG MONITOR(1) ORDER,XMAX
14500      C
14600      C
14700      C THIS SUBROUTINE READS THE PROVISIONS WITH THEIR ASSOCIATE
14800      C D ARGUMENTS, THE LIST OR ARGUMENTS WITH THEIR PARENTS AND
14900      C BUILDS THE VECTORS LAA,NAA, AND NAP
15000      C
15100      $ INCLUDE 'OUTCOM'
15200      $ INCLUDE 'PARCOM'
15300          IF (ORD) GO TO 110
15310          IF(.NOT.MODIFY) GO TO 9
15320          DO 6 K=1,TNP

```



```
15330      6 P(K)=K
15340      DU 7 K=1,TNA
15350      8 A(K)=K
15400      9 WRITE(6,1000)
15500      K=0
15600      C
15700      C BEGIN SCANNING FOR INPUT
15800      C
15900      CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
16000      10 IF(MATCH("PROVIS",3)) GO TO 20
16100      15 IF(MATCH("END",3)) GO TO 100
16200      IF(MATCH("ARGUME",3)) GO TO 60
16300      IF(MATCH("ORDER",3)) GO TO 110
16400      C
16500      C IMPROPER INPUT DATA
16600      C
16700      WRITE(6,1100)
16800      CALL RDLIN
16900      K=K+1
17000      IF(K.EQ.10) CALL ERROR(1)
17100      GO TO 10
17200      C
17300      C HERE WE HAVE THE LIST OF PROVISIONS AND ASSOCIATED ARG.
17400      C
17510      20 WRITE(6,1060)
17700      C      PN= COUNTER OF PROVISIONS
17800      PN=J
17900      K=0
18000      CALL RDLIN
18050      26 IF(.NOT.FIXED(X)) GO TO 28
18060      PN=PN+1
18065      IF(MODIFY) PN=IVALUE
18070      P(PN)=IVALUE
18080      TNP=MAX0(TNP,P(PN))
18100      IF(STRING(X)) GO TO 30
18200      27 IF(FIXED(X)) GO TO 40
18300      28 IF(END(X)) GO TO 50
18400      IF(ENLIN) GOTO 29
18500      C IMPROPER INPUT DATA
18600      WRITE(6,1100)
18700      CALL RDLIN
18800      K=K+1
18900      IF(K.EQ.10) CALL ERROR(1)
19000      GO TO 26
19100      29 CONTINUE
19300      C NEW LINE
19400      GO TO 15
19500      C
19600      C READ TITLE OF PROVISION
19700      C
19800      C      II: COUNTER OF ASSOCIATED ARGUMENTS OF PROVISION
19900      30 II=0
20100      C
20300      NWD1=(NCHAR+5)/6
20410      NWD1=MIN0(NWD1,5)
20420      IF(.NOT.MODIFY) GO TO 32
20425      DU 31 J=1,5
20430      31 PROV(P(PN),J)=BLANK
20500      32 DU 35 J=1,NWD1
20600      35 PROV(P(PN),J)=ENTITY(J)
20700      GO TO 27
20900      40 II=II+1
```

