UILU-ENG-75-2031

CIVIL ENGINEERING STUDIES

STRUCTURAL RESEARCH SERIES NO. 425

TECHNOLOGY FOR THE FORMULATION AND EXPRESSION OF SPECIFICATIONS

VOLUME III: TECHNICAL REFERENCE MANUAL

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

UILU-ENG-74-2027

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

By

Akenori Shibata

and

Mete 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, β = 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.

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

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 streps:

(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.

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

where $(ES)_{si}$ 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-Θ 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.

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

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:

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

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

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).

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

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.

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

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.

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

where F_i = design force in element i, $F_{i\,rss}$ = 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 μ = 1.0 for columns and μ = 6.0 for beams. Let $E = 3.6 \times 10^6$ psi $= 2.5 \times 10^4$ 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.

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

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

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.

	Mode 1	Mode 2	Mode 3
$\Sigma P_{girders}/\Sigma P_{girders + col.}$	0.55	0.21	0.04
$\Sigma P_{columns}/\Sigma P_{girders + col.}$	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.

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

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

$$\nu_1 = 0.77 * 0.48 = 0.37$$

$$\nu_2 = 0.18 * 1.4 = 0.25$$

$$\nu_3 = 0.053 * 0.92 = 0.049$$

$$\nu_{abs} = 0.37 + 0.25 = 0.62$$

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

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 kip-ft = 760 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.

	Mode 1	Mode 2	Mode 3
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, ν, 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.

<u>Test Results</u>. 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 GI29934 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., "Sinusiodal 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. II 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; $-_{rss}$ 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

ν = ratio of base shear to weight of structure; $-_m$ for mode m; $-_{abs}$ maximum for sum of any two ν_m; $-_{rss}$ Square root of sum of ν_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 / Total Weight	Top Story Shear / Base Shear	Base Shear / Total Weight	Top Story Shear / 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

[a]All stories have the same stiffness

[b]Stiffness 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

	Period			Damping		Period			Damping
Mode	Uncracked[a]	Cracked[b]	Substitute[c]	Factor[d]	Mode	Uncracked[a]	Cracked[b]	Substitute[c]	Factor[d]
	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-Story Frame				5	0.075	0.11	0.14	0.032
1	0.53	0.85	1.58	0.099	6	0.056	0.083	0.093	0.027
2	0.17	0.26	0.41	0.068	7	0.043	0.063	0.068	0.024
3	0.090	0.14	0.18	0.041	8	0.035	0.051	0.053	0.022
4	0.059	0.087	0.097	0.028	9	0.030	0.043	0.044	0.021
5	0.046	0.065	0.067	0.022	10	0.028	0.039	0.040	0.020

[a]Based on gross plain section

[b]Based on transformed cracked section (assumed).

[c]Based on stiffness properties from Eq. 2 with $\mu = 6.0$ for beams and $\mu = 1.0$ for columns.

[d]From 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

	Calculated Damage Ratios		
	Frame X	Frame Y	Frame Z
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

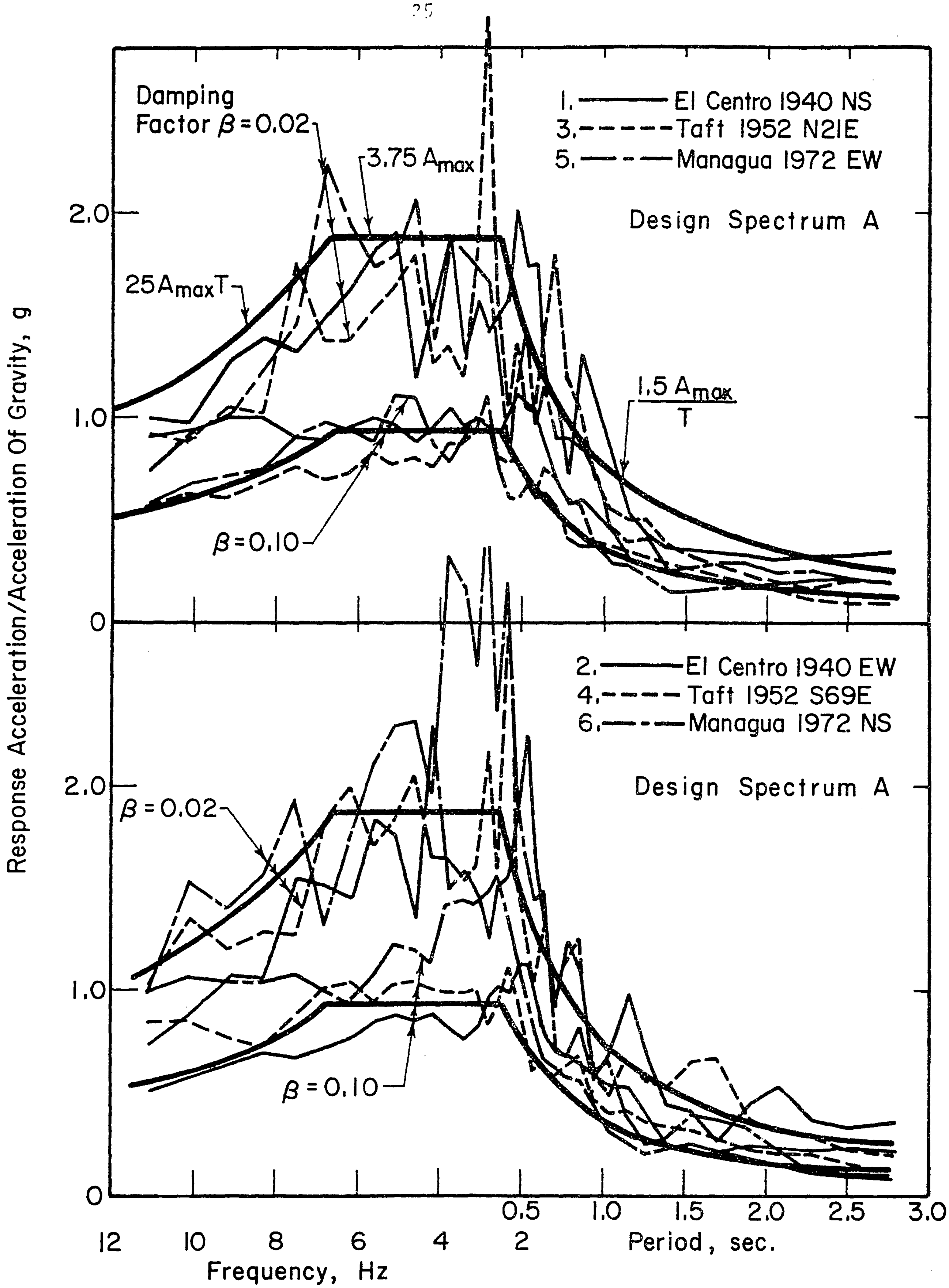


Fig. 1 Acceleration Response to Ground Motions 1 through 6 (Normalized to 0.5g) and Design Acceleration-Response Spectrum A.

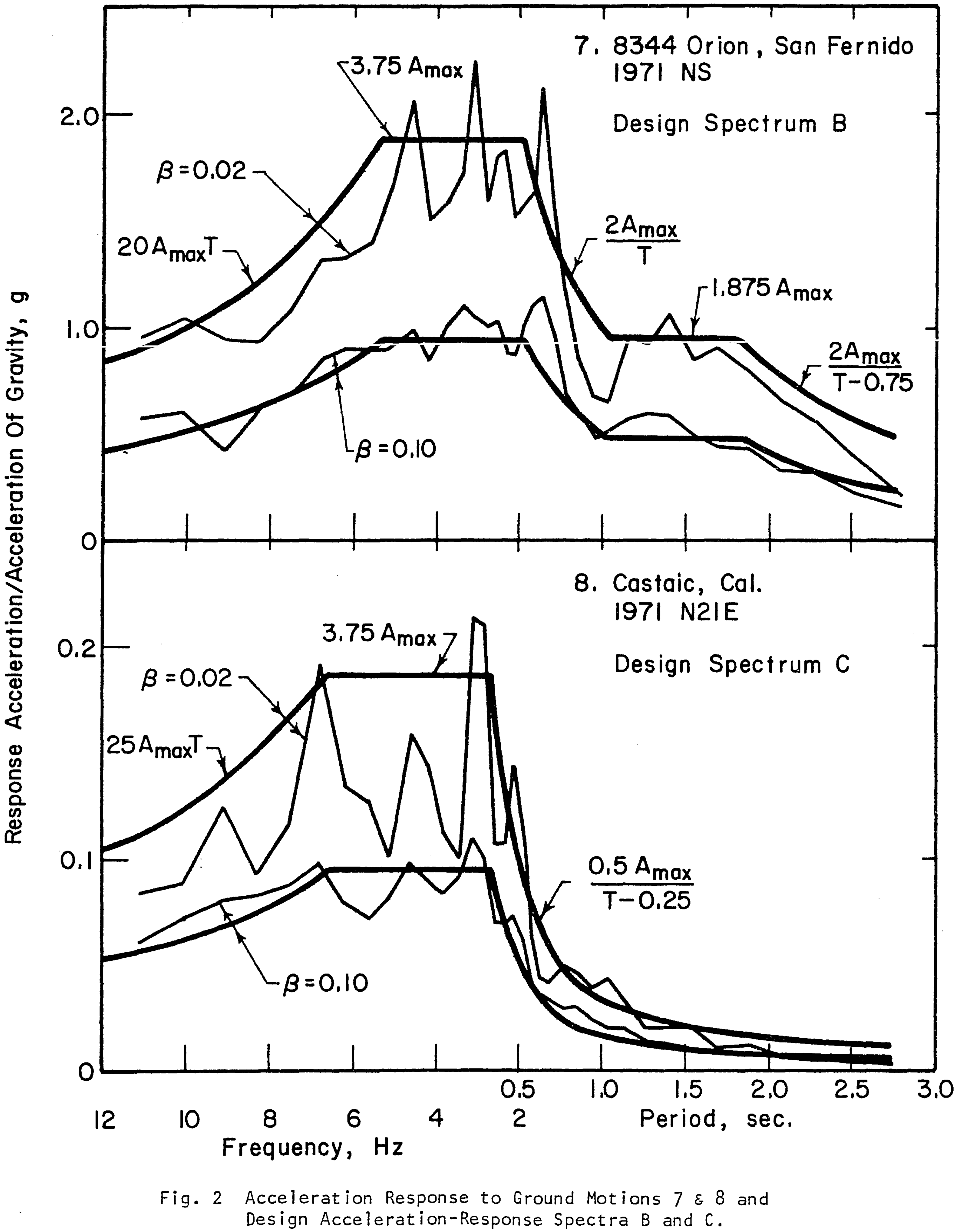


Fig. 2 Acceleration Response to Ground Motions 7 & 8 and Design Acceleration-Response Spectra B and C.

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	N21E	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	N21E	0.32

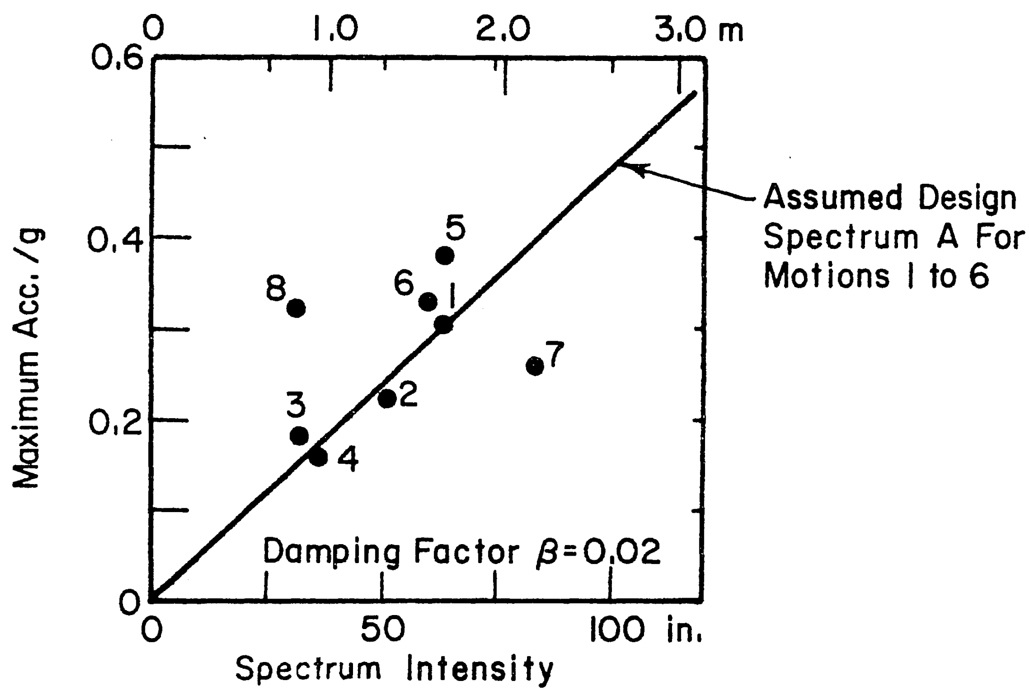


Fig. 3 Comparison of Maximum Acceleration with Spectrum Intensity

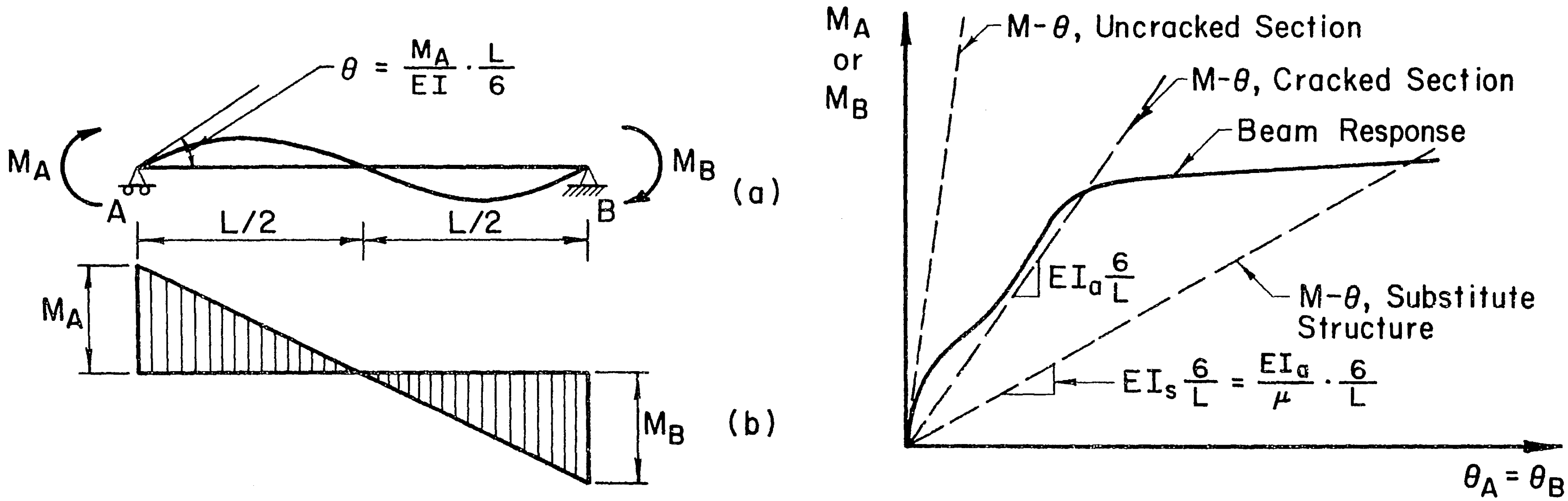


Fig. 4 Interpretation of Damage Ratio

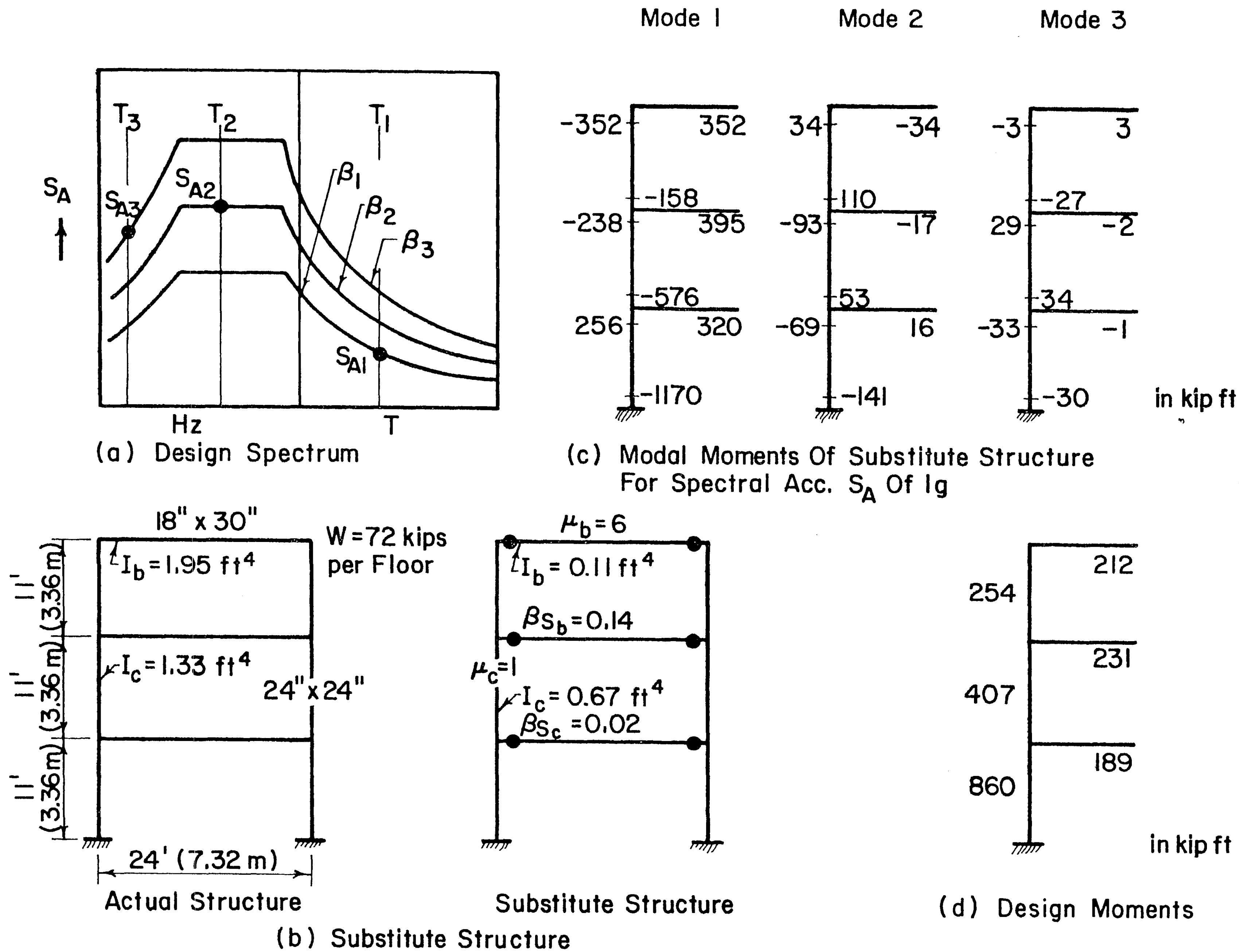


Fig. 5 Data for Numerical Example (1.0 kip = 4.45 kN;
1.0 ft = 0.30m; 1.0 k-ft - 1.36 kN-m)

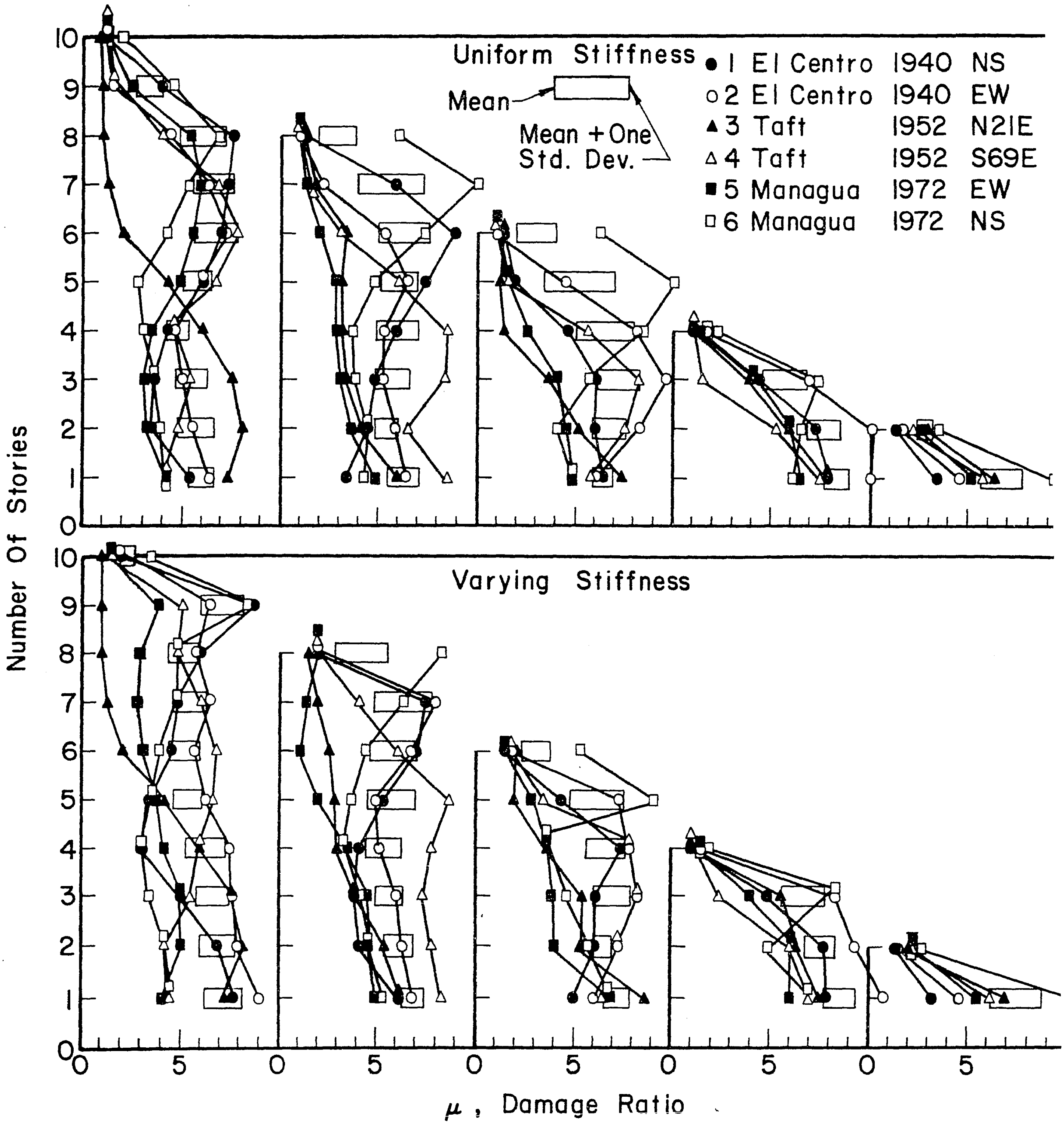


Fig. 6 Calculated Damage Ratios for Frames with Rigid Beams (Ground Motions 1 through 6)