```

21100      LA(P(PN),II)=IVALUE
21200      NUA(P(PN))=II
21300      ENLIN=.FALSE.
21400      GU TO 27
21500      50 ENLIN=.TRUE.
21600      GU TO 26
21700      C
21800      C
21900      C
22000      C          AN: COUNTER OF ARGUMENTS
22010      60 WRITE(6,1070)
22100      AN=J
22200      K=0
22300      CALL RDLIN
22310      ENLIN=.FALSE.
22315      65 IF(.NOT.FIXED(X)) GO TO 90
22320      AN=AN+1
22325      A(AJ)=IVALUE
22330      IF(.MODIFY) AN=IVALUE
22400      IF(.NOT.STRING(X)) GO TO 80
22600      NWD1=(NCHAR+5)/6
22700      NWD1=MINO(5,NWD1)
22800      C
22900      C
23000      C
23100      C
23110      IF(.NOT.MODIFY) GO TO 68
23115      DU 67 J=1,5
23120      67 ARG(A(AN),J)=BLANK
23200      68 DU 70 J=1,NWD1
23300      70 ARG(A(AN),J)=ENTITY(J)
23400      80 IF(.NOT.FIXED(X)) GO TO 90
23500      PARENT(A(AN))=IVALUE
23600      ENLIN=.FALSE.
23700      90 IF(END(X)) GO TO 95
23800      C
23900      C      ENLIN.= WHEN THE ENTITY IS NOT STRING, INTEGER, OR END
24000      C          OF LINE, ENLIN IS USED TO SPECIFY THAT THIS
24100      C          IS A NEW LINE IF ITS VALUE IS TRUE. BUT IF
24200      C          ITS VALUE IS FALSE, THERE IS AN ERROR.
24300      C
24400      IF(ENLIN) GO TO 100
24500      WRITE(6,1100)
24600      CALL RDLIN
24700      K=K+1
24800      IF(K.EQ.10) CALL ERROR(1)
24900      GU TO 60
25000      C NEW LINE
25100      100 TNA=AN
25200      GU TO 10
25300      95 ENLIN=.TRUE.
25400      GU TO 95
25500      110 I=0
25510      WRITE(6,1080)
25600      K=0
25700      120 IF(.NOT.FIXED(X)) GO TO 130
25800      I=I+1
25900      ORDER(I)=IVALUE
26000      ENLIN=.FALSE.
26100      GO TO 120
26200      130 IF(.NOT.END(X)) GO TO 140
26300      ENLIN=.TRUE.

```

```

26400      C
26500      GO TO 120
26600      140 IF(ENLIN) GO TO 150
26700      WRITE(6,1100)
26800      CALL RDLINE
26900      K=K+1
27000      IF(K.EQ.10) CALL ERROR(1)
27100      GO TO 10
27200      150 XMAX=I
27300      GO TO 10
27310      160 IF(.NOT.ORD) GO TO 165
27320      ORD=.FALSE.
27330      RETURN
27400      165 REWIND DF
27500      WRITE(DF=1,1040) TNP,TNA
27600      DO 170 I=1,TNP
27700      NI=P(I)
27800      NUMA=NUA(NI)
27900      MPI=NI+1
28000      170 WRITE(DF=NPI,1050)NI,(PROV(NI,J),J=1,5),NUMA,(LA(NI,II),II=1,NUMA)
28100      DO 180 I=1,TNA
28200      NI=A(I)
28300      NAI=A(I)+31
28400      180 WRITE(DF=NAI,1050)NI,(ARG(NI,J),J=1,5),PARENT(NI)
29100      END FILE DF
29200      LOCK DF
29300      1000 FORMAT(" BEGIN INPUT INSTRUCTIONS.",
29400      * " ENTER THE WORD END WHEN FINISHED.")
29500      1100 FORMAT(" INPUT ERROR --- RE-ENTER ON A NEW LINE")
29600      1040 FORMAT(1H ,2I2)
29700      1050 FORMAT(1H ,I2,5A6,6I3)
29710      1060 FORMAT(" ENTER THE LIST OF PROVISIONS AND ASSOCIATED ARGUMENTS",
29715      :/, " ONE LINE FOR EACH PROVISION")
29720      1070 FORMAT(" ENTER THE LIST OF ARGUMENTS AND THEIR PARENTS",
29730      :/, " ONE LINE FOR EACH ARGUMENT")
29740      1080 FORMAT(" ENTER THE WORD END WHEN FINISHED")
29800      RETURN
29900      END
30000
30100      C
30200      C
30300      SUBROUTINE UDATA
30400      $ INCLUDE 'OUTCOM'
30500      REWIND DF
30510      READ(DF=1,1040) TNP,TNA
30610      DO 170 I=1,TNP
30620      N=I+1
31010      READ(DF=N,1050)(PROV(I,J),J=1,5),NUMA,(LA(I,II),II=1,NUMA)
31020      170 NUA(I)=NUMA
31110      DO 180 I=1,TNA
31310      NAI=I+31
31410      180 READ(DF=NAI,1050)(ARG(I,J),J=1,5),PARENT(I)
31600      REWIND DF
31700      OLDAT=.TRUE.
31800      1040 FORMAT(1H ,2I2)
31900      1050 FORMAT(1H ,2X,5A6,6I3)
32000      RETURN
32100      END
32200      C
32300      C
32400      C
32500      SUBROUTINE AKPROV

```