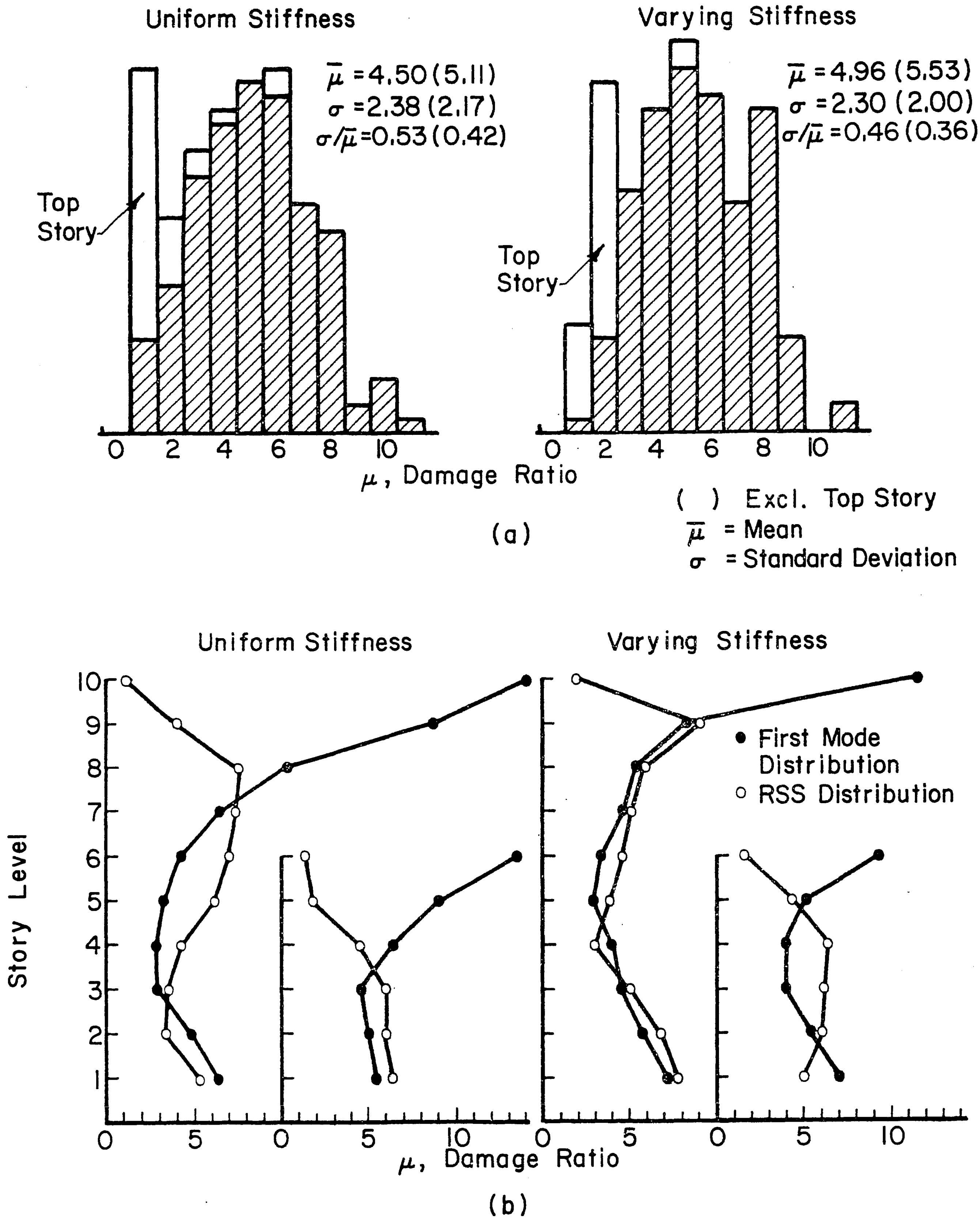


Fig. 7 Distribution of Damage Ratios and Effect of Story Shear Strength Distribution

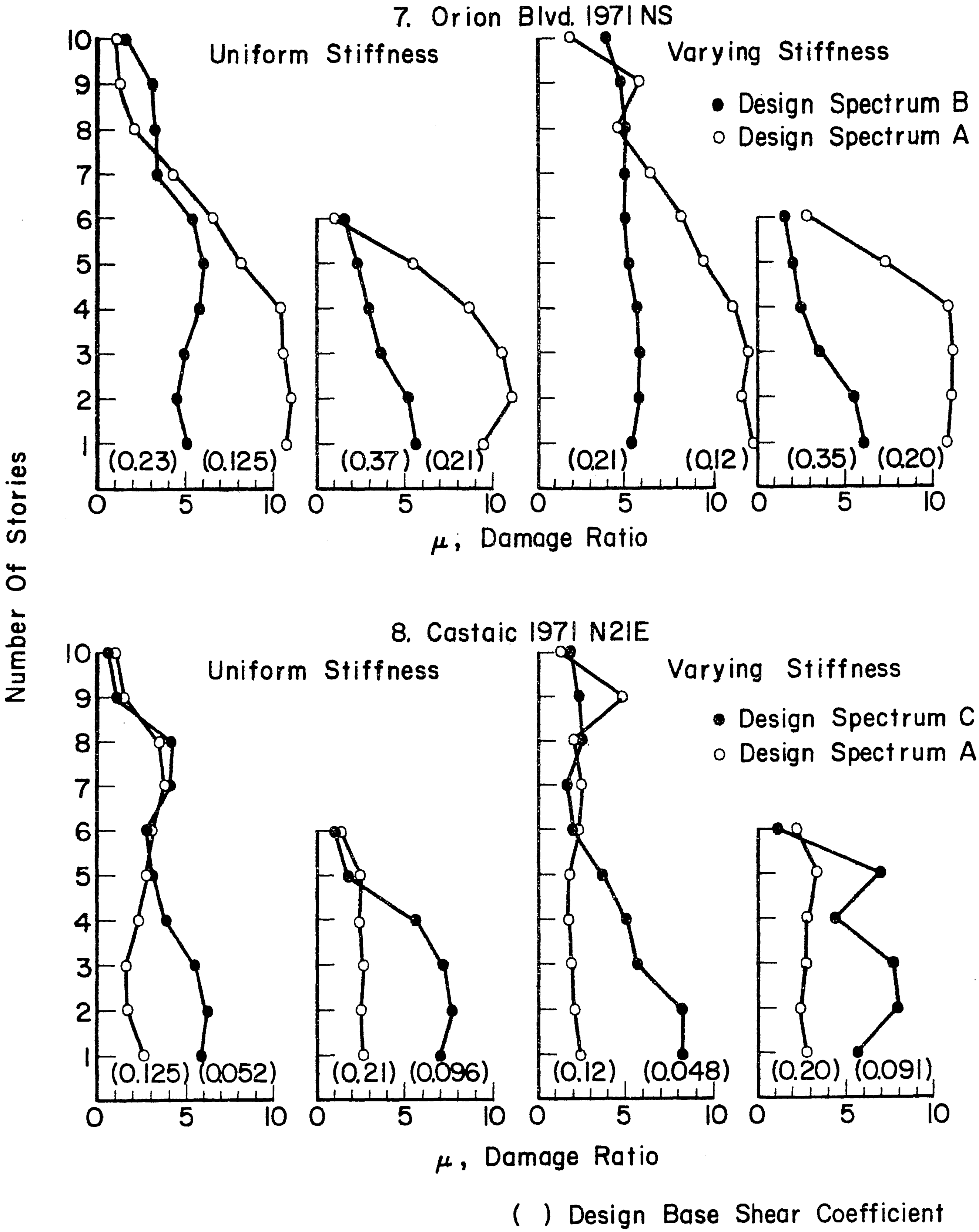


Fig. 8 Calculated Damage Ratios for Frames with Rigid Beams (Ground Motions 7 and 8)

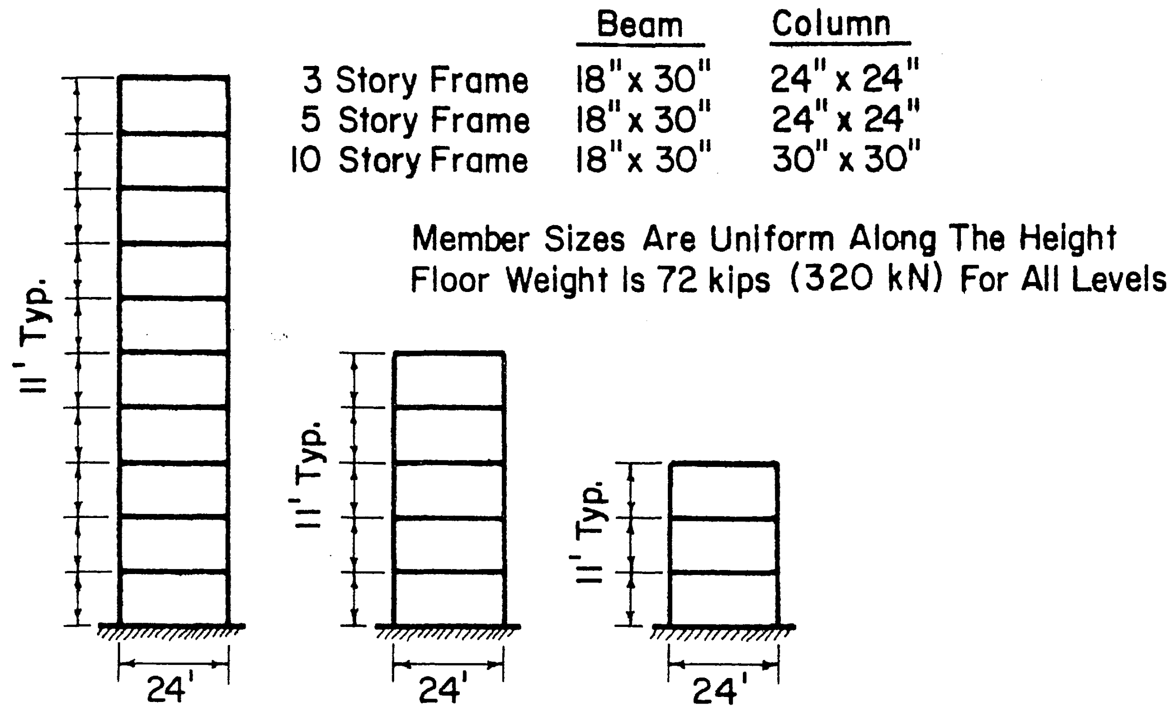


Fig. 9 Properties of Frames with Flexible Beams
(1.0 in. = 0.025m)

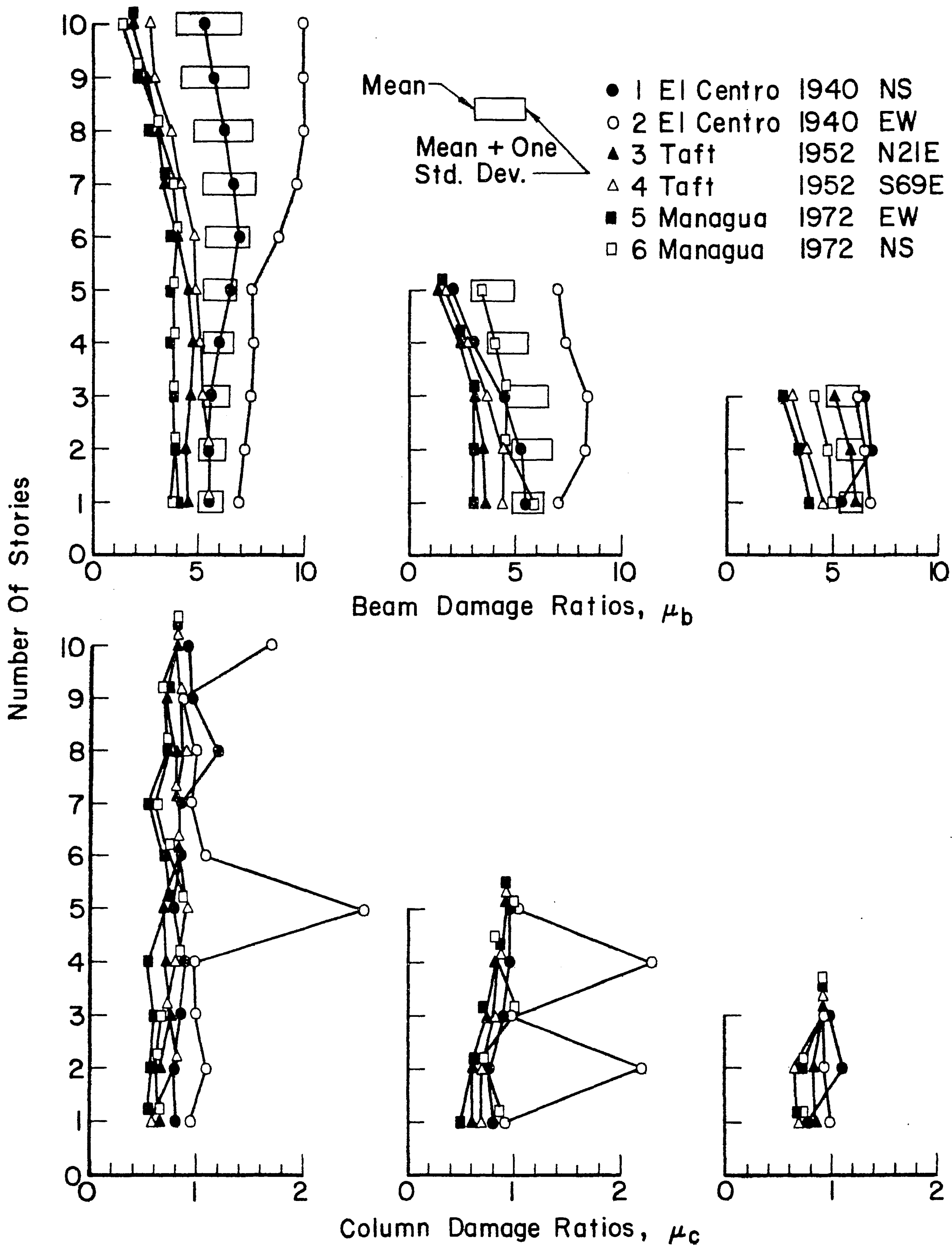


Fig. 10 Calculated Damage Ratios for Frames with Flexible Beams (Ground Motions 1 through 6)

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 reporgramming 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

	Page
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 explicity false entries amont 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.

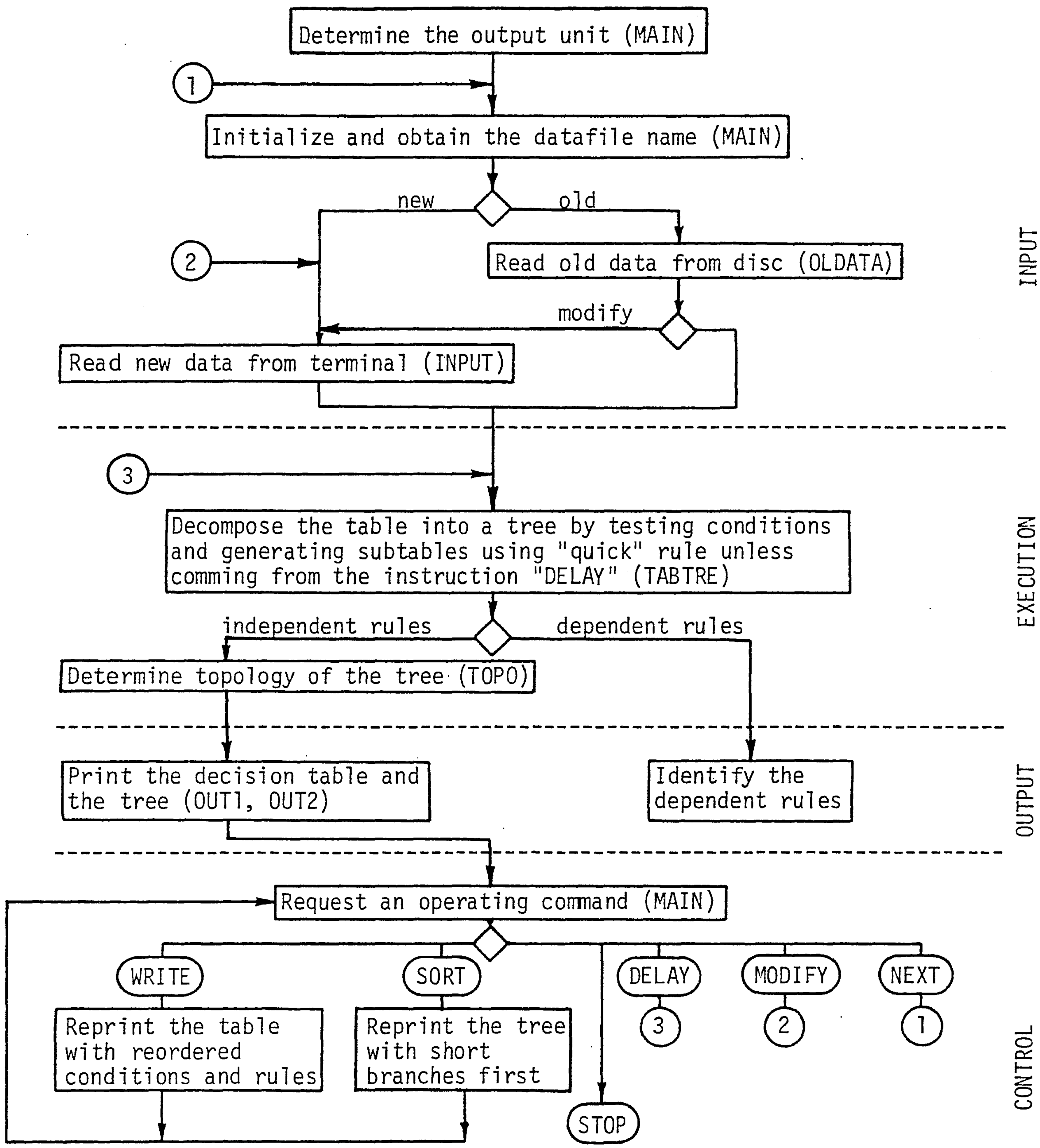
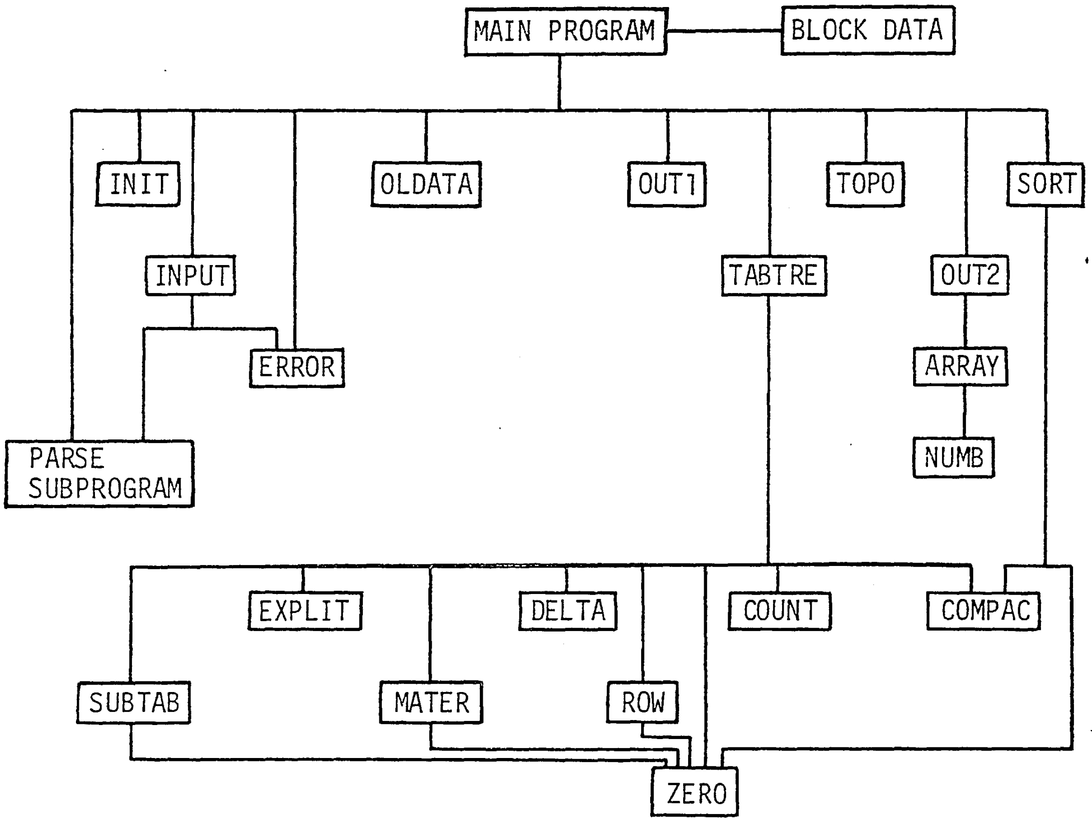


Figure 1.1   Block Diagram for the Decision Table Program



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, RDLINE, END, MATCH, FIXED, STRING, and MODE.

Figure 1.2 Subroutine Linkage (TABLE)

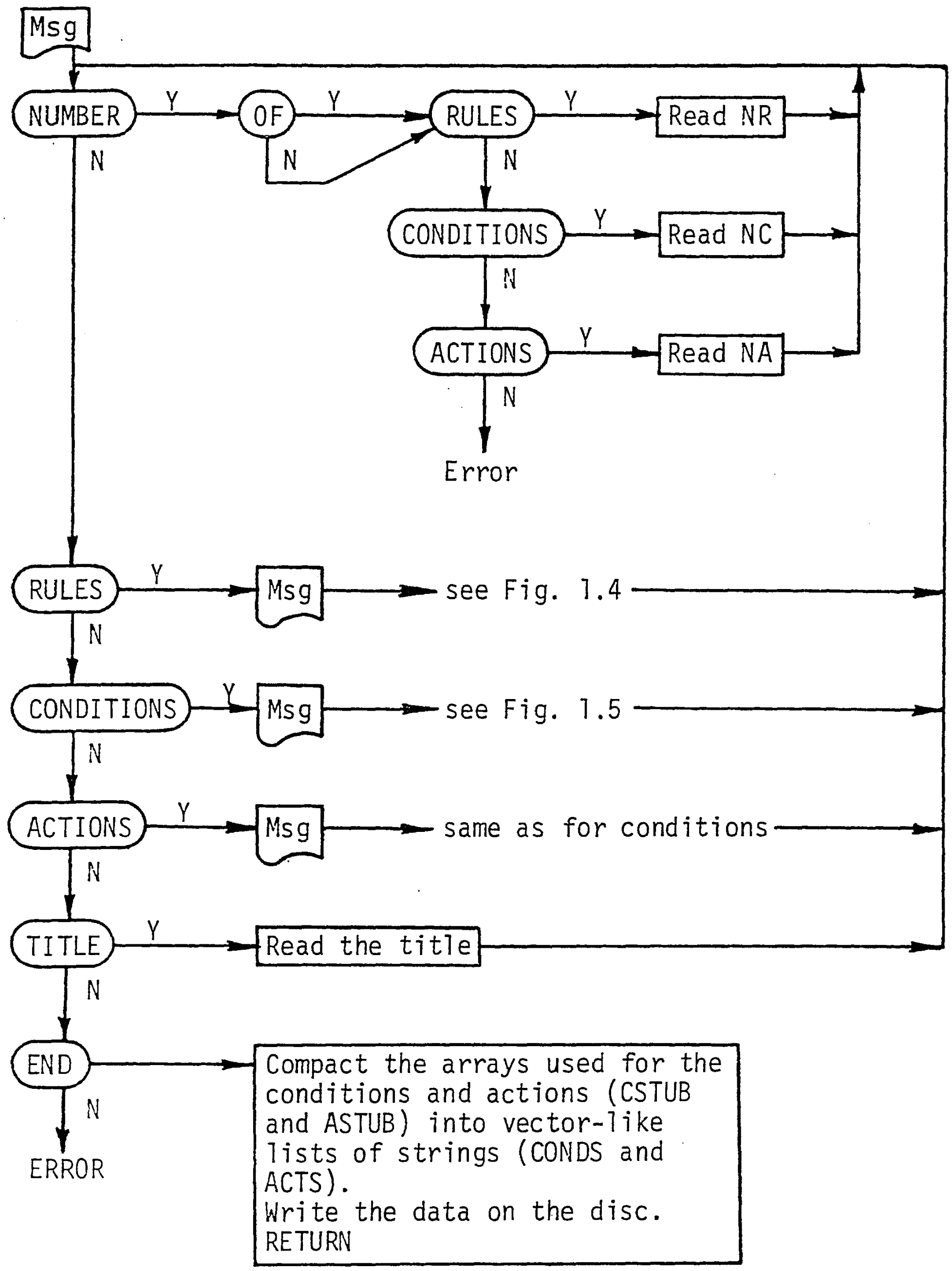


Figure 1.3 Subroutine INPUT (TABLE)

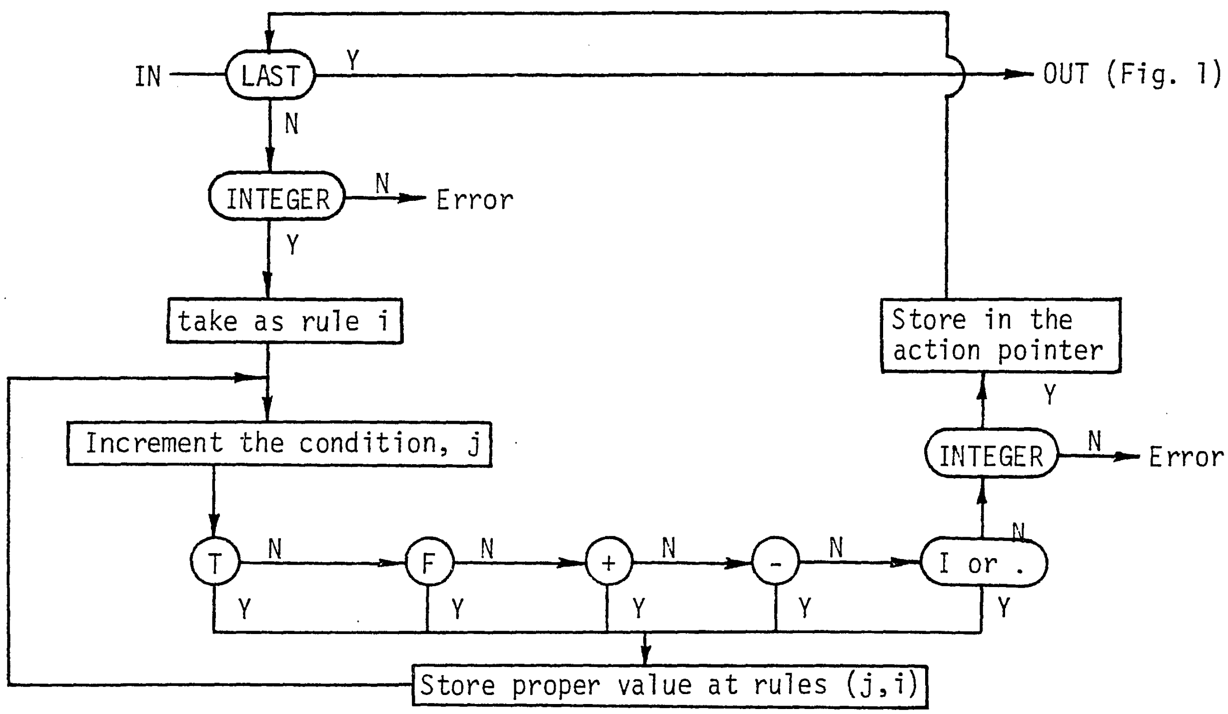


Figure 1.4 Input of the Rules (TABLE)

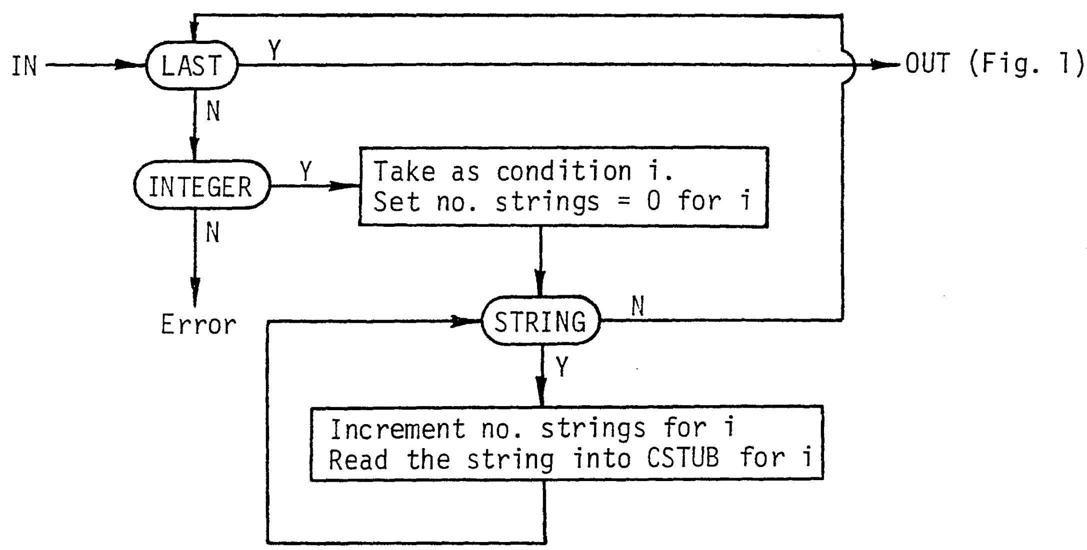


Figure 1.5 Input of the Conditions (TABLE)

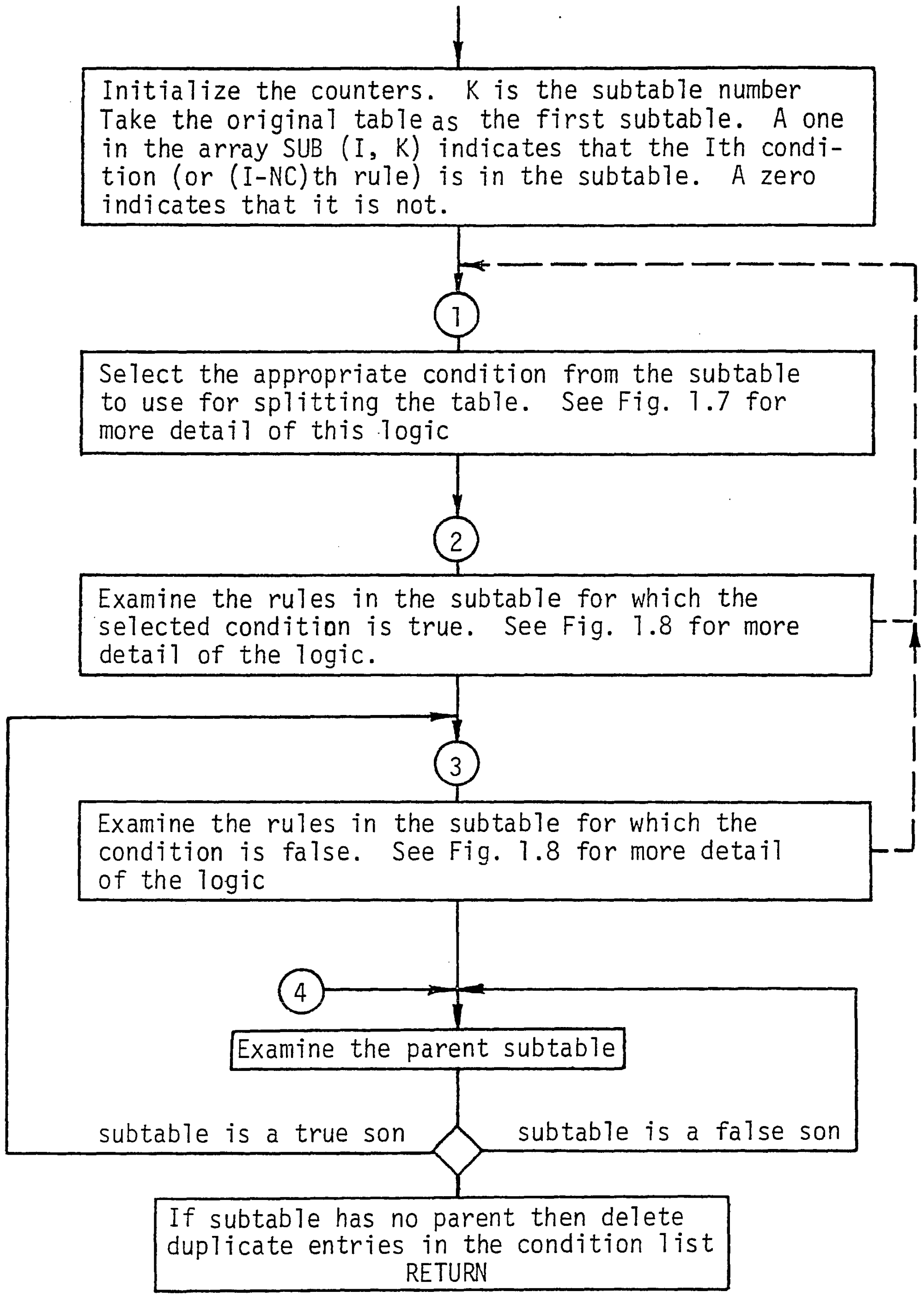


Figure 1.6 Subroutine TABTRE (TABLE)

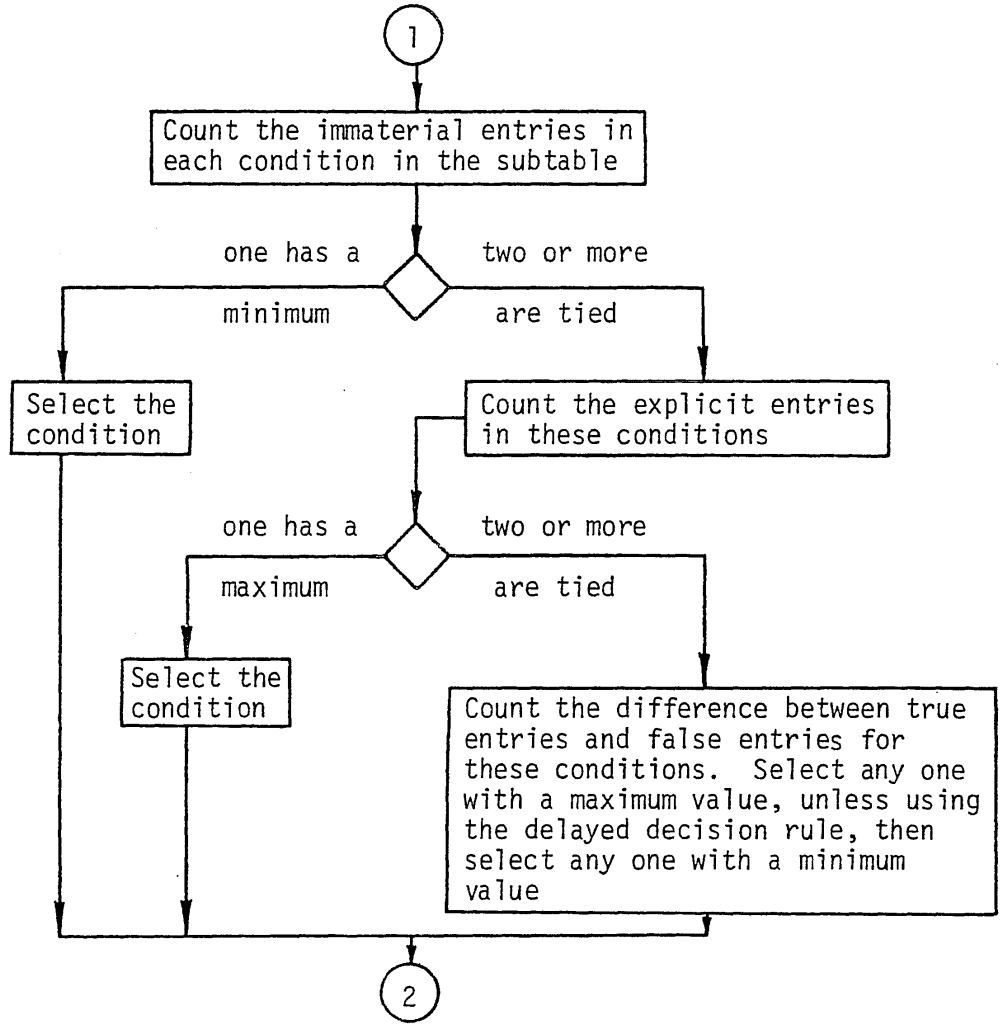
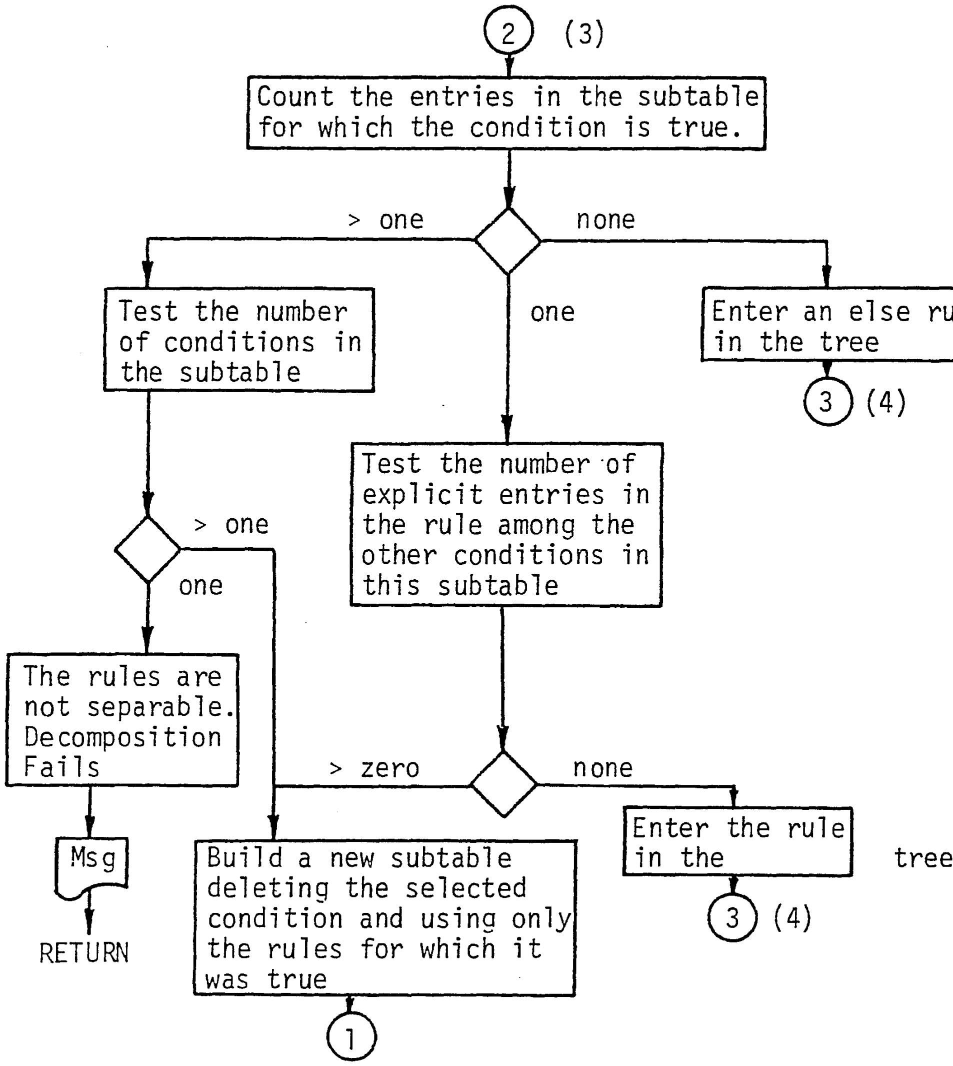


Figure 1.7 Selection of Condition in TABTRE (TABLE)



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)

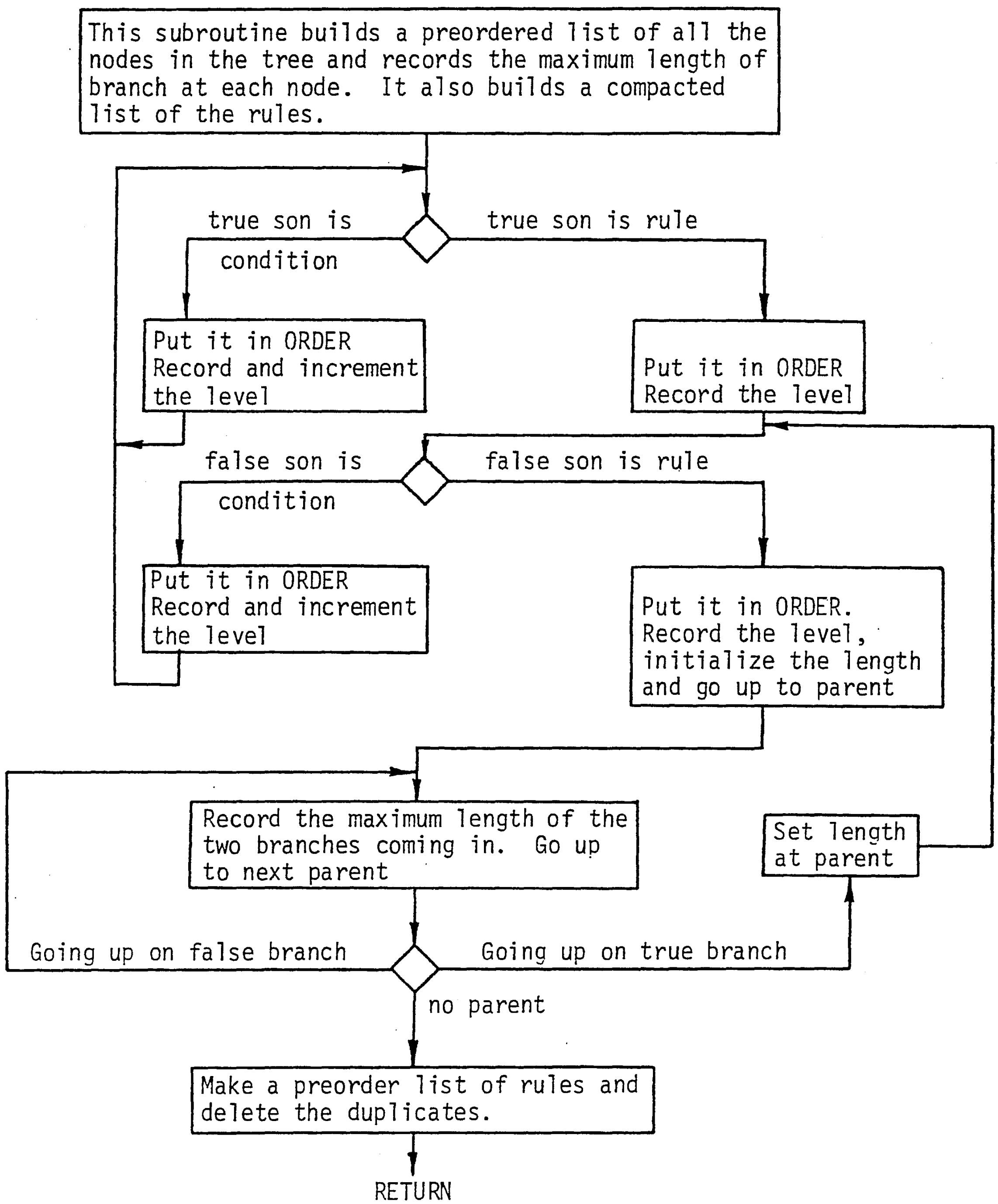


Figure 1.9 Subroutine TOPO (TABLE)