```

32600 CERUG MONITOR(1) ROOT,SON,BROTH,PAP,PAA,NUMA,NAA,LAA,NAP
32610 CERUG MONITOR(1) PARENT
32700 $ INCLUDE 'OUTCOM'
32800 INTEGER RROOT,PPAP
32900 C
33000 C THIS SUBROUTINE COMPLETES THE ARGUMENT AND PROVISION
33100 C DATA BY BUILDING THE ARRAYS ROOT,SON,BROTH,PAP,NAA,LAA,NAP
33200 C
33210 I=0
33215 DO 9 K=1,TNP
33220 NUMA=NUMA(K)
33222 NAA(K)=NUMA
33225 DO 5 J=1,NUMA
33230 I=I+1
33235 LAA(I)=LAA(K,J)
33240 5 NAP(LAA(I))=NAP(LAA(I))+1
33245 9 CONTINUE
33250 TLAA=I
33300 PPAP=NAP(1)+1
33400 DO 60 I=1,INA
33410 PARENT(I)=PARENT(I)
33500 IF(PARENT(I).EQ.0) GO TO 20
33600 IF(I.EQ.TNA) GO TO 10
33700 IF(PARENT(I).EQ.PARENT(I+1)) GO TO 10
33800 ROOT(I)=RROOT
33810 IF(PARENT(I+1).NE.0) GO TO 30
33820 SUN(I)=0
33830 GO TO 35
33900 GO TO 30
34000 10 ROOT(I)=RROOT
34100 BROTH(I)=I+1
34200 GO TO 40
34300 20 RROOT=PARENT(I+1)
34310 ROOT(I)=I
34400 BROTH(I-1)=0
34500 30 SUN(I)=I+1
34600 35 BROTH(I)=0
34700 40 IF(NAP(I).NE.0) GO TO 50
34800 PAP(I)=0
34900 GO TO 60
35000 50 PAP(I)=NAP(I)+PPAP
35100 PPAP=PAP(I)
35200 60 CONTINUE
35300 BROTH(INA)=0
35400 C BUILD PAA
35500 PAA(1)=1
35600 DO 70 I=2,TNP
35700 70 PAA(I)=PAA(I-1)+NAA(I-1)
35800 C BUILD LAP
35900 II=INP
36000 DO 80 M=1,TNP
36100 I=PAA(II)
36200 J=I+NAA(II)-1
36300 DO 80 K=1,J
36400 AN=LAA(K)
36500 PAP(AN)=PAP(AN)-1
36600 L=PAP(AN)
36700 80 LAP(L)=II
36750 II=II-1
36760 90 CONTINUE
36800 RETURN
36900 END

```