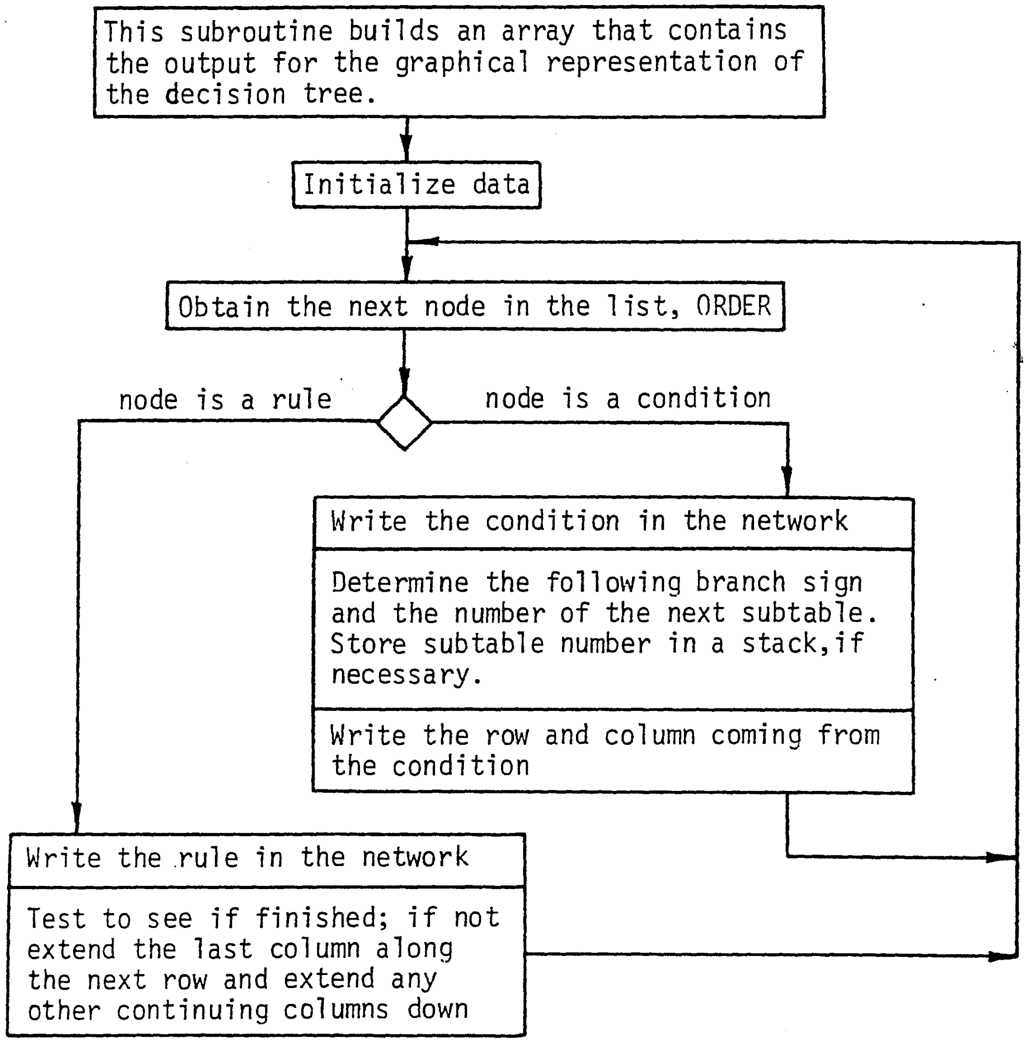


Figure 1.10 Subroutine ARRAY (TABLE)

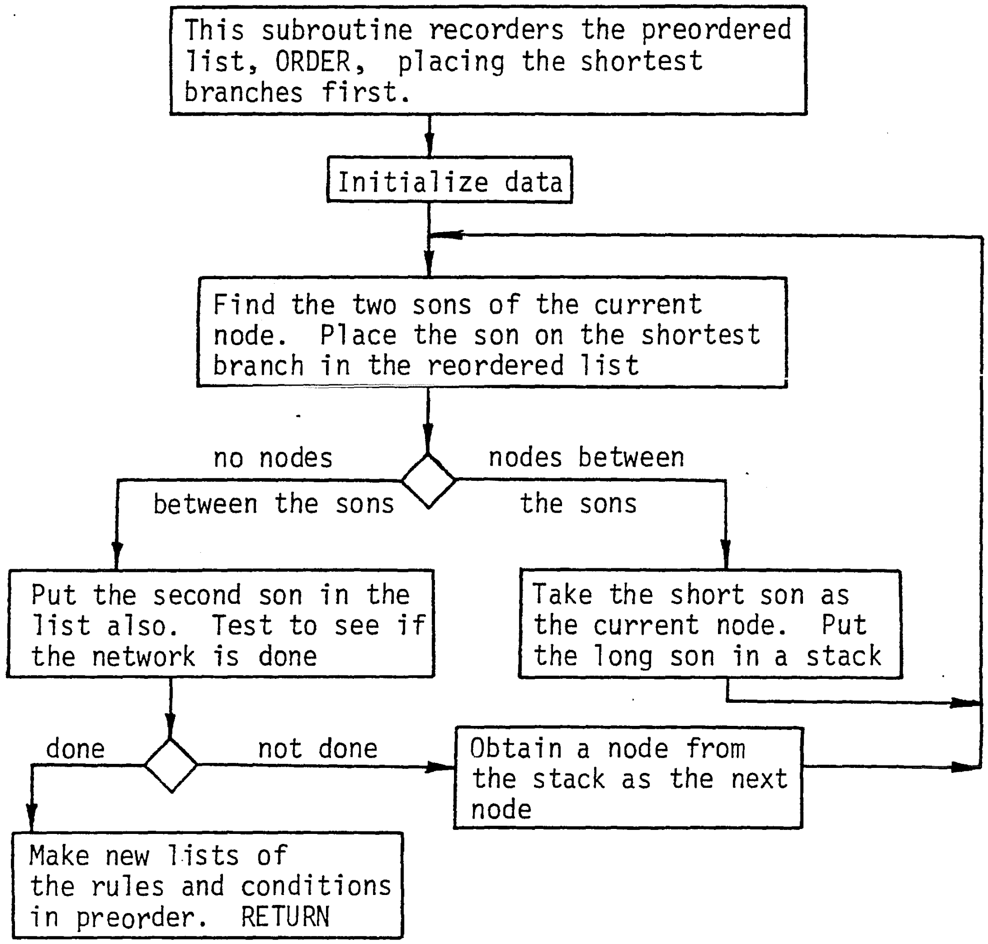


Figure 1.11 Subroutine SORT (TABLE)

1.3 Data Structure

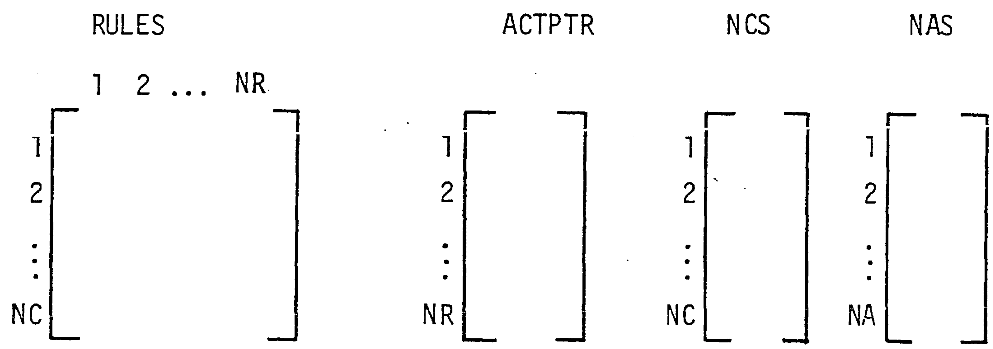
All of the manipulations in the program are based on operations with integer numbers and comparisons of logical variables. Consequently most of the data structure is composed of vectors and arrays of integers. The exceptions to this are the arrays of condition and action stubs that are stored for use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

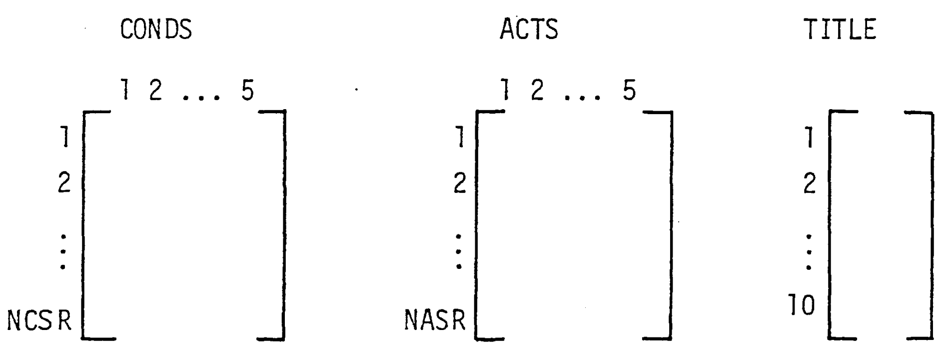
- NR - number of rules
- NC - number of conditions
- NA - number of actions
- NCSR - number of condition stub rows
- NASR - number of action stub rows

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:



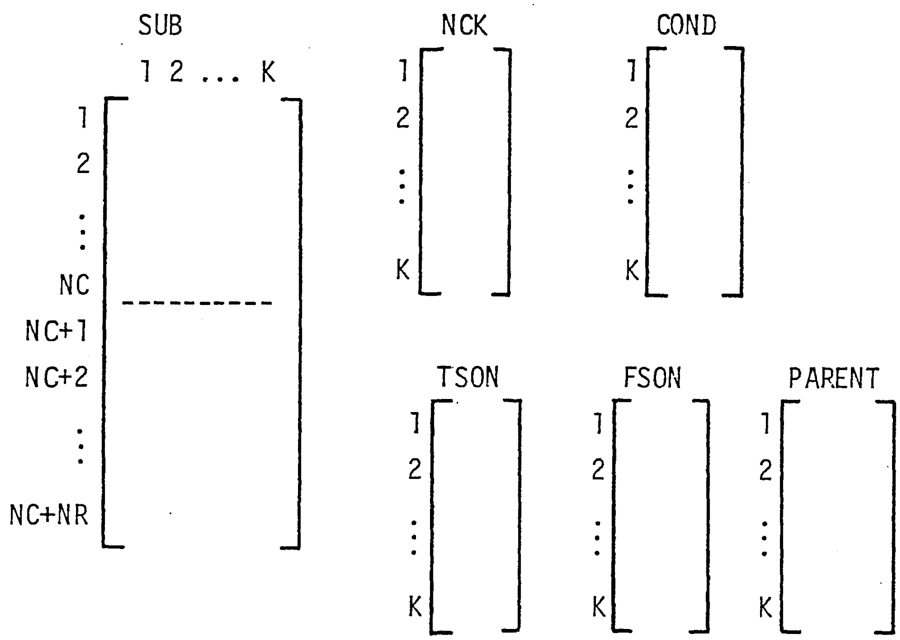
CONDS	Contains the condition stubs. Each word is six characters, so each row is 30 characters.
ACTS	Contains the action stubs
TITLE	Contains the title. Space is available for 60 characters

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 RULES, row 2, 1 to NR ⋮ RULES, row NC, 1 to NR	27I3
ACTPTR, 1 to NR	27I3
CONDS, row 1 CONDS, row 2 ⋮ CONDS, row NCSR	5A6
ACTS, row 1 ACTS, row 2 ⋮ ACTS, row NASR	5A6
NCS, 1 to NC	27I3
NAS, 1 to NA	27I3

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

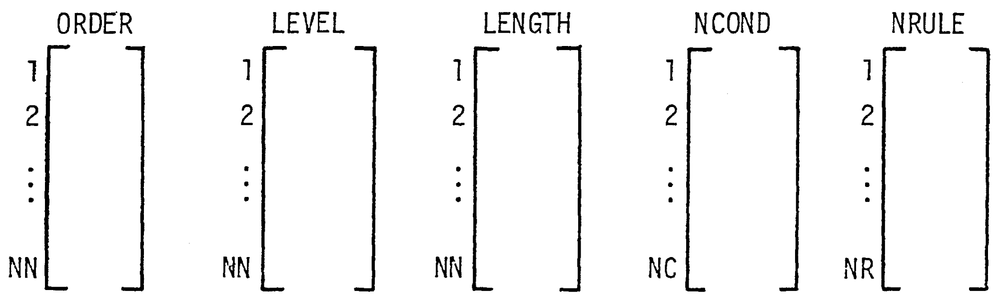
Figure 1.13 Permenent Data File (TABLE)



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 enamating 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)

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 *
ASTUB	ARRAY CONTAINING THE ACTION STUBS
B	DUMMY NAME FOR AN ARGUMENT
BKSTK	VECTOR USED IN TOPO 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 NET
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
COUNT	A SUBROUTINE
CSTUB	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 ELGIBLE CONDITIONS
DELTA	A SUBROUTINE
DIVIDE	VECTOR CONTAINING ASTERISK SYMBOLS

```
                :
DR              :   VECTOR CONTAINING THOSE CONDITIONS WITH MAXIMUM
                :   (OR MINIMUM) DC
                :
E               :   THE LETTER E
                :
EC              :   VECTOR CONTINING THE NUMBER OF EXPLICIT ENTRIES FOR THE
                :   ELGIBLE CONDITIONS
                :
EL              :   THE LETTER L
                :
END             :   LOGICAL FUNCTION IN PARSE
                :
ENTITY          :   VECTOR CONTAINING AN ITEM OF INPUT FROM PARSE
                :
ER              :   VECTOR CONTAINING THOSE CONDTITIONS WITH MAXIMUM EC
                :
ERROR           :   A SUBROUTINE
                :
EXPLIT          :   A SUBROUTINE
                :
FIXED           :   LOGICAL FUNCTION IN PARSE
                :
FLOUT           :   THE FILE NUMBER FOR OUTPUT
                :
FRULES          :   VECTOR CONTAINING THE RULES IN A SUBTABLE WITH FALSE OR
                :   IMMATERIAL ENTRIES FOR THE SELECTED CONDITION
                :
FSON            :   VECTOR CONTAINING THE SUBTABLE OR RULE THAT IS ON THE FALSE
                :   BRANCH FROM ANY SUBTABLE
                :
IC              :   VECTOR CONTAINING THE NUMBER OF IMMATERIAL ENTRIES FOR EACH
                :   CONDITION IN A SUBTABLE
                :
IMAX            :   THE NUMBER OF COLUMNS USED FOR THE OUTPUT OF THE NETWORK
                :
INIT            :   A SUBROUTINE
                :
INPUT           :   A SUBROUTINE
                :
IVALUE          :   INTEGER VALUE FROM PARSE
                :
JMAX            :   THE NUMBER OF LINES USED FOR THE OUTPUT OF THE NETWORK
                :
K               :   THE NUMBER OF THE SUBTABLE
                :
KK              :   THE NUMBER OF THE NEXT SUBTABLE
                :
LENGTH          :   VECTOR CONTAINING THE NUMBER OF STEPS TO THE RULE ALONG THE
                :   LONGEST BRANCH FROM THE NODE FOR EACH NODE IN THE NETWORK
                :
LEVEL           :   VECTOR CONTAINING THE NUMBER OF STEPS FROM THE BEGINNING TO
                :   THE NODE FOR EACH NODE IN THE NETWORK
                :
LOGIC           :   DUMMY NAME FOR AN ARGUMENT
                :
LVR             :   VECTOR CONTAINING THE COLUMN NUMBER OF EACH INCOMPLETE
                :   BRANCH IN THE NETWORK
                :
MATCH           :   LOGICAL FUNCTION IN PARSE
                :
MATER           :   A SUBROUTINE
```

```
                 :
        MINUS    :   THE NEGATIVE SYMBOL
                 :
        MODE     :   VALUE FROM PARSE INDICATING THE TYPE OF INPUT ITEM
                 :
        MR       :   VECTOR CONTAINING THE CONDITIONS WITH THE MINIMUM TC
                 :
        N        :   NC + NF
                 :
        NA       :   NUMBER OF ACTIONS
                 :
        NAS      :   VECTOR CONTAINING THE NUMBER OF STUBS FOR EACH ACTION
                 :
        NASR     :   TOTAL NUMBER OF ROWS OF ACTION STUBS
                 :
        NC       :   NUMBER OF CONDITIONS
                 :
        NCHAR    :   FROM PARSE: THE NUMBER OF CHARACTERS IN THE INPUT ITEM
                 :
        NCK      :   VECTOR CONTAINING THE NUMBER OF CONDITIONS IN THE SUBTABLE
                 :
        NCOND    :   VECTOR CONTAINING A PREORDER LIST OF THE CONDITIONS IN
                 :   THE NETWORK, DELETING REPETITIOUS ENTRIES
                 :
        NCS      :   VECOTR CONTAININGTHE NUMBER OF STUBS FOR EACH CONDITION
                 :
        NCSR     :   TOTAL NUMBER OF ROWS OF CONDITION STUBS
                 :
        NDR      :   NUMBER OF CONDITIONS WITH MAXIMUM (OR MINIMUM) DC
                 :
        NEE      :   NUMBER OF EXPLICIT ENTRIES LEFT IN THE RULE IN THE SUBTABLE
                 :
        NER      :   NUMBER OF CONDITIONS WITH MAXIMUM FC
                 :
        NET      :   ARRA FOR OUTPUT OF THE NETWORK
                 :
        NEXT     :   LOGICAL FUNCTION IN PARSE
                 :
        NF       :   NUMBER OF RULES WITH FALSE ENTRIES IN THE SUBTABLE FOR
                 :   THE SELECTED CONDITION
                 :
        NMR      :   NUMBER OF CONDITIONS WITH MINIMUM TC
                 :
        NN       :   NUMBER OF NODES IN THE NETWORK
                 :
        NR       :   NUMBER OF RULES
                 :
        NRULES   :   VECTOR CONTAINING PREORDER LIST OF THE RULES IN THE
                 :   NETWORK, DELETING REPETITIOUS ENTRIES
                 :
        NT       :   NUMBER OF RULES WITH TRUE OR IMMATERIAL ENTRIES IN THE
                 :   SUBTABLE FOR THE SELECTED CONDITION
                 :
        NWD      :   FROM PARSE: NUMBER OF WORDS IN THE INPUT ITEM
                 :
        OLDATA   :   A SUBROUTINE
                 :
        ONPACK   :   LOGICAL VARIABLE FOR TESTING THE PRESENCE OF A DATA FILE
                 :
        ORDER    :   VECTOR CONTAINING THE LIST OF NODES IN NETWORK IN PREORDER
                 :
        OUT1     :   A SUBROUTINE
```

```
                  :
OUT2              :    A SUBROUTINE
                  :
PARCOM            :    FILE CONTAINING COMMON DECLARATIONS FOR PARSE
                  :
PARENT            :    VECTOR CONTAINING THE NUMBER OF THE SUBTABLE FROM WHICH
                  :    THE SUBTABLE IS DERIVED
                  :
PLUS              :    THE ADDITION SYMBOL
                  :
POINT             :    VARIABLE CARRYING THE SYMBOLS PLUS OR MINUS
                  :
PRT               :    CARRIES THE LETTER OFF
                  :
PTRBK             :    VECTOR USED AS A POINTER BACK TO THE PARENT NODE IN TABLE
                  :
QUICK             :    LOGICAL VARIABLE THAT IS TRUE UNLESS THE NETWORK IS BEING
                  :    FORMED WITH THE DELAYED RULE
R                 :    THE LETTER R
                  :
RDLINE            :    SUBROUTINE IN PARSE
                  :
ROW               :    A SUBROUTINE
                  :
RULES             :    ARRAY CONTAINING THE CONDITION ENTRY
                  :
S                 :    THE LETTER S
                  :
SETOUT            :    SUBROUTINE IN PARSE
                  :
SIGN              :    VECTOR CONTAINING THE SYMBOLS USED IN PRINTING THE
                  :    CONDITION ENTRY
                  :
SKC               :    VECTOR CONTAINING A STACK OF SUBTABLE NUMBERS WITH
                  :    UNUSED SONS
                  :
SORT              :    A SUBROUTINE
                  :
STRING            :    FROM PARSE: ANY INPUT ITEM ENCLOSED IN QUOTES
                  :
SUB               :    THE ARRAY CONTAINING THE DATA FOR INCLUSION OF CONDITIONS
                  :    AND RULES IN THE SUBTABLES
                  :
SUBTAB            :    A SUBROUTINE
                  :
SWITCH            :    LOGICAL VARIABLE FOR THE SUCCESS OF THE DECOMPOSITION
                  :
SYMBOL            :    VECTOR CONTAINING THE SYMBOLS FOR THE TEN NUMERALS
                  :
TARCOM            :    FILE CONTAINING THE COMMON DECLARATIONS
                  :
TARTRE            :    A SUBROUTINE
                  :
TARTWO            :    A LOGICAL VARIABLE FOR USE IN PRINTING REORDERED VERSION
                  :    OF THE TABLE
                  :
TITLE             :    THE TITLE OF THE TABLE
                  :
TOPO              :    A SUBROUTINE
                  :
TRULES            :    VECTOR CONTAINING THOSE RULES WITH TRUE OR IMMATERIAL
                  :    ENTRIES IN THE SUBTABLE FOR THE SELECTED CONDITION
```

```
            :
TSON        :   VECTOR CONTAINING THE SUBTABLE OR RULE THAT IS ON THE TRUE
            :   BRANCH FROM THE SUBTABLE
            :
VALUE       :   REAL NUMBER FROM PARSE
            :
WDINIT      :   SUBROUTINE IN PARSE
            :
XX          :   THE LETTER X
            :
Y, Z        :   DMMY NAMES FOR ARGUMENTS
            :
ZFED        :   A SUBROUTINE
```