```

37000 C
37100 C - - - - -
37200 C
37300 C SUBROUTINE OUTLN
37310 CERUG MONITOR(1) STACK,AN,ADV,INIT,SUPP,LI,LU,NP,LIST,NPR,INC
37320 CERUG MONITOR(1) MISS,NPE,POINT,PPE,LN,NAP,TLN,ROOT,SON,BROTH,PAP
37340 CERUG MONITOR(1) LAP,PPOUT,K,X,N1,NZ,INDENT,HEAD,LPE,ORDER
37400 $ INCLUDE 'OUTCOM'
37500 C
37600 C 1. INITIALIZE OUTLINING VARIABLES
37700 C
37710 C
37800 C CALL INIOUT
37900 C
38000 C 2. ADVANCE ON THE TREE (SEE TABLE 1. DECISION TABLE FOR
38100 C ADVANCE)
38200 C
38300 C 20 NST=2
38305 C WRITE(1,300) NST
38310 C CALL ADVANC
38315 C IF(NPR.EQ.0) RETURN
38330 C 300 FORMAT(" ",T5,"STEP=",I2)
38400 C
38500 C 3. DETERMINE IF THERE ARE PROVISIONS TO BE ENTERED UNDER
38600 C THE HEADING OF THE CURRENT ARGUMENT
38700 C
38710 C 30 NST=3
38720 C WRITE(1,300) NST
38800 C IF(NAP(AN).EQ.0) GO TO 100
38900 C NP=J
39000 C N1=PAP(AN)
39100 C N2=N1+NAP(AN)-1
39200 C DO 31 J=N1,N2
39210 C PPOUT(LAP(J))=PPOUT(LAP(J))
39300 C IF(PPOUT(LAP(J)).NE.0) GO TO 31
39310 C ROOT(LAA(PAA(LAP(J))))=ROOT(LAA(PAA(LAP(J))))
39400 C IF (ROOT(LAA(PAA(LAP(J))))).NE.STACK(1)) GO TO 31
39500 C NP=NP+1
39600 C LIST(NP)=LAP(J)
39700 C 31 CONTINUE
39800 C
39900 C THE PROVISION(S) IS(ARE) ENTERED WHEN THE ARGUMENT FIRST
40000 C ON ITS LIST OF ARGUMENTS IS IN THE HIERARCHICALLY FIRST
40100 C ARGUMENT TREE OF THE STACK
40200 C
40210 C NP=NP
40300 C IF (NP.EQ.0) GO TO 100
40400 C
40500 C 4. ADVANCE OUTLINE
40600 C
40610 C NST=4
40620 C WRITE(1,300) NST
40630 C ADV=ADV
40700 C IF(.NOT.ADV) GO TO 41
40800 C LN=LN+1
40900 C INDENT(LN)=LU
41000 C HEAD(LN)=AN
41100 C SUPP=.FALSE.
41110 C NPE(LN)=0
41120 C PPE(LN)=POINT
41200 C 41 INC=0
41500 C

```

```
41600 C 5. ENTERING PROVISIONS
41700 C CHECK IF PROVISIONS CAN BE ENTERED. IF THEY ARE
41800 C COMPLETE (ALL THEIR ARGUMENTS APPEAR IN THE CURRENT
41900 C STACK OF ARGUMENTS) THEY MAY BE ENTERED
42000 C
42010 NST=5
42020 C WRITE(1,300) NST
42100 50 DO 59 J=1,NP
42200 N1=PAA(LIST(J))
42300 N2=N1-1+ NAA(LIST(J))
42400 C
42500 C 6. COMPLETE?
42600 C
42610 NST=6
42620 C WRITE(1,300) NST
42700 DO 58 JJ=N1,N2
42800 DO 57 KK=1,K
42810 STACK(KK)=STACK(KK)
42900 IF(STACK(KK).EQ.LAA(JJ)) GO TO 58
43000 57 CONTINUE
43100 MISS=LAA(JJ)
43200 INC=INC+1
43300 GO TO 59
43400 58 CONTINUE
43500 C
43600 C ALL ITS ARGUMENTS ARE IN THE CURRENT STACK, THEREFORE
43700 C PROVISION IS COMPLETE
43800 C
43900 C 7. ENTER PROVISION
43910 NST=7
43920 C WRITE(1,300) NST
44000 NPE(LN)=NPE(LN)+1
44100 LPE(POINT)=LIST(J)
44200 POINT=POINT+1
44300 PPOUT(LIST(J))=LN
44400 NPR=NPR-1
44500 59 CONTINUE
44600 C
44700 C 8. ANY INCOMPLETE PROVISION. A NEW TREE OF ARGUMENTS IS
44800 C APPENDED AT THE ROOT AS THE ELDEST SON OF THE CURRENT
44900 C ARGUMENT
45000 C
45010 NST=8
45020 C WRITE(1,300) NST
45100 IF(INC.EQ.0) GO TO 20
45200 C
45300 C 9. APPEND TREE
45400 C
45410 NST=9
45420 C WRITE(1,300) NST
45500 K=K+1
45600 STACK(K)=ROOT(MISS)
45700 IF(STACK(K).NE.0) GO TO 95
45800 STACK(K)=ROOT(MISS+1)
45900 95 ADV=.TRUE.
46000 LI=LO
46100 LU=LO+1
46200 INIT=.TRUE.
46300 AN=STACK(K)
46400 C
46500 GO TO 30
46600 C
```

```

46610      NPR=NPR
46700      IF(NPR.EQ.0) RETURN
46800      C
46900      C 10. PRIMARY TREE.    NO PROVISIONS FOR POTENTIAL ENTRIES ARE
47000      C      ENCOUNTERED AT A HEADING
47100      C
47200      C 100 IF(ROOT(AN).EQ.STACK(1)) GO TO 105
47210          NST=10
47220      C      WRITE(1,300) NST
47300          SUPP=.TRUE.
47400          LU=LI
47500          GO TO 20
47510      C
47600      C
47700      C 105 SUPP=.FALSE.
47710          ADV=ADV
47715          NST=10
47720      C      WRITE(1,300) NST
47800          IF(.NOT.ADV) GO TO 20
47900          LN=LN+1
48000          INDENT(LN)=LD
48100          HEAD(LN)=AN
48200          GO TO 40
48300      C
48400          END
48500      C
48600      C -----
48700      C
48800          SUBROUTINE INIDUT
48900      $INCLUDE 'OUTCUM'
49000      C THIS SUBROUTINE INITIALIZES OUTLINING VARIABLES
49100      C
49200          LN=0
49300          LI=0
49400          LU=0
49500          K=1
49600          X=1
49700          STACK(K)=ORDER(X)
49800          SUPP=.FALSE.
49900          NPR=INP
50000          POINT=1
50100          AN=STACK(K)
50200          PPE(I)=1
50300          RETURN
50400          END
50500      C
50600      C -----
50700      C
50800          SUBROUTINE ADVANC
50810      CERUG MONITOR(1) STACK,AN,ADV,INIT,SUPP,LI,LU,NP,LIST,NPR,INC
50820      CERUG MONITOR(1) MISS,NPE,POINT,PPE,LN,ILN,NAP,ROOT,SON,BROTH,PAP
50830      CERUG MONITOR(1) LAP,PPOUT,K,X
50900      $INCLUDE 'OUTCUM'
51000      C THIS SUBROUTINE DOES THE ADVANCE ON THE TREE (SEE TABLE 1
51100      C DECISION TABLE FOR ADVANCE)
51200      C
51300      C R U L E 1.  NPR=0  MEANS:  OUTLINING FINISHED
51310          NPR=NPR
51400          IF(NPR.NE.0) GO TO 2
51410          NRU=1
51420      C      WRITE(1,200) NRU
51430      C 200 FORMAT("U ",T70,"RULE ",I2)

```