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
C
 $ INCLUDE 'TABCOM'
C
C
C       THE FOLLOWING LINES ARE A LISTING OF TABCOM
C
CC      THIS FILE DIMENSIONS ALL THE ARRAYS USED IN THE PROGRAM
CC      AND ASSIGNS THEM TO COMMON.
CC
C       INTEGER RULES(30,30), ACTPTR(30), SUB(60,60), COND(60),
C      1  TSON(60), FSON(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), FLOUT,
C      4  NRULE(30), NCOND(30), CVB(15), PTRBK(60), BKSTK(60), CK(30),
C      5  NCS(30), NAS(30)
CC
C       REAL TITLE(10), CONDS(5,40), ACTS(5,40), SIGN(5),
C      1  SYMBOL(10), NET(132,120), DIVIDE(132), ACTION(30), MINUS
CC
C       LOGICAL SWITCH, QUICK, BRANCH, UNPACK, TABTWO
CC
C       COMMON/INT/DF, NR, NC, NA, RULES, ACTPTR, SUB, K, KK, N,
C      1  COND, TSON, FSON, 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, FLOUT, IMAX, NCS, NAS, NCSR, NASR
CC
C       COMMON/REAL/TITLE, CONDS, ACTS, SIGN, SYMBOL, NET, DIVIDE,
C      1  ACTION, BLANK, ASTER, XX, C, E, R, PLUS, MINUS, EL, S
CC
C       COMMON/LOG/SWITCH, QUICK, BRANCH, TABTWO
CC
CC      END OF TABCOM
C
C
 $ INCLUDE 'PARCOM'
C
C       THE FOLLOWING LINES ARE A LISTING OF PARCOM
C
CC      THIS FILE LISTS THE DECLARATIONS FOR USE OF THE INPUT SCANNER
CC
```

```
C       LOGICAL END, FIXED, MATCH, STRING, NEXT
C       COMMON/SCANER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
C       EQUIVALENCE (IVALUE,VALUE)
CC
CC      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
        CALL WDINIT(10,72,PRI)
C
C       READ THE OUTPUT UNIT IN
        FLOUT = 6
        WRITE(FLOUT,1005)
        CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
        IF(.NOT.MATCH("P",1)) GO TO 10
        FLOUT = 1
C
C       CHECK TO SEE IF THE ECHO IS DESIRED
C
        WRITE(6,1200)
        CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
        IF(MATCH("YES",3)) CALL WDINIT(10,72,"ON     ")
        CALL SETOUT(FLOUT)
C
C       INITIALIZE THE ARRAYS
     10 CALL INIT
C
C       READ THE DATA FILE NAME IN
     20 I = 0
        WRITE(6,1000)
        CALL RDLINE
        ENTITY(NWD+1)=6H.
        CLOSE (3)
        CHANGE(3,TITLE=ENTITY)
        DF=3
C
C       CHECK FOR EXISTING DATAFILE
        INQUIRE(3, RESIDENT=ONPACK)
        IF(.NOT. ONPACK) GO TO 50
C
C       EXISTING FILE, DOUBLE CHECK THE NAME
        WRITE(6,1010)
        CALL RDLINE
        IF(MATCH("YES",3)) GO TO 30
        GO TO 20
C
C       USE EXISTING FILE
     30 CALL OLDATA
        GO TO 60
C
C       READ NEW NETWORK DATA IN FROM THE TERMINAL
     50 CALL INPUT
        GO TO 70
C
C       CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
     60 WRITE(6,1020)
        CALL RDLINE
        IF(.NOT.MATCH("YES",3)) GO TO 70
```

```
      CALL INPUT
C
C     WRITE OUT THE ORIGINAL TABLE
   70 IF(FLOUT .EQ. 6) WRITE(6,999)
      TABTWO = .FALSE.
      CALL OUT1
C
C     DECOMPOSE THE DECISION TABLE INTO THE NETWORK
   75 QUICK = .TRUE.
      CALL TABTRE
      IF(.NOT.SWITCH) GO TO 110
      IF(FLOUT.EQ.6) WRITE(6,999)
      WRITE(6,1040)
C
C     DETERMINE THE TOPOLOGY OF THE DERIVED NETWORK
      CALL TOPO
C
C     PRINT OUT THE DATA
      IF(FLOUT .EQ. 6) WRITE(6,999)
      WRITE(FLOUT,1050)
      CALL OUT2
      IF(FLOUT .EQ. 6) WRITE(6,999)
C
C     BEGIN SCANNING FOR OPERATING COMMANDS
C
      I = 0
  100 WRITE(6,1030)
      CALL RDLINE
  110 IF(END(X)) GO TO 100
      IF(MATCH("WRITE",3)) GO TO 140
      IF(MATCH("MODIFY",3)) GO TO 150
      IF(MATCH("SORT",3)) GO TO 160
      IF(MATCH("DELAY",3)) GO TO 170
      IF(MATCH("NEXT",3)) GO TO 10
      IF(MATCH("STOP",3)) STOP
      I = I  + 1
      CALL ERROR(I)
      GO TO 100
C
C     WRITE OUT THE REARRANGED TABLE
C
  140 WRITE(FLOUT,1060)
      TABTWO = .TRUE.
      CALL OUT1
      IF(FLOUT .EQ. 6) WRITE(6,999)
      GO TO 100
C
C     MODIFY THE BASIC INPUT DATA
C
  150 CALL INIT
      CALL OLDATA
      CALL INPUT
      I = 0
      GO TO 70
C
C     SORT THE BRANCHES BY LENGTH
  160 CALL SORT
      IF(FLOUT .EQ. 6) WRITE(6,999)
      WRITE(FLOUT,1070)
      CALL OUT2
      IF(FLOUT .EQ. 6) WRITE(6,999)
      I = 0
```

```
      GO TO 400
C
C     DECOMPOSE THE NETWORK USING THE DELAYED DECISION RULE
  170 QUICK = .FALSE.
      CALL INIT
      CALL OLDATA
      CALL TABIRE
      IF(.NOT.SWITCH) GO TO 110
      WRITE(6,1040)
      CALL TOPU
      IF(FLOUT .EQ. 6) WRITE(6,999)
      WRITE(FLOUT,1090)
      CALL OUT2
      IF(FLOUT .EQ. 6) WRITE(6,999)
      I = 0
      GO TO 100
C
C
  999 FORMAT(1H ,/,1H ,/)
 1000 FORMAT(1H ,"ENTER THE DATA FILE NAME",/)
 1005 FORMAT(1H , "ENTER P FOR OUTPUT ON THE ON-SITE PRINTER,",
     1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL",/)
 1010 FORMAT(1H , "DATA FILE EXISTS WITH THIS NAME.",/,
     1  " DO YOU WANT TO USE IT?",/)
 1020 FORMAT(1H ,"DO YOU WANT TO MODIFY THE EXISTING DATA?",/)
 1030 FORMAT(1H ,"ENTER A PROGRAM COMMAND",/)
 1040 FORMAT(1H , "DECISION NETWORK SUCCESSFULLY COMPLETED",/)
 1050 FORMAT(1H1, "DERIVED DECISION NETWORK",//)
 1060 FORMAT(1H1,"DECISION TABLE WITH CONDITIONS AND RULES IN PREORDER",
 1070 FORMAT(1H1,"DECISION NETWORK SORTED BY BRANCH LENGTH",//)
 1080 FORMAT(1H1,"DECISION TABLE WITH CONDITIONS AND ",/,
     1 " RULES IN THE PREORDER OF THE SORTED NETWORK")
 1090 FORMAT(1H1,"DECISION NETWORK DERIVED WITH THE ",
     1 " DELAYED DECISION RULE",//)
 1100 FORMAT(1H1,"DECISION TABLE WITH CONDITIONS AND ",/,
     1 " RULES IN THE PREORDER OF THE DELAYED NETWORK")
 1200 FORMAT(" DO YOU WANT THE PRINTOUT OF THE INPUT?",/)
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INIT
$ INCLUDE 'TABCOM'
      DO 10 I = 1, 60
      DO 10 J = 1, 60
   10 SUB(J,I) = 0
      DO 20 I = 1, 60
   20 COND(I) = 0
      DO 30 I = 1, 60
   30 TSON(I) = 0
      DO 40 I = 1, 60
   40 FSON(I) = 0
      DO 50 I = 1, 60
   50 PARENT(I) = 0
      DO 60 I = 1, 60
   60 MCK(I) =  0
      DO 70 I = 1, 60
   70 ORDER(I) = 0
      DO 80 I = 1, 60
   80 LEVEL(I) = 0
      DO 90 I = 1, 60
   90 LENGTH(I) = 0
```

```
      DO 100 I = 1, 30
  100 NCS(I) = 1
      DO 110 I = 1, 30
  110 NAS(I) = 1
      DO 120 I = 1, 40
      DO 120 J = 1, 5
  120 ACTS(J,I) = BLANK
      DO 130 I = 1, 40
      DO 130 J = 1, 5
  130 CONDS(J,I) = BLANK
      DO 140 I = 1, 132
  140 DIVIDE(I) = ASTER
      DO 150 I = 1, 10
  150 TITLE(I) = BLANK
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INPUT
C
$ INCLUDE 'TABCOM'
C
C
$ INCLUDE 'PARCOM'
C
C
C     CSTUB AND ASTUB ARE USED TO STORE CHANGE IN THE STUBS
      REAL CSTUB(50,30), ASTUB(50,30)
      DO 6 I = 1, 30
      DO 2 J = 1, 50
    2 CSTUB(J,I) = BLANK
      DO 4 J = 1, 50
    4 ASTUB(J,I) = BLANK
    6 CONTINUE
      WRITE(6,1000)
      I = 0
C
C     BEGIN SCANNING FOR INPUT
      CALL PARSE(ENTITY, MODE, VALUE, NCHAR)
   10 IF(MATCH("NUMBER",3)) GO TO 20
      IF(MATCH("RULES",3)) GO TO 70
      IF(MATCH("CONDIT",3)) GO TO 160
      IF(MATCH("ACTION",3)) GO TO 200
      IF(MATCH("TITLE",3)) GO TO 240
      IF(MATCH("END",3)) GO TO 270
      IF(END(X)) GO TO 10
      I = I  + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 10
C
C     HERE WE HAVE THE NUMBER OF RULES, CONDITIONS, OR ACTIONS
   20 IF(MATCH("OF",2)) GO TO 30
   30 IF(MATCH("RULES",3)) GO TO 40
      IF(MATCH("CONDIT",3)) GO TO 50
      IF(MATCH("ACTION",3)) GO TO 60
   35 I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 10
   40 IF(.NOT.(FIXED(X))) GO TO 35
```

```
      NR = IVALUE
      I = 0
      GO TO 10
   50 IF(.NOT.(FIXED(X))) GO TO 35
      NC = IVALUE
      I = 0
      GO TO 10
   60 IF(.NOT.(FIXED(X))) GO TO 35
      NA = IVALUE
      I = 0
      GO TO 10
C
C     HERE WE HAVE THE RULES ARRAY
   70 WRITE(6,1010)
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
   80 IF(MATCH("LAST",3)) GO TO 10
      IF(FIXED(X)) GO TO 85
      I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 80
   85 I = IVALUE
      J = 0
   90 J = J + 1
      IF(MATCH("T",1) .OR. MATCH("Y",1)) GO TO 100
      IF(MATCH("F",1) .OR. MATCH("N",1)) GO TO 110
      IF(MATCH("+",1)) GO TO 120
      IF(MATCH("-",1)) GO TO 130
      IF(MATCH("I",1) .OR. MATCH(".",1)) GO TO 140
      IF(FIXED(X)) GO TO 150
      IF(END(X)) GO TO 80
      I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 80
  100 RULES(J,I) = 1
      GO TO 90
  110 RULES(J,I) = -1
      GO TO 90
  120 RULES(J,I) = 2
      GO TO 90
  130 RULES(J,I) = -2
      GO TO 90
  140 RULES(J,I) = 0
      GO TO 90
  150 ACTPTR(I) = IVALUE
      CALL RDLINE
      I = 0
      GO TO 80
C
C     HERE WE HAVE THE CONDITION STRINGS
  160 WRITE(6,1020)
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
  170 IF(MATCH("LAST",3)) GO TO 195
      IF(FIXED(X)) GO TO 175
      IF(END(X)) GO TO 170
  172 I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 170
  175 I = IVALUE
      NCS(I) = 0
```

```
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
  180 IF(MODE.NE.7) GO TO 170
      NCS(I) = NCS(I) + 1
      IF(NWD .GT. 5) NWD = 5
      DO 190 J =1, NWD
      IJ = J + 5*(NCS(I)-1)
  190 CSTUB(IJ,I) = ENTITY(J)
      CALL RDLINE
      GO TO 180
  195 I = 0
      GO TO 10
C
C     HERE WE HAVE THE ACTION STRINGS
  200 WRITE(6,1030)
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
  210 IF(MATCH("LAST",3)) GO TO 10
      IF(FIXED(X)) GO TO 215
      IF(END(X)) GO TO 210
  212 I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 210
  215 I = IVALUE
      NAS(I) = 0
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
  220 IF(MODE.NE.7) GO TO 210
      NAS(I) = NAS(I) + 1
      IF(NWD .GT. 5) NWD = 5
      DO 230 J =  1, NWD
      IJ = J + 5*(NAS(I) - 1)
  230 ASTUB(IJ,I) = ENTITY(J)
      CALL RDLINE
      GO TO 220
  235 I = 0
      GO TO 10
C
C     HERE WE HAVE THE TITLE
  240 IF(END(X)) CALL RDLINE
  250 IF(MODE.EQ.7) GO TO 255
      I = I + 1
      CALL ERROR(I)
      CALL RDLINE
      GO TO 240
  255 DO 260 J = 1, NWD
  260 TITLE(J) = ENTITY(J)
      CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
      I = 0
      GO TO 10
C
C
C
C     COMPACT THE CONDITION AND ACTION STUB ARRAYS FOR STORAGE
C
  270 I = 1
      DO 273 II = 1, NC
      JJ = NCS(II)
      DO 272 J = 1, JJ
      DO 271 IJ = 1, 5
      JI = IJ + 5*(J-1)
      IF(.NOT.(CSTUB(JI,II).EQ.BLANK)) CONDS(IJ,I) = CSTUB(JI,II)
  271 CONTINUE
  272 I = I + 1
```

```
      273 CONTINUE
          NCSR = I - 1
C
          I = 1
          DO 276 II = 1, NA
          JJ = NAS(II)
          DO 275 J = 1, JJ
          DO 274 IJ = 1, 5
          JI = IJ + 5*(J - 1)
          IF(.NOT.(ASTUB(JI,II).EQ.BLANK)) ACTS(IJ,I) = ASTUB(JI,II)
      274 CONTINUE
      275 I = I + 1
      276 CONTINUE
          NASR = I - 1
C
C         WRITE THE DATA ON DISK
          REWIND DF
          WRITE(DF,1040) (TITLE(J),J=1,10)
          WRITE(DF,1050) NR, NC, NA, NCSR, NASR
          DO 280 I = 1, NC
      280 WRITE(DF,1060) (RULES(I,J),J = 1, NR)
          WRITE(DF,1060) (ACTPTR(J),J = 1, NR)
          DO 290 I = 1, NCSR
      290 WRITE(DF,1040) (CONDS(J,I),J=1,5)
          DO 300 I = 1, NASR
      300 WRITE(DF,1040) (ACTS(J,I),J=1,5)
          WRITE(DF,1060) (NCS(I), I = 1, NC)
          WRITE(DF,1060) (NAS(I), I = 1, NA)
          END FILE DF
          LOCK DF
C
     1000 FORMAT(1H ,"BEGIN INPUT INSTRUCTIONS.",
         1    "  ENTER THE WORD END WHEN FINISHED.",/)
     1010 FORMAT(1H ,"ENTER THE RULE NUMBER, THE CONDITION ENTRIES,",
         1 " AND THE ACTION ENTRY,",/," ONE RULE TO A LINE."
         2 "   ENTER THE WORD LAST WHEN FINISHED.",/)
     1020 FORMAT(1H ,"ENTER THE CONDITION NUMBER AND THE STRINGS,",/,
         1 " 30 CHARACTERS TO A LINE.",
         2 "  ENTER THE WORD LAST WHEN FINISHED.",/)
     1030 FORMAT(1H ,"ENTER THE ACTION NUMBER AND THE STRING, ONE",
         1 " ACTION TO A LINE.",/," ENTER THE WORD LAST WHEN FINISHED.",/)
     1040 FORMAT(1H ,10A6)
     1050 FORMAT(1H ,5I3)
     1060 FORMAT(1H ,27I3)
          RETURN
          END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
          SUBROUTINE DFDATA
     $ INCLUDE 'TABCOM'
C
C
          LOGICAL TEST
C
C         FILL THE ARRAYS FROM FILE DF
          REWIND DF
          READ(DF, 1000) (TITLE(I),I=1,10)
          READ(DF, 1010) NR, NC, NA, NCSR, NASR
          TEST = .TRUE.
          IF(NCSR .EQ. 0) TEST = .FALSE.
          IF(NCSR .EQ. 0) NCSR = NC
```

```
      IF(NASR .EQ. 0) NASR = NA
      DO 10 I = 1, NC
   10 READ(DF, 1020) (RULES(I,J),J=1,NR)
      READ(DF, 1020) (ACTPTR(I),I=1,NR)
      DO 20 I = 1, NCSR
   20 READ(DF,1000) (CONDS(J,I),J=1,5)
      DO 30 I = 1, NASR
   30 READ(DF,1000) (ACTS(J,I),J=1,5)
      IF(.NOT. TEST) GO TO 40
      READ(DF,1020) (NCS(I), I = 1, NC)
      READ(DF,1020) (NAS(I), I = 1, NA)
   40 REWIND DF
 1000 FORMAT(1H , 10A6)
 1010 FORMAT(1H , 5I5)
 1020 FORMAT(1H , 27I3)
      RETURN
      END
C
C   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE TABTRE
 $ INCLUDE 'TABCOM'
C
C     SET UP COUNTERS AND FIRST SUBTABLE
C
      K = 1
      KK = 2
      N = NC + NR
      NCK(K) = NC
      KEY = 1
      IF(QUICK) KEY = -1
      DO 10 I = 1, N
   10 SUB(I,K) = 1
C
C     SELECT THE CONDITION TO BE TESTED FROM SUBTABLE K
C
C     COUNT THE IMMATERIAL ENTRIES IN EACH ROW
   20 CALL ZERO(IC, NC)
      CALL MATER
      AA = NCK(K)
      CALL ROW(IC, MR, NMR, 1, AA, CK)
      COND(K) = MR(1)
      IF(NMR .EQ. 1) GO TO 30
C
C     COUNT THE EXPLICIT ENTRIES IN EACH ROW
      CALL ZERO(EC, NC)
      CALL EXPLIT
      CALL ROW(EC, ER, NER, -1, NMR, MR)
      COND(K) = ER(1)
      IF(NER .EQ. 1) GO TO 30
C
C     COUNT THE DIFFERENCE BETWEEN TRUE AND FALSE ENTRIES
      CALL ZERO(DC, NC)
      CALL DELTA
      CALL ROW(DC, DR, NDR, KEY, NER, ER)
      COND(K) = DR(1)
C
C     COUNT TRUE ENTRIES IN SELECTED CONDITION
C
   30 NT = 0
      DO 40 I = 1, NR
      IF(SUB(NC+I,K) .EQ. 0) GO TO 40
```