```
51500      TLN=LN
51600      RETURN
51700      C  R U L E  2.  LO=0  INITIAL PRIMARY TREE
51800      C
51810      2  IF(LO.NE.0) GO TO 3
51820      NRU=2
51830      C      WRITE(1,200) NRU
51900      IF(LO.EQ.0) GO TO 110
52000      3  IF(.NOT.INIT) GO TO 7
52100      IF(SUN(AN).EQ.0) GO TO 5
52200      C
52300      C  R U L E  3.  SON IS NEW ARGUMENT
52400      C
52410      NRU=3
52420      C      WRITE(1,200) NRU
52500      K=K+1
52600      STACK(K)=SUN(AN)
52700      AN=STACK(K)
52800      GO TO 110
52900      5  IF(BROTH(AN).EQ.0) GO TO 65
53000      K=K+1
53100      STACK(K)=BROTH(AN)
53200      AN=STACK(K)
53210      NRU=4
53220      IF(SUPP) NRU=5
53230      C      WRITE(1,200) NRU
53300      IF(SUPP) GO TO 110
53400      C
53500      C  R U L E  4.  SUPP=.FALSE.  BROTHER IS NEW ARGUMENT
53600      C  R U L E  5.  SUPP=.TRUE.  BROTHER IS NEW ARGUMENT
53700      C
53800      GO TO 105
53900      C  R U L E S  6 AND 8  REENTER WITH PARENT
54000      C
54010      66 NRU=6
54020      C      WRITE(1,200) NRU
54030      67 K=K
54040      STACK(K)=STACK(K)
54050      PARENT(AN)=PARENT(AN)
54100      IF(STACK(K).EQ.PARENT(AN)) GO TO 69
54200      K=K-1
54300      GO TO 67
54400      69 INIT=.FALSE.
54500      LU=LO-1
54600      AN=STACK(K)
54700      GO TO 3
54800      7  IF(BROTH(AN).EQ.0) GO TO 70
54900      C  R U L E  7.  BROTHER IS NEW ARGUMENT
55000      C
55010      NRU=7
55020      C      WRITE(1,200) NRU
55100      K=K+1
55200      STACK(K)=BROTH(AN)
55300      AN=STACK(K)
55400      GO TO 105
55410      IF(PARENT(AN).NE.0) NRU=8
55420      C      WRITE(1,200) NRU
55500      70 IF(PARENT(AN).NE.0) GO TO 68
55600      C
55700      C  R U L E  8.  PARENT .NE. 0  REENTER WITH PARENT
55800      C
55900      IF(K.EQ.1) GO TO 100
```



```
56000 C
56100 C R U L E 9. PROCEED TO APEND ANOTHER TREE
56110 NRU=9
56120 C WRITE(1,200) NRU
56200 K=K+1
56300 AN=STACK(K)
56400 ADV=.FALSE.
56500 LI=LO
56600 GO TO 120
56610 100 X=X
56620 XMAX=XMAX
56700 IF(X.LT.XMAX) GO TO 102
56800 CALL ERROR(2)
56900 C
57000 C R U L E 10. NEW PRIMARY TREE
57100 102 X=X+1
57110 NRU=10
57120 C WRITE(1,200) NRU
57200 STACK(1)=ORDER(X)
57210 AN=STACK(1)
57300 105 LI=LO
57400 ADV=.TRUE.
57500 GO TO 120
57600 110 LI=LO
57700 LU=LO+1
57800 ADV=.TRUE.
57900 120 INIT=.TRUE.
58000 RETURN
58100 END
58200 C
58300 C - - - - -
58400 SUBROUTINE OUT1
58500 3 INCLUDE 'OUTCOM'
58600 WRITE(1,100)
58700 DO 50 I=1,TLN
58800 50 WRITE(1,200) I,INDENT(I),HEAD(I),NPE(I),PPE(I),LPE(I)
58900 C
59000 C
59100 100 FORMAT('11',T5,'SUBSCRIPT',T20,'LO P.ARG',T35,'N.ENTRIES',
59200 *T45,'P.PROV',T55,'PROV.')
59300 200 FORMAT(' ',T7,I3,T4,'I2',5X,I2,I3,T47,I3,T57,I3)
59400 RETURN
59500 END
59600 C
59700 C - - - - -
59800 C
59900 SUBROUTINE ERROR(N)
60000 GO TO (1,2),N
60100 1 WRITE(6,100)
60200 WRITE(1,100)
60300 STOP
60400 2 WRITE(6,200)
60500 WRITE(1,200)
60600 STOP
60700 100 FORMAT(' INCORRECT INPUT--- PROGRAM EXECUTION STOPPED. ')
60800 200 FORMAT(' PROGRAM STOPPED--- SOME PROVISIONS HAVE NOT
60900 *BEEN OUTLINED')
61000 RETURN
61100 END
61200 C
61300 C - - - - -
61400 C
```