```
      IF(RULES(COND(K),I) .LT. 0) GO TO 40
      NT = NT + 1
      TRULES(NT) = I
   40 CONTINUE
      IF(NT-1) 70, 60, 50
C
C     HERE WE HAVE MULTIPLE RULES, CHECK FOR REDUNDANCY
   50 IF(NCK(K).GT.1) GO TO 80
      WRITE(6,1000) (TRULES(I),I=1,NT)
      WRITE(6,1010)
      SWITCH = .FALSE.
      CALL RDLINE
      RETURN
C
C     HERE WE HAVE ONE RULE, COUNT THE NUMBER OF OTHER EXPLICIT ENTRIES
C     IN THIS RULE IN THE CURRENT SUBTABLE.
   60 CALL COUNT(TRULES(1))
      IF(NEE.GT.0) GO TO 80
      TSON(K) = -TRULES(1)
      GO TO 90
C
C     HERE WE HAVE SATISFIED AN ELSE RULE
   70 TSON(K) = 0
      GO TO 90
C
C     HERE WE BUILD A NEW SUBTABLE
   80 PARENT(KK) = K
      TSON(K) = KK
      CALL SUBTAB(1)
      GO TO 20
C
C     COUNT THE FALSE ENTRIES IN SELECTED CONDITION
C
   90 NF = 0
      DO 100 I = 1, NR
      IF(SUB((NC+I),K) .EQ. 0) GO TO 100
      IF(RULES(COND(K),I) .GT. 0) GO TO 100
      NF = NF + 1
      FRULES(NF) = I
  100 CONTINUE
      IF(NF-1) 130, 120, 110
C
C     HERE WE HAVE MULTIPLE RULES, CHECK FOR REDUNDANCY
  110 IF(NCK(K).GT.1) GO TO 140
      WRITE(6,1000) (FRULES(I),I=1,NF)
      WRITE(6,1010)
      SWITCH = .FALSE.
      CALL RDLINE
      RETURN
C
C     HERE WE HAVE ONE RULE, COUNT THE NUMBER OF OTHER EXPLICIT ENTRIES
C     IN THIS RULE IN THE CURRENT SUBTABLE.
  120 CALL COUNT(FRULES(1))
      IF(NEE.GT.0) GO TO 140
      FSON(K) = -FRULES(1)
      GO TO 150
C
C     HERE WE HAVE THE ELSE RULE
  130 FSON(K) = 0
      GO TO 150
C
C     HERE WE BUILD A NEW SUBTABLE
```

```
  140 PARENT(KK) = -K
      FSON(K) = KK
      CALL SUBTAB(-1)
      GO TO 20
C
C     LOOK FOR THE PARENT SUBTABLE
C
  150 IF(PARENT(K)) 160, 180, 170
C
C     FALSE SON HAS BEEN CHECKED, GO UP ONE LEVEL
  160 K = IABS(PARENT(K))
      GO TO 150
C
C     FALSE SON  HAS NOT BEEN CHECKED
  170 K = PARENT(K)
      GO TO 90
C
C     PARENT IS THE ROOT, FINISHED WITH THE DECOMPOSITION
  180 SWITCH = .TRUE.
C
C     MAKE THE ORDERED LIST OF CONDITIONS SINGLE-VALUED
      CALL COMPAC(COND, NC, NCOND)
      RETURN
C
 1000 FORMAT(1H , "THE FOLLOWING RULES ARE REDUNDANT OR CONTRADICTORY:",
     1  (5I5))
 1010 FORMAT(1H , "YOUR MUST MODIFY THE DATA.",
     1    "   ENTER THE MODIFY OR STOP COMMAND.")
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ZERO(A, B)
      INTEGER A(1), B
      DO 10 I = 1, B
   10 A(I) = 0
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE MATER
 $ INCLUDE 'TABCOM'
C
C     COUNT THE IMMATERIAL ENTRIES IN THE APPLICABLE RULES
C     FOR EACH CONDITION IN THIS SUBTABLE.
      I1 = 0
      CALL ZERO(CK, NC)
      DO 20 I = 1, NC
      IF(SUB(I,K) .EQ. 0) GO TO 20
      I1 = I1 + 1
      CK(I1) = I
      DO 10 J = 1, NR
      IF(SUB((NC+J), K) .EQ. 0) GO TO 10
      IF( RULES(I,J) .EQ. 0 )  IC(I1) = IC(I1) + 1
   10 CONTINUE
   20 CONTINUE
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
```

```
      SUBROUTINE ROW(A, B, Y, Z, AA, AAA)
$ INCLUDE 'TABCOM'
C
      INTEGER A(1), B(1), Y, Z, TEST, AA, AAA(1)
C
C     INITIALIZE SORTING VARIABLES
      CALL ZERO(B,NC)
      Y = 1
      B(1) = AAA(1)
      TEST = A(1)
C
C     SORT THE LIST TO FIND THE ROWS WITH THE SMALLEST
C     (OR LARGEST) NUMBERS IN THE LIST A
      DO 30 J = 2, AA
      IF(Z*( A(J) - TEST )) 10, 20, 30
   10 Y = 1
      B(1) = AAA(J)
      TEST = A(J)
      GO TO 30
   20 Y = Y + 1
      B(Y) = AAA(J)
   30 CONTINUE
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE EXPLIT
$ INCLUDE 'TABCOM'
C
C     COUNT THE EXPLICIT ENTRIES IN THE APPLICABLE RULES
C     FOR EACH OF THE MATERIAL CONDITIONS.
      DO 20 I1 = 1, NMR
      I = MR(I1)
      DO 10 J = 1, NR
      IF(SUB(NC+J,K) .EQ. 0) GO TO 10
      IF(IABS(RULES(I,J)) .EQ. 1)  EC(I1) = EC(I1) + 1
   10 CONTINUE
   20 CONTINUE
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE DELTA
$ INCLUDE 'TABCOM'
C
C     COUNT THE DIFFERENCE BETWEEN THE YES AND NO ENTRIES
C     IN THE APPLICABLE RULES FOR EACH OF THE EXPLICIT CONDITIONS
      DO 20 I1 = 1, NER
      I = ER(I1)
      DO 10 J = 1, NR
      IF(SUB(NC+J,K) .EQ. 0) GO TO 10
      IF(IABS(RULES(I,J)) .LE. 1) DC(I1) = DC(I1) + RULES(I,J)
   10 CONTINUE
      DC(I1) = IABS(DC(I1))
   20 CONTINUE
C
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
C
      SUBROUTINE SUBTAB(LOGIC)
 $ INCLUDE 'TABCOM'
C
      DO 10 I = 1, NC
   10 SUB(I,KK) = SUB(I,K)
      SUB(COND(K),KK) = 0
      IF(LOGIC.LT.0) GO TO 30
      DO 20 I = 1, NT
   20 SUB(NC+TRULES(I),KK) = 1
      GO TO 50
   30 DO 40 I = 1, NF
   40 SUB(NC+FRULES(I),KK) = 1
   50 NCK(KK) = NCK(K) - 1
      K = KK
      KK = KK + 1
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE COUNT(J)
 $ INCLUDE 'TABCOM'
C
      NEE = 0
      DO 10 I = 1, NC
      IF(I .EQ. COND(K)) GO TO 10
      IF(SUB(I,K).EQ.0) GO TO 10
      IF(IABS(RULES(I,J)).EQ.1) NEE = NEE + 1
   10 CONTINUE
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE COMPAC(I, J, L)
      INTEGER I(1), L(1), HIT(60)
      DO 5 M = 1, 60
    5 HIT(M) = 0
      II = 0
      DO 10 M = 1, 60
      IF(I(M) .EQ. 0)  GO TO 20
      IF( HIT(I(M)) .NE. 0 )  GO TO 10
      II = II + 1
      HIT(I(M)) = 1
      L(II) = I(M)
   10 CONTINUE
   20 IF(II .NE. J)  WRITE(6,1000)
      RETURN
 1000 FORMAT(1H , "NETWORK IS INCOMPLETE---EXAMINE FOR ERRORS.")
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE TOPO
 $ INCLUDE 'TABCOM'
C
C     SET THE INITIAL CONDITIONS TO TRAVERSE THE BINARY TREE IN PREORDER
C     FINDING THE DISTANCE OF THE NODES FROM EACH END AS WE GO.
C
      J = 2
      K = 1
```

```
      L = 1
      M = 1
      ORDER(1) = COND(1)
      LEVEL(1) = 0
      PTRBK(1) = 0
      BKSTK(1) = 1
C
C     TEST THE TRUE SON TO SEE IF IT IS A RULE OR A CONDITION
   10 IF(TSON(K).LE.0) GO TO 20
C     HERE IT IS A CONDITION
      ORDER(J) = COND(TSON(K))
      LEVEL(J) = L
      PTRBK(J) = J - 1
      M = M + 1
      BKSTK(M) = J
      J = J + 1
      K = TSON(K)
      L = L + 1
      GO TO 10
C     HERE IT IS A RULE
   20 ORDER(J) = TSON(K)
      LEVEL(J) = L
      PTRBK(J) = J - 1
      J = J + 1
C
C     TEST THE FALSE SON TO SEE IF IT IS  CONDITION OR A RULE
   30 IF(FSON(K).LE.0) GO TO 40
C     HERE IT IS A CONDITION
      ORDER(J) = COND(FSON(K))
      LEVEL(J) = L
      PTRBK(J) = BKSTK(M)
      BKSTK(M) = J
      J = J + 1
      K = FSON(K)
      L = L + 1
      GO TO 10
C     HERE IT IS A RULE
   40 ORDER(J) = FSON(K)
      LEVEL(J) = L
      PTRBK(J) = BKSTK(M)
      M = M - 1
      JJ = J
      J = J + 1
      LL = 1
C
C     PROCEED BACK UP TO THE PARENT CONDITION
   50 L = L - 1
      PK = PARENT(K)
      K = IABS(PK)
C     SET THE LENGTH AT THE PARENT EQUAL TO THE MAXIMUM OF THE TWO PATHS
      IF(LENGTH(PTRBK(JJ)).LT.LL) LENGTH(PTRBK(JJ)) = LL
      LL = LENGTH(PTRBK(JJ)) + 1
      JJ = PTRBK(JJ)
C     FIND OUT WHETHER WE ARE APPROACHING FROM TRUE OR FALSE BRANCH
      IF(PK) 50, 60, 55
C     SET THE LENGTH AT THE PARENT
   55 LENGTH(PTRBK(JJ)) = LL
      GO TO 30
C
C     HERE WE ARE AT THE ROOT NODE. STORE THE
C     NUMBER OF NODES IN THE NETWORK.
   60 NN = J - 1
```

```
C
C     MAKE THE PREORDER LIST OF THE RULES
      K = 0
      DO 70 I = 1, NN
      IF(ORDER(I) .GE. 0) GO TO 70
      K = K + 1
      NRULE(K) = IABS(ORDER(I))
   70 CONTINUE
C
C     CHECK THE NUMBER OF THE RULES
      IF(K .NE. NR) CALL COMPAC(NRULE, NR, NRULE)
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OUT1
 $ INCLUDE 'TABCOM'
C
C     WRITE THE PAGE HEADING AND THE ORIGINAL TABLE
C
      IF(.NOT. TABTWO) WRITE(FLOUT,1000)
      ICL = 1
      IRUL = 1
      JRUL = NR
      IF((FLOUT .NE. 6) .OR. (NR .LE. 12)) GO TO 10
      JRUL = 12
   10 IF(.NOT. TABTWO) WRITE(FLOUT,1010) TITLE, (I,I=IRUL,JRUL)
      IF(TABTWO) WRITE(FLOUT,1010) TITLE, (NRULE(I),I=IRUL,JRUL)
      DO 30 II = 1, NC
      I = II
      IF(TABTWO) I = NCOND(II)
      IF(TABTWO) CALL SUBSCR(NCS, I, ICL)
      WRITE(FLOUT,1020) I, (CONDS(J,ICL),J=1,5), ASTER,
     1  (SIGN(RULES(I,J)+3),J=IRUL,JRUL)
      ICL = ICL + 1
      IF(NCS(I) .LT. 2) GO TO 30
      JJ = NCS(I)
      DO 20 K = 2, JJ
      WRITE(FLOUT,1040) (CONDS(J,ICL),J=1,5), ASTER
   20 ICL = ICL + 1
   30 CONTINUE
      J = 36 + 3*(JRUL + 1 - IRUL)
      WRITE(FLOUT,1030)  (DIVIDE(I),I=1,J)
      IAL = 1
      DO 60 I = 1, NA
      DO 40 JJ = IRUL, JRUL
      ACTION(JJ) = BLANK
      J = JJ
      IF(TABTWO) J = NRULE(JJ)
      IF(ACTPTR(J).EQ.I) ACTION(JJ) = XX
   40 CONTINUE
      WRITE(FLOUT,1020) I, (ACTS(K,IAL),K=1,5), ASTER,
     1  (ACTION(J),J=IRUL,JRUL)
      IAL = IAL + 1
      IF(NAS(I) .EQ. 1) GO TO 60
      JJ = NAS(I)
      DO 50 K = 2, JJ
      WRITE(FLOUT,1040) (ACTS(J,IAL),J=1,5), ASTER
   50 IAL = IAL + 1
   60 CONTINUE
      IF(NR .LE. JRUL) RETURN
```

```
      IRUL = IRUL + 12
      JRUL = JRUL + 12
      IF(JRUL .GT. NR) JRUL = NR
      GO TO 10
      RETURN
C
 1000 FORMAT(1H1, "ORIGINAL DECISION TABLE")
 1010 FORMAT(1H0,//, 10X, 10A6,//, 36X, 12I3)
 1020 FORMAT((1H , I2, X, 5A6), X, 12(A1, 2X), A1)
 1030 FORMAT(1H , 72A1)
 1040 FORMAT(1H , 3X, 5A6, X, A1)
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE SUBSCR(I, J, K)
      DIMENSION I(1)
      K = 1
      IF(J .EQ. 1) RETURN
      M = J - 1
      DO 10 L = 1, M
   10 K = K + I(L)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OUT2
 $ INCLUDE 'TABCOM'
C
C     WRITE THE DECISION NETWORK
C
      CALL ARRAY
      IF(FLOUT .EQ. 6) GO TO 30
      DO 20 J = 1, JMAX
   20 WRITE(FLOUT,1010) (NET(I,J),I=1,132)
      RETURN
   30 DO 40 J = 1,JMAX
   40 WRITE(FLOUT,1030) (NET(I,J),I=1,72)
      IF(IMAX.LE.72) RETURN
      DO 50 J = 1, JMAX
   50 WRITE(FLOUT,1030) (NET(I,J),I=73,132)
      RETURN
 1010 FORMAT(1H , 132A1)
 1030 FORMAT(1H , 72A1)
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ARRAY
 $ INCLUDE 'TABCOM'
C
C      CONSTRUCT THE OUTPUT FOR THE DECISION NETWORK
      INTEGER SKC(15)
      DO 5 I = 1, 120
      DO 5 J = 1, 132
    5 NET(J,I) = BLANK
      DO 8 I = 1,15
      CVB(I) = 0
    8 LVB(I) = 0
      JMAX = 0
      IMAX = 0
```

```
      J = 1
      K = 0
      KC = 1
      M = 0
      MK = 0
   10 K = K + 1
      IF(K.GT.MN) RETURN
      I = 10*LEVEL(K)
C
C     CHECK TO SEE IF THE NODE IS A CONDITION OR A RULE
      IF(ORDER(K))  60, 50, 1
C
C     THE NODE IS A CONDITION, WRITE IT IN THE ARRAY
    1 NET((I+1),J) = C
      CALL NUMB(I,J)
C
C     FIND THE SIGN OF THE NEXT BRANCH AND THE NUMBER OF THE
C     NEXT SUBTABLE
C
      IF(TSON(KC)) 14, 14, 11
   11 IF(FSON(KC)) 13, 13, 12
   12 IF(ORDER(K+1) .EQ. COND(TSON(KC))) GO TO 17
      GO TO 19
   13 IF(ORDER(K+1) .EQ. COND(TSON(KC))) GO TO 18
      GO TO 20
   14 IF(FSON(KC)) 16, 16, 15
   15 IF(ORDER(K+1) .EQ. TSON(KC)) GO TO 22
      GO TO 23
   16 IF(ORDER(K+1) .EQ. TSON(KC)) GO TO 25
      GO TO 26
C     TRUE BRANCH, NEXT TRUE SUBTABLE
   17 MK = MK + 1
      SKC(MK) = FSON(KC)
   18 KC = TSON(KC)
      BRANCH = .TRUE.
      GO TO 28
C     FALSE BRANCH, NEXT SUBTABLE ALONG IT
   19 MK = MK + 1
      SKC(MK) = TSON(KC)
      KC = FSON(KC)
      GO TO 21
   20 KC = TSON(KC)
   21 BRANCH = .FALSE.
      GO TO 28
C     NEXT SUBTABLE IS ON FALSE BRANCH
   22 BRANCH = .TRUE.
      GO TO 24
   23 BRANCH = .FALSE.
   24 KC = FSON(KC)
      GO TO 28
C     NEXT SUBTABLE IS BACK IN STACK
   25 BRANCH = .TRUE.
      GO TO 27
   26 BRANCH = .FALSE.
   27 IF(MK .EQ. 0) GO TO 28
      KC = SKC(MK)
      MK = MK -1
C     PROCEED WITH THE NETWORK
   28 IF(BRANCH) GO TO 29
      ROW = MINUS
      COLUMN = PLUS
      CVB(LEVEL(K) + 1) = -1
```

```
      GO TO 30
   29 ROW = PLUS
      COLUMN = MINUS
      CVB(LEVEL(K) + 1) = 1
C
C     EXTEND THE HORIZONTAL BRANCH
   30 I5 = I + 5
      I9 = I + 9
      DO 35 II = I5, I9, 2
   35 NET(II,J) = ROW
C
C     EXTEND A VERTICAL BRANCH DOWN FROM THE CONDITION
      J1 = J + 1
      J3 = J + 3
      DO 40 JJ = J1, J3
   40 NET((I+2),JJ) = COLUMN
      M = M + 1
      LVB(M) = I
      GO TO 10
C
C     THE NODE IS AN ELSE RULE, WRITE IT IN THE ARRAY
   50 NET((I-1),J) = E
      NET(I,J) = EL
      NET((I+1),J) = S
      NET((I+2),J) = E
      IF((I+2).GT.IMAX) IMAX=I+2
      GO TO 70
C
C     THE NODE IS A RULE, WRITE IT IN THE ARRAY
   60 NET((I+1),J) = R
      CALL NUMB(I,J)
      IF((I+2) .GT. IMAX) IMAX = I + 2
C
C     EXTEND THE LAST VERTICAL BRANCH ACROSS TO THE NEXT NODE
   70 IF(M .LT. 1) RETURN
      J = J + 3
      IF(JMAX .LT. J) JMAX = J
      I = LVB(M)
      I5 = I + 5
      I9 = I + 9
      POINT = MINUS
      IF(CVB((I/10) + 1) .EQ. -1) POINT = PLUS
      CVB((I/10) + 1) = 0
      M = M - 1
      DO 80 II = I5, I9, 2
   80 NET(II,J) = POINT
C
C     EXTEND THE INCOMPLETE VERTICAL BRANCHES
      IF(M.EQ.0) GO TO 10
      J1 = J + 1
      J3 = J + 3
      IM = LVB(M) + 2
      DO 130 II = 2, IM, 10
      IF(CVB(((II-2)/10) + 1)) 90, 130, 100
   90 POINT = PLUS
      GO TO 110
  100 POINT = MINUS
  110 DO 120 JJ = J1, J3
      NET(II,JJ) = POINT
  120 CONTINUE
  130 CONTINUE
      GO TO 10
```

```
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE NUMB(I,J)
$ INCLUDE 'TABCOM'
C
      NUMBER = IABS(ORDER(K))
      IF(NUMBER.GT.9) GO TO 10
      NET((I+2),J) = SYMBOL(NUMBER + 1)
      IF((I+2).GT.IMAX) IMAX=I+2
      RETURN
C
   10 N10 = NUMBER / 10
      N1 = NUMBER - 10*N10
      NET((I+2),J) = SYMBOL(N10+1)
      NET((I+3),J) = SYMBOL(N1+1)
      IF((I+3).GT.IMAX) IMAX=I+3
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE SORT
$ INCLUDE 'TABCOM'
C
C     SET THE INITIAL CONDITIONS.  THE VECTORS PTRBK, LVB, AND
C     BKSTK WHICH WERE USED IN THE TOPO ROUTINE WILL BE
C     USED HERE FOR TEMPORARY STORAGE.
      CALL ZERO(PTRBK,NN)
      CALL ZERO(BKSTK,NN)
      CALL ZERO(LVB,10)
      J = 1
      K = 1
      M = 0
C
C     PUT THE FIRST CONDITION NODE IN THE SORTED LIST
      PTRBK(K) = ORDER(J)
      BKSTK(K) = LEVEL(J)
      K = K + 1
C     FIND THE SONS OF THE CURRENT NODE
   10 J1 = J + 1
      J2 = J + 2
      L = LEVEL(J1)
C
C     J1 IS THE TRUE SON.  FIND THE FALSE SON OF THIS CONDITION
      DO 20 I = J2, NN
      IF(LEVEL(I) .EQ. L) GO TO 30
   20 CONTINUE
C
C     FIND THE SHORTEST BRANCH
   30 IF(LENGTH(J1) .GT. LENGTH(I)) GO TO 40
      N1 = J1
      N2 = I
      GO TO 45
   40 N1 = I
      N2 = J1
C
C     PUT THE NODE ON THE SHORTEST BRANCH IN THE ORDERED LIST
   45 PTRBK(K) = ORDER(N1)
      BKSTK(K) = LEVEL(N1)
      K = K + 1
```

```
C
C     CHECK FOR NODES BETWEEN I AND J2
      IF(I .EQ. J2) GO TO 50
      IF(ORDER(N1)) 50, 50, 48
   48 M = M + 1
      LVB(M) = N2
      J = N1
      IF(I .GE. NN) GO TO 55
      GO TO 10
   50 PTRBK(K) = ORDER(N2)
      BKSTK(K) = LEVEL(N2)
      K = K + 1
C
C     SEE IF N2 IS A CONDITION
      IF(ORDER(N2) .LE. 0) GO TO 55
C     N2 IS A CONDITION
      J = N2
      GO TO 10
C
C     CHECK TO SEE IF WE ARE FINISHED
   55 IF(K .GT. NN) GO TO 70
C
C     CHECK TO SEE IF WE HAVE NODES WAITING IN THE STACK
      IF(M.EQ.0) GO TO 60
      J = LVB(M)
      PTRBK(K) = ORDER(J)
      BKSTK(K) = LEVEL(J)
      K = K + 1
      M = M - 1
      GO TO 10
   60 J = N2
      GO TO 10
C
C     HERE WE ARE DONE. PUT THE ORDERED NETWORK INTO ORDER.
C     REORDER THE LEVEL VECTOR, AND SET NCOND AND NRULES.
   70 J = 0
      K = 0
      DO 100 I = 1, NN
      ORDER(I) = PTRBK(I)
      LEVEL(I) = BKSTK(I)
      IF(ORDER(I)) 80, 100, 90
C     HERE IT IS A RULE
   80 J = J + 1
      NRULE(J) = IABS(ORDER(I))
      GO TO 100
C     HERE IT IS A CONDITION
   90 K = K + 1
      NCOND(K) = ORDER(I)
  100 CONTINUE
C
C     CHECK THE NUMBER OF RULES
      IF(J.NE.NR) CALL COMPAC(NRULE, NR, NRULE)
C
C     CHECK THE NUMBER OF CONDITIONS
      IF(K.NE.NC) CALL COMPAC(NCOND, NC, NCOND)
C
C     THE LISTS ORDER, NCOND, AND NRULE ARE NOW SORTED
C     BY BRANCH LENGTH AND READY TO BE USED FOR OUTPUT.
C
      RETURN
      END
C
```

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ERROR(I)
      IF(I .GT. 10) GO TO 10
      WRITE(6,1000)
      RETURN
   10 WRITE(6,1010)
      WRITE(1,1010)
      STOP
C
 1000 FORMAT(1H , "INCORRECT INPUT---REENTER ON A NEW LINE")
 1010 FORMAT(1H , "INCORRECT INPUT---PROGRAM EXECUTION STOPPED.")
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      BLOCK DATA
 $ INCLUDE 'TABCOM'
C
      DATA BLANK, ASTER, XX, C, E, R, PLUS, MINUS, EL, S/6H      ,
     1 6H*     , 6HX     , 6HC     , 6HE     , 6HR     ,
     2 6H+     , 6H-     , 6HL     , 6HS     /,
     3 (SYMBOL(I),I=1,10), (SIGN(I),I=1,5)/6H0     ,
     4 6H1     , 6H2     , 6H3     , 6H4     , 6H5     ,
     5 6H6     , 6H7     , 6H8     , 6H9     , 6H-     ,
     6 6HF     , 6H.     , 6HT     , 6H+     /
      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.

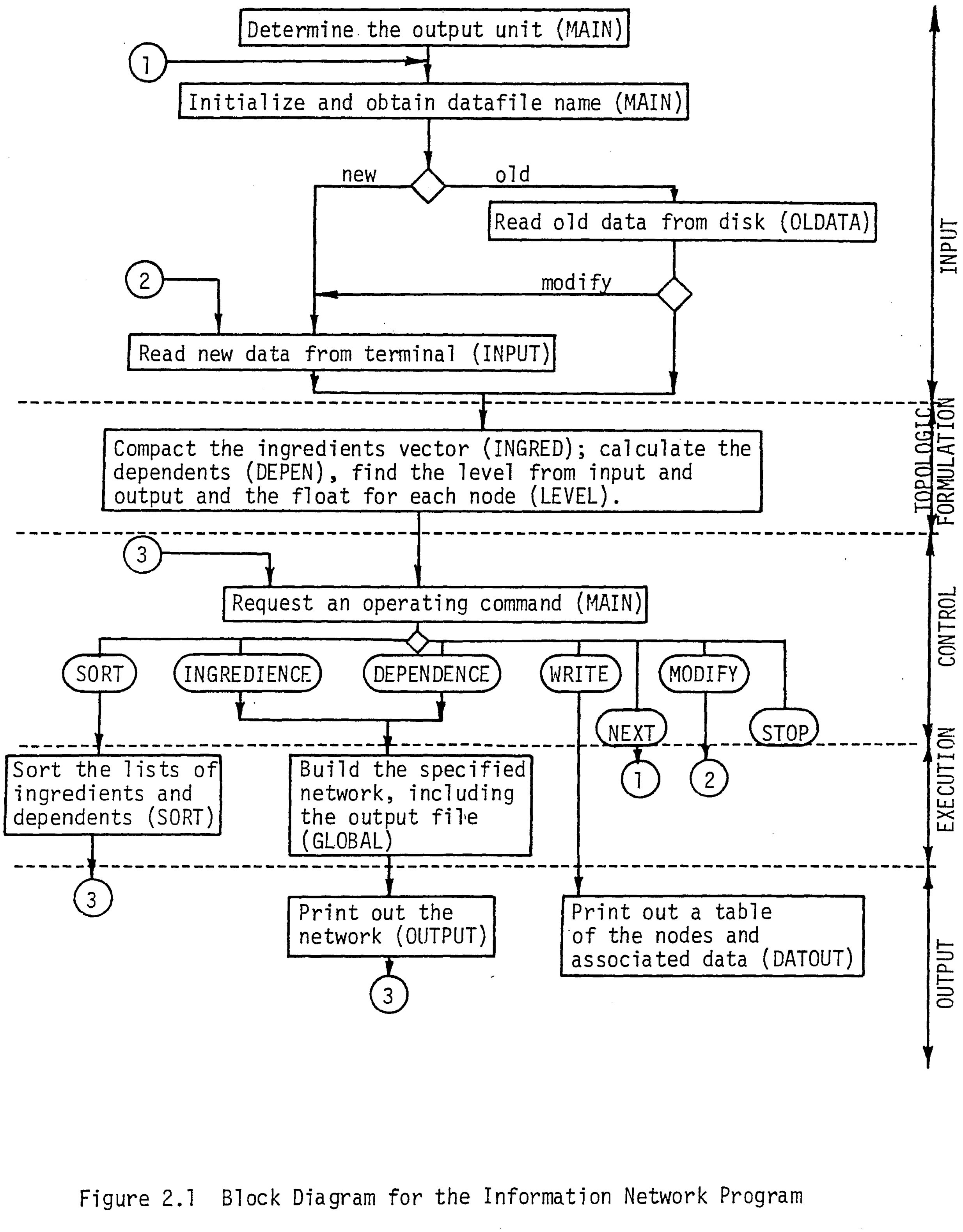
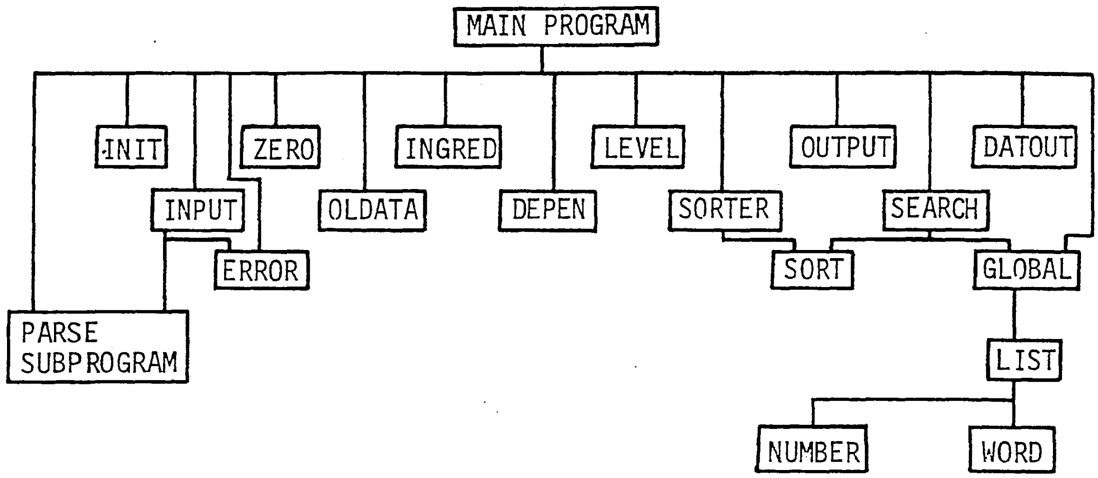


Figure 2.1 Block Diagram for the Information Network Program



NOTES:

1. PARSE contains several functions and subroutines, including WDINIT, RDLINE, 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)

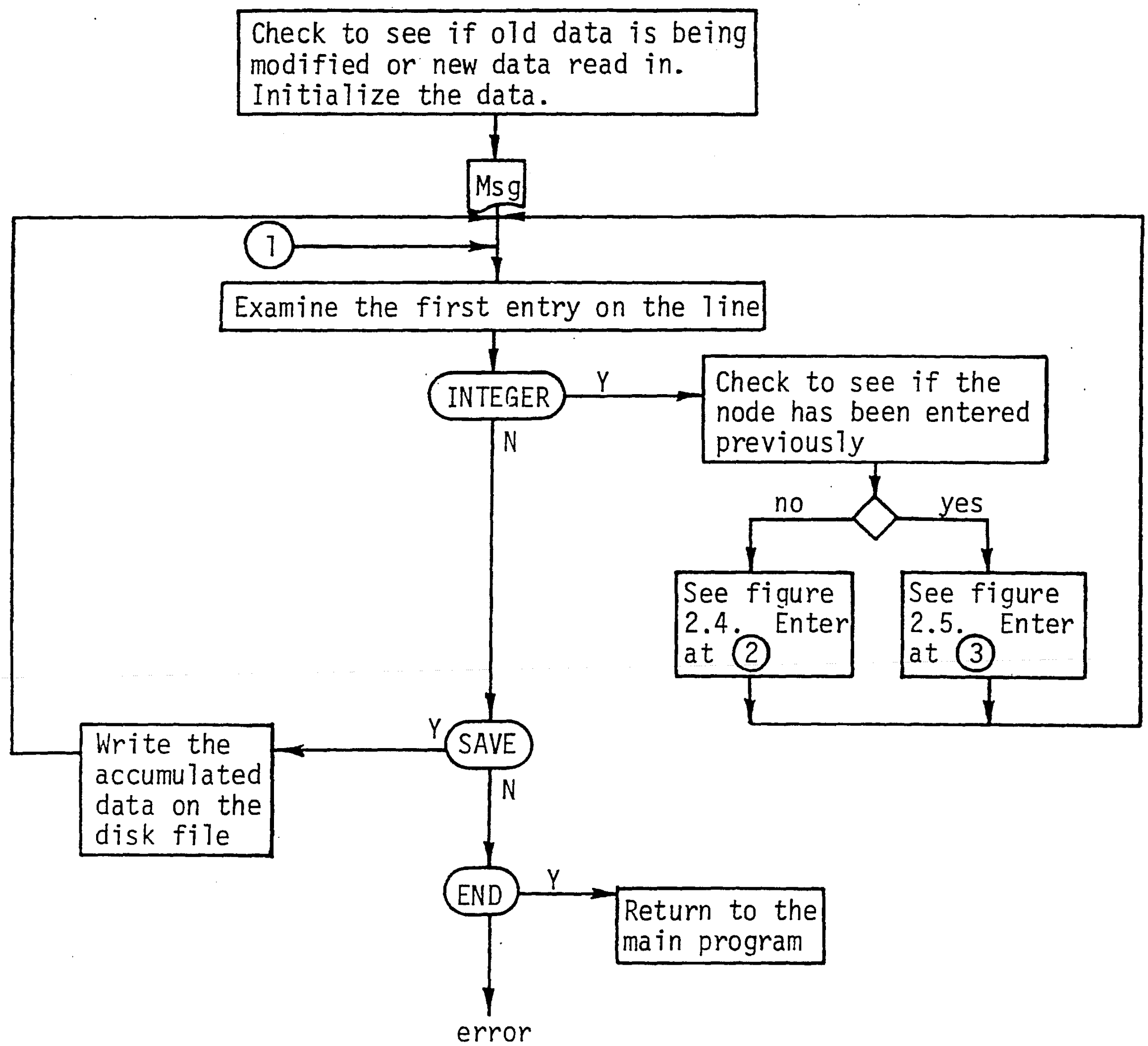


Figure 2.3 Subroutine INPUT (NETWORK)

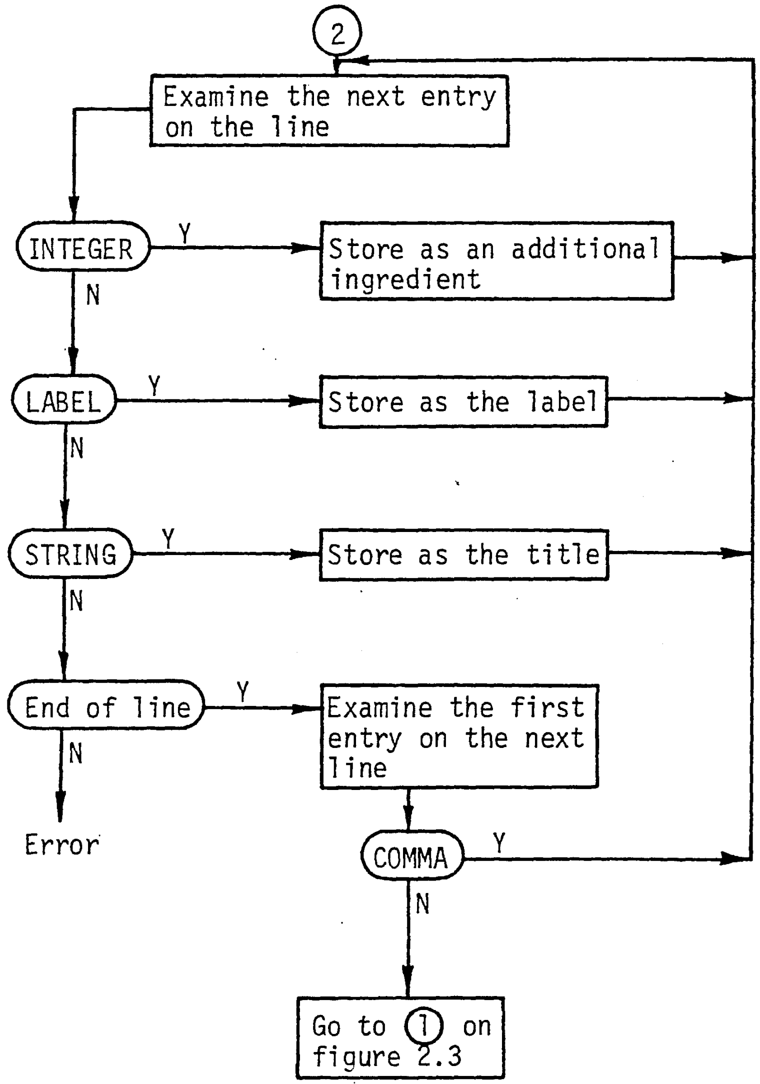


Figure 2.4 Input of Data for a New Node (NETWORK)

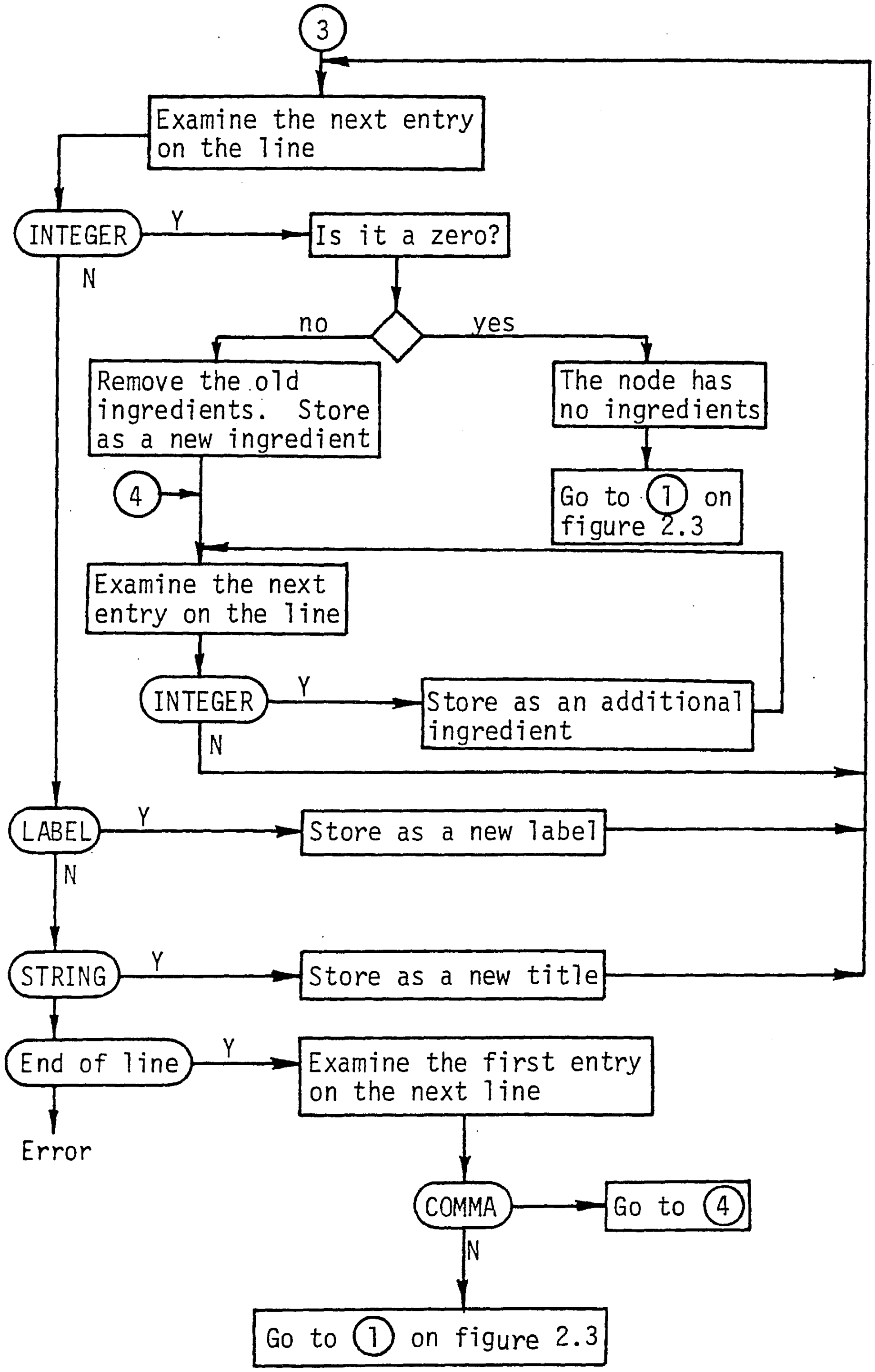


Figure 2.5   Input of New Data for a Node (NETWORK)

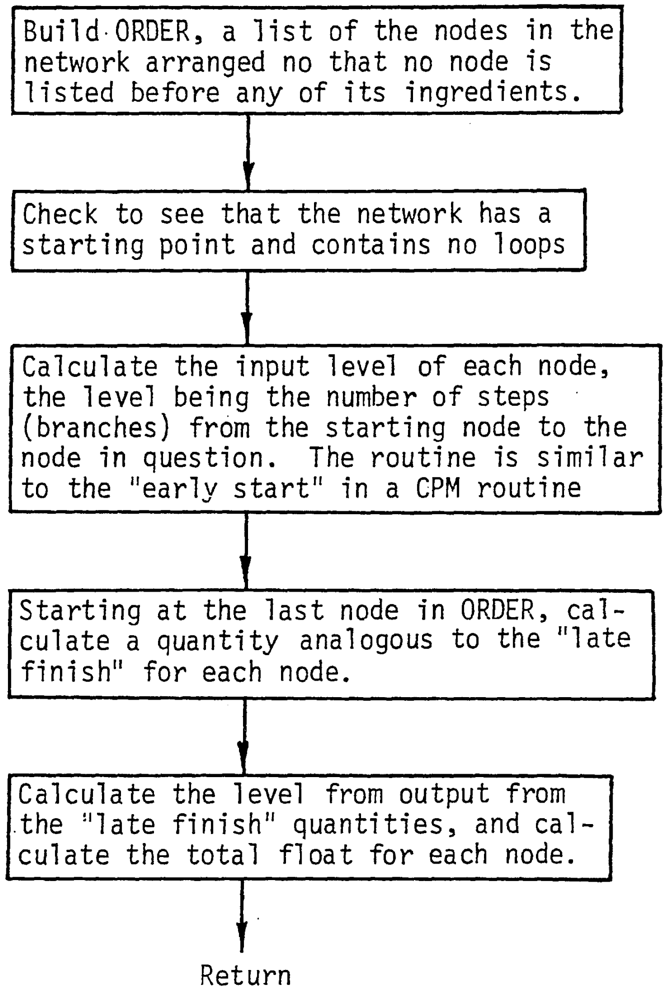


Figure 2.6 Subroutine LEVEL (NETWORK)

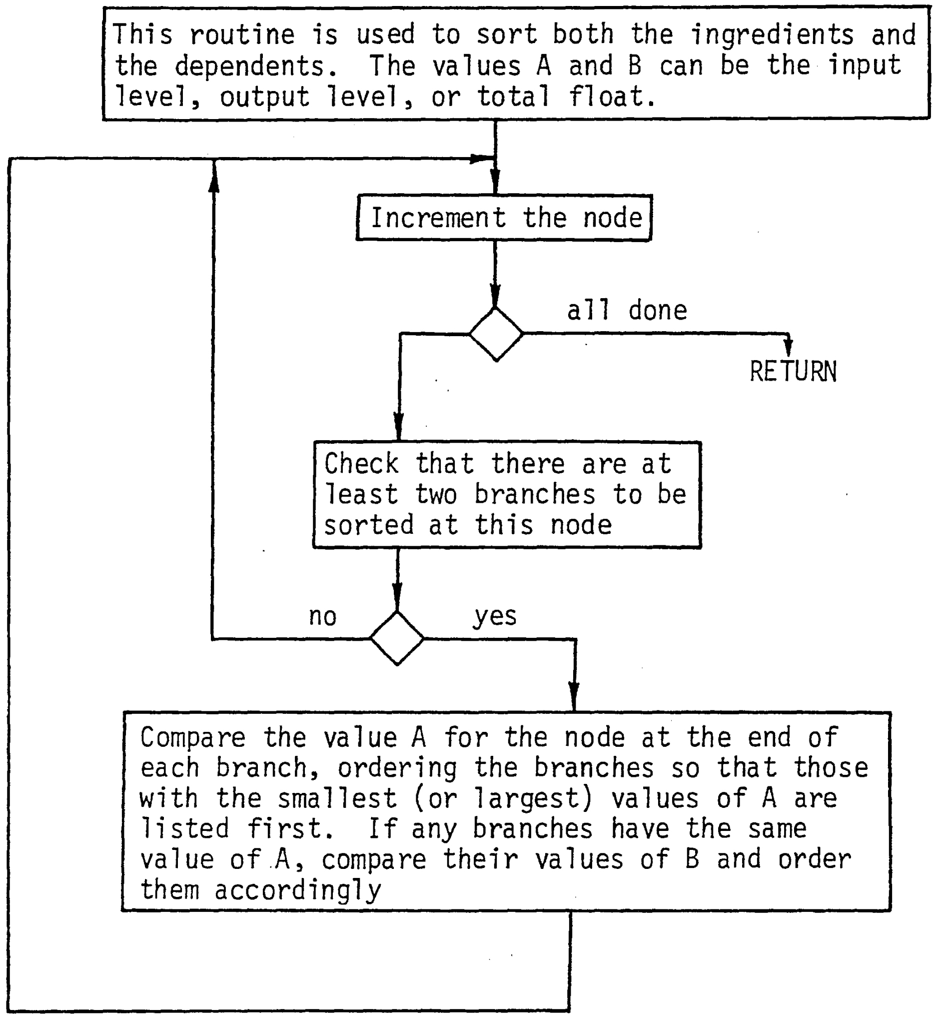


Figure 2.7 Subroutine SORT (NETWORK)

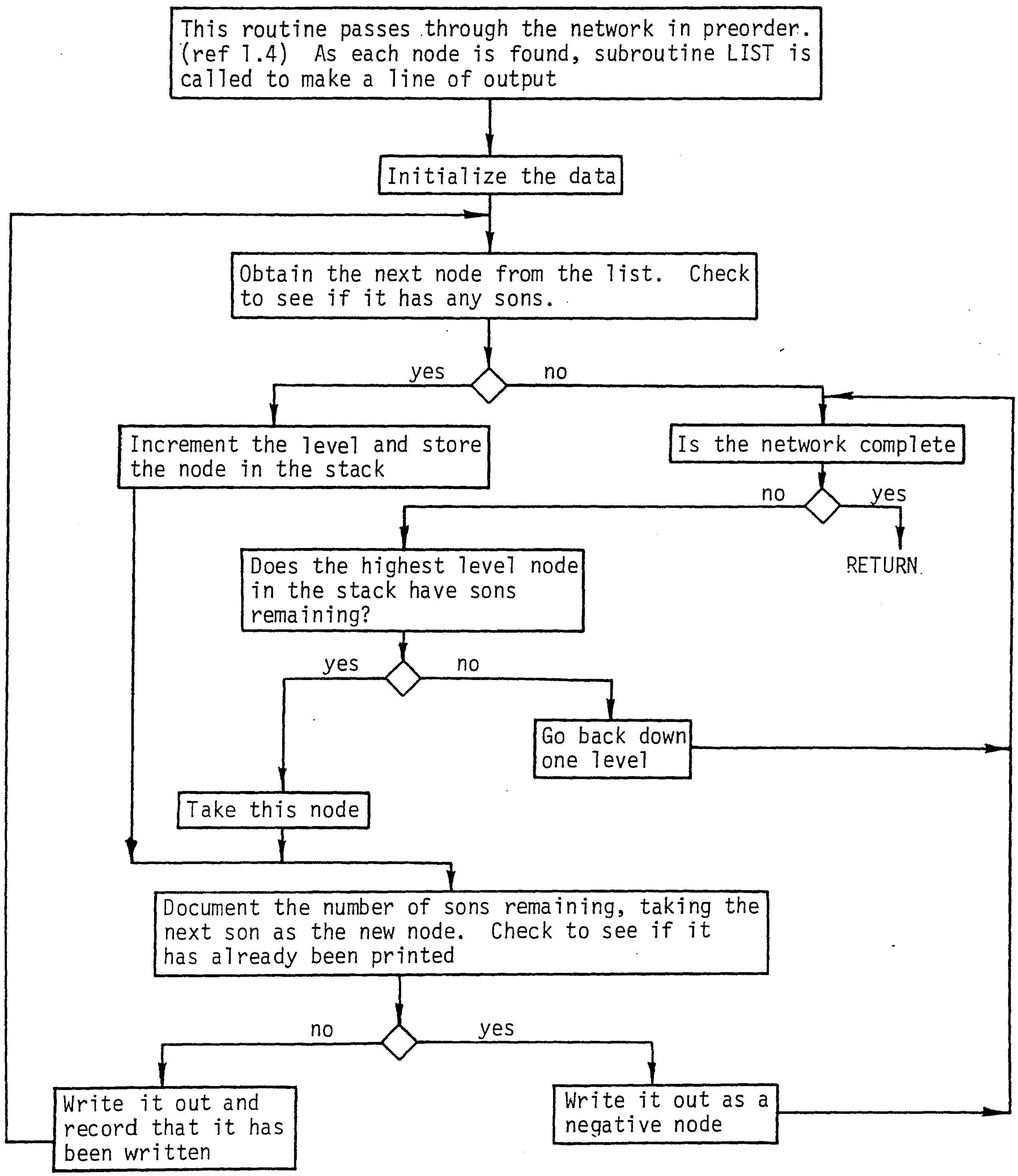


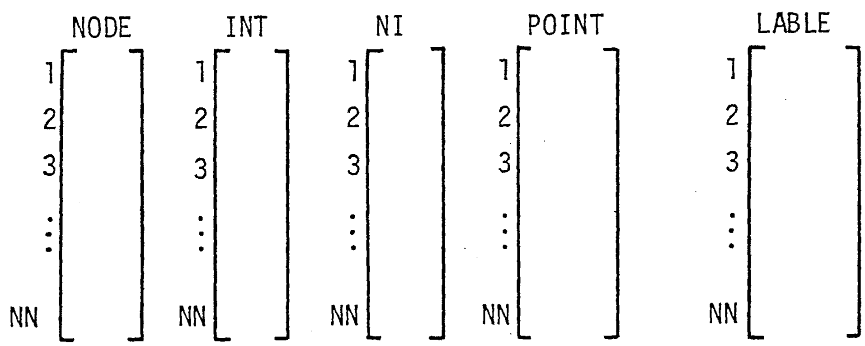
Figure 2.8 Subroutine GLOBAL (NETWORK)

2.3 Data Structure

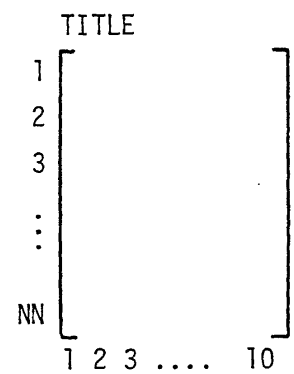
The figures on the following pages show the principal items of data used in the program. A complete list of all the data items used in the program is in the glossary, section 2.4.

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 incredients 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.

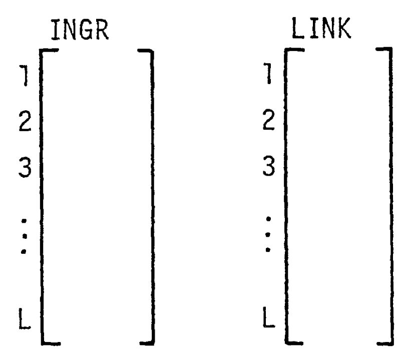
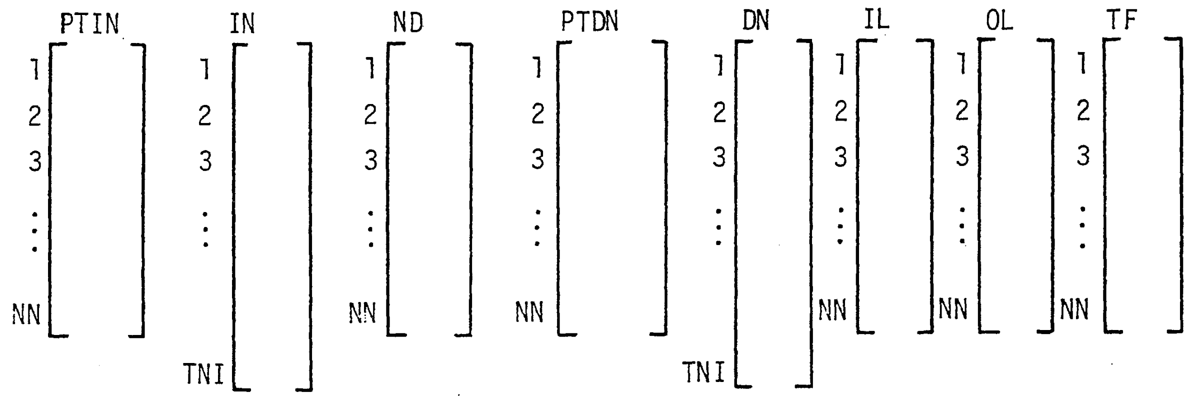


Figure 2.9 Permanent Data (NETWORK)

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

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)

2.4 GLOSSARY

SYMBOL	DESCRIPTION
A()	DUMMY NAME FOR IL, OL, OR TF
ALPHA1, ALPHA2, ALPHA3, ALPHA4	VARIABLES USED TO CARRY ALPHABETIC INFORMATION FOR USE IN DESCRIBING THE MANNER OF SORTING
ASTER	CARRIES THE ASTERISK SYMBOL
B()	DUMMY NAME FOR IL, OL, OR TF
BLANK	CARRIES A BLANK SYMBOL. IN SOME SUBROUTINES IT IS A SIX CHARACTER WORD, WHILE IN OTHERS IT IS ONLY ONE CHARACTER
COLON	THE COLON SYMBOL
DASH	A SIX CHARACTER WORD; ALL CARRYING THE DASH SYMBOL
DATOUT	A SUBROUTINE
DEPEN'	A SUBROUTINE
DF	THE NUMBER OF THE DATAFILE BEING USED
DN()	VECTOR WHICH LISTS THE DEPENDENTS OF ALL THE NODES
EIGHT	THE SYMBOL 8 IN ALPHAMERIC FORM
END	A LOGICAL FUNCTION IN THE PARSE ROUTINES THAT IS TRUE WHEN THE SCANNER IS AT THE END OF A LINE
ENTITY()	THE VECTOR WHICH CONTAINS THE MATERIAL SCANNED BY PARSE
ERROR	A SUBROUTINE
FLO	THE WORD FLOAT (THE SIXTH CHARACTER IS A BLANK)
FIVE	THE SYMBOL 5 IN ALPHAMERIC FORM
FIXED	A LOGICAL FUNCTION IN THE PARSE ROUTINES THAT IS TRUE IF THE ITEM SCANNED IS AN INTEGER
FOUR	THE SYMBOL 4 IN ALPHAMERIC FORM
GLOB()	A VECTOR CONTAINING THE NODES LISTED IN PREORDER