```

61500      BLOCK DATA
61600      $ INCLUDE 'OUTCOM'
61700      DATA BLANK/6H      /,BL/4H      /,DOT/4H.      /
61800      END
61810      C
61820      C - - - - -
61830      C
61900      SUBROUTINE OUT2
62000      $INCLUDE "OUTCOM"
62010      WRITE(Z,1050)
62100      WRITE (1,1000)
62110      WRITE(Z,1060)
62200      DO 60 I=1,TLN
62300      NBL=INDEXT(I)
62400      WRITE(Z,2000)(BL,K=1,NBL)
62510      GO TO (10,20,30,40,50),NBL
62511      10 WRITE(Z,100) (ARG(HEAD(I),J),J=1,5)
62512      GO TO 55
62513      20 WRITE(Z,200) (ARG(HEAD(I),J),J=1,5)
62514      GO TO 55
62515      30 WRITE(Z,300) (ARG(HEAD(I),J),J=1,5)
62516      GO TO 55
62517      40 WRITE(Z,400) (ARG(HEAD(I),J),J=1,5)
62518      GO TO 55
62519      50 WRITE(Z,500) (ARG(HEAD(I),J),J=1,5)
62700      C
62800      55 J=PPE(I)
62900      IF(J.EQ.0) GO TO 60
63000      K=J+NPE(I)-1
63100      DO 56 L=J,K
63110      NBL1=NBL+1
63120      WRITE(Z,2000) (BL,N=1,NBL),(OUT,M=NBL1,10)
63200      56 WRITE(Z,4000) LPE(L),(PRDV(LPE(L),M),M=1,5)
63400      60 CONTINUE
63402      WRITE(Z,4500)
63405      1050 FORMAT("1",T6,"J U T L I N E")
63410      1000 FORMAT("0",T6,"H E A D I N G",T40,"P R O V I S I O N")
63412      1060 FORMAT(" ",T6,13(1H-),T40,17(1H-),///)
63415      2000 FORMAT(" ",30A4)
63430      4000 FORMAT("+",T40,12,2X,5A6)
63440      4500 FORMAT("1")
63650      100 FORMAT("+",T6,5A6)
63700      200 FORMAT("+",T10,5A6)
63800      300 FORMAT("+",T14,5A6)
63900      400 FORMAT("+",T18,5A6)
63950      500 FORMAT("+",T22,5A6)
63955      C
63960      C
63965      RETJRN
63970      END
145000      C

```

REFERENCES

- 1.1 Fenves, S.J., Computer Methods in Civil Engineering, Prentice Hall, Englewood Cliffs, N.J., 1967.
- 1.2 Pollack, S.L., Decision Tables: Theory and Practice, Wiley-Interscience, New York, New York, 1971.
- 1.3 Montalbano, M., "Tables, Flow Charts, and Program Logic," IBM Systems Journal, Sept. 1962, pp. 51-63.
- 1.4 Knuth, D.E., The Art of Computer Programming, Vol. I, Fundamental Algorithms, Addison-Wesley, New York, New York, 1968.
- 2.1 Melin, J.W., CRAM User's Manual, Civil Engineering Systems Lab, University of Illinois, Urbana, Illinois, April 1971.