```
GLOBAL    :  A SUBROUTINE
          :
HIT()     :  A VECTOR THAT IS USED TO RECORD THE OCCURANCE OF A NODE
          :  IN THE PREORDER LIST
          :
I         :  A COUNTING INDEX: AS USED IN "INPUT", IT IS THE NUMBER OF
          :  NODES WHICH HAVE BEEN ENTERED INTO THE SYSTEM
          :
II, IJ    :  COUNTING INDICES
          :
IL()      :  VECTOR WHICH CONTAINS THE INPUT LEVEL:  THE LONGEST
          :  PATH FROM THE NODE TO THE INPUT NODES
          :
IN()      :  VECTOR THAT LISTS THE INGREDIENTS OF ALL THE NODES
          :
INGIS     :  A SUBROUTINE
          :
INGR()    :  THE LINKED LIST OF INGREDIENTS AS ENTERED
          :
INGRED    :  A SUBROUTINE
          :
INIT      :  A SUBROUTINE
          :
INPUT     :  A SUBROUTINE
          :
INT()     :  VECTOR THAT CONTAINS THE INTERNAL NODE NUMBER.  THE
          :  ROW CORRESPONDS TO THE EXTERNAL NUMBER
          :
IVALUE    :  VARIABLE WHICH CARRIES AN INTEGER NUMBER BACK FROM PARSE
          :
IP1, IP2, :  VARIOUS COUTNING INDICES AND TEMPORARY INTEGERS
I1, I2,   :
I4, I9,   :
          :
J         :  AS USED IN THE INPUT ROUTINE IT IS THE NODE THAT HAS
          :  INGREDIENTS BEING ENTERED. ELSWHERE IT IS A
          :  COUNTING INDEX
          :
JL        :  THE ROW IN ORDER WITH THE LAST ENTRY
          :
JK        :  THE LAST ROW IN ORDER THAT HAS HAD IT DEPENDENTS PROCESSED
          :
JJ        :  NODE WHICH IS HAVING ITS DEPENDENTS PROCESSED FOR ORDER
          :
K         :  A COUNTING INDEX
          :
KEY       :  AN INTEGER USED TO TRACE THE PATH INTO SUBROUTINE GLOBAL
          :
KL, K1,   :  COUNTING INDICES
K2        :
          :
KOUNT     :  RECORD THE NUMBER OF LINES OF OUTPUT IN THE GRAPHICAL
          :  TREE PRODUCED IN GLOBAL
          :
KOUT      :  FILE NUMBER FOR PRINTED OUTPUT.  IT ALWAYS IS FILE NO. 1
          :  IN THE PRESENT PROGRAM
          :
L         :  USED TO RECORD THE LENGTH OF THE LINKED LIST OF INGREDIENTS.
          :  IT ALSO IS USED TO RECORD THE CORRENT LEVEL INTO THE
          :  NETWORK IN THE ROUTINES GLOBAL AND OUTPUT
          :
LABEL()   :  VECTOR CONTAINING THE ALPHA LABELS (UP TO SIX
```

```
                 :  CHARACTERS) FOR EACH NODE
                 :
LARGE            :  THE WORD LARGE (THE SIXTH CHARACTER IS BLANK)
                 :
LENGTH           :  USED IN SUBROUTINE LEVEL TO RECORD THE INPUT LEVEL OF
                 :  NODE NF.  WHEN COMPLETED, IT IS THE LENGTH OF THE
                 :  LONGEST PATH THROUGH THE NETWORK
                 :
LEV()            :  DUMMY NAME FOR IL OR OL
                 :
LEVEL            :  A SUBROUTINE
                 :
LI, L1,          :  VARIOUS COUNTING INDICES AND TEMPORARY NODE NUMBERS
L2, LL           :
                 :
LINE()           :  ALPHAMERIC LIST USED TO STORE ONE LINE OF OUTPUT
                 :
LINEI            :  A PARTICULAR SPACE IN LINE()
                 :
LINK()           :  VECTOR WHICH LINKS THE INGREDIENTS OF EACH NODE
                 :  TOGETHER IN THE LIST INGR
                 :
LIST             :  A SUBROUTINE
                 :
LN               :  THE LEVEL OF THE NODE BEING PRINTED
                 :
LV               :  THE WORD LEVEL (THE SIXTH CHARACTER IS BLANK)
                 :
M                :  COUNTS THE NUMBER OF INGREDIENTS AS THEY ARE ENTERED
                 :
MARK             :  INTEGER USED TO TRACE THE PATH INTO THE OUTPUT ROUTINE
                 :
MATCH            :  A LOGICAL FUNCTION IN PARSE THAT IS TRUE IF THE INPUT
                 :  AGREES WITH THE ARGUMENT
                 :
MINUS            :  THE MINUS SYMBOL
                 :
MODE             :  AN INTEGER RETURNED BY PARSE THAT INDICATES THE KIND
                 :  OF INPUT BEING SCANNED
                 :
MULT1,           :  INTEGERS USED TO DIFFERENTIATE SORTING FOR LARGE
MULT2            :  DIFFERENCES AND SMALL DIFFERENCES
                 :
N                :  A PARTICULAR NODE NUMBER
                 :
NCHAR            :  THE NUMBER OF CHARACTERS IN THE WORD SCANNED BY PARSE
                 :
ND()             :  VECTOR CONTAINING THE NUMBER OF DEPENDENTS OF
                 :  EACH NODE
                 :
NEXT             :  A LOGICAL FUNCTION IN PARSE
                 :
NF               :  THE FARTHEST NODE ALONG THE NETWORK WHICH HAS
                 :  BEEN PROCESSED
                 :
NI()             :  VECTOR CONTAINING THE NUMBER OF LOCAL INGREDIENTS
                 :  OF EACH NODE
                 :
NINE             :  THE SYMBOL 9 IN ALPHAMERIC FORM
                 :
NN               :  TOTAL NUMBER OF NODES
                 :
```

NNR() : DUPLICATE OF A DUMMY FOR THE LISTS NI() OR ND()

NODE() : VECTOR CONTAINING THE EXTERNAL NUMBERS FOR EACH NODE. IT IS ALSO USED TO DENOTE A SPECIFIC ONE OF THESE IN SUBROUTINE LIST

NOUGHT : THE SYMBOL 0 IN ALPHAMERIC FORM

NP() : DUMMY NAME FOR NI() OR ND()

NP : COUNTER OF THE PAGE NUMBER FOR OUTPUT

NR() : DUMMY NAME FOR NI() OR ND()

NUM : USED FOR STORAGE OF THE NUMBER OF THE ARTIFICIAL ROOT

NUMB : STORES ONE DIGIT OF A NODE NUMBER

NUMBER : A SUBROUTINE

NWD : A VALUE RETURNED BY PARSE

NWDS : THE NUMBER OF SIX CHARACTER WORDS OCCUPPIED BY A TITLE

N1 : THE UNITS DIGIT OF A NODE NUMBER

N10 : THE TENS DIGIT OF A NODE NUMBER

N100 : THE HUNDREDS DIGIT OF A NODE NUMBER

OL() : OUTPUT LEVEL; THE LONGEST PATH FROM THE NODE TO OUTPUT

OLDATA : A SUBROUTINE

ONE : THE SYMBOL 1 IN ALPHAMERIC FORM

ORDER() : VECTOR WHICH LISTS THE NODES STARTING WITH THE INPUT NODES AND ENDING WITH THE OUTPUT NODES

OUTPUT : A SUBROUTINE

PARSE : A FAMILY OF SUBROUTINES THAT IS AN INTERPRETIVE INPUT SCANNER

POINT() : VECTOR THAT POINTS TO THE LIST INGR()

PRT : VARIABLE WHICH TURNS THE ECHO PRINT OF THE INPUT SCANNER ON OR OFF

PTDN() : VECTOR THAT POINTS TO THE LIST OF DEPENDENTS, DN()

PTIN() : VECTOR THAT POINTS TO THE LIST OF INGREDIENTS, IN()

PTRN() : DUMMY NAME FOR PTDN() OR PTIN()

RDLINE : A SUBROUTINE IN PARSE THAT BRINGS A LINE OF INPUT IN

RN() : A DUMMY NAME FOR IN() OR DN()

SCAN : A SUBROUTINE IN PARSE

SEARCH : A SUBROUTINE

```
                 :
     SEVEN       :  THE SYMBOL 7 IN ALPHAMERIC FORM
                 :
     SIX         :  THE SYMBOL 6 IN ALPHAMERIC FORM
                 :
     SMALL       :  THE WORD SMALL (THE SIXTH CHARACTER IS BLANK)
                 :
     SN()        :  IN() OR DN() MODIFIED TO CONTAIN THE INGREDIENTS
                 :  OR DEPENDENTS OF THE ARTIFICIAL ROOT NODE
                 :
     SORT        :  A SUBROUTINE
                 :
     SORTER      :  A SUBROUTINE
                 :
     STACK()     :  ARRAY THAT STORES THE NODE THAT IS THE FATHER OF THE
                 :  CURRENT BRANCH FOR EACH LEVEL AND THE NUMBER OF SONS
                 :  THAT IT HAS REMAINING
                 :
     STRING      :  A LOGICAL FUNCTION IN PARSE THAT IS TRUE WHEN THE
                 :  INPUT IS A STRING
                 :
     SWITCH      :  A LOGICAL VARIABLE USED TO INDICATE THE PATH INTO
                 :  THE INPUT ROUTINE
                 :
     TB          :  THE TOTAL NUMBER OF BRANCHES IN THE NETWORK AFTER
                 :  ADDING THOSE TO THE ARTIFICIAL ROOT NODE
                 :
     TEMP        :  VARIABLE USED AS TEMPORARY STORAGE WHILE SORTING
                 :
     TF()        :  THE TOTAL FLOAT, THE DIFFERENCE BETWEEN THE LONGEST
                 :  PATH THROUGH THE NETWORK AND THE LONGEST SUCH PATH
                 :  THROUGH A GIVEN NODE
                 :
     THREE       :  THE SYMBOL 3 IN ALPHAMERIC FORM
                 :
     TITLE()     :  ARRAY USED TO STORE THE ALPHAMERIC TITLES OF THE NODES
                 :
     TNT         :  THE TOTAL NUMBER OF BRANCHES IN THE NETWORK
                 :
     TSORT       :  A LOGICAL VARIABLE THAT IS TRUE IF THE NETWORK HAS
                 :  BEEN SORTED
                 :
     TWO         :  THE SYMBOL 2 IN ALPHAMERIC FORM
                 :
     VALUE       :  VARIABLE THAT CARRIES A REAL NUMBER BACK FROM PARSE
                 :
     WDINIT      :  A SUBROUTINE IN PARSE USED TO SET INITIAL CONDITIONS
                 :
     WORD        :  A SUBROUTINE
                 :
     X           :  ARGUMENT OF SEVERAL LOGICAL FUNCTIONS IN PARSE
                 :
     ZERO        :  A SUBROUTINE
```

2.5 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=FREELIST,UNIT=DISKPACK,RECORD=40,BLOCKING=30
FILE  3=DUMMY,UNIT=DISKPACK,RECORD=14,BLOCKING=30
FILE  4=DUMMY,UNIT=DISKPACK,RECORD=14,BLOCKING=30,AREA=200*10,SAVE=30
 $ INCLUDE 'PARSE'
C
C       DIMENSION ALL OF THE ARRAYS USED IN THE PROGRAM
        INTEGER NODE(500), INT(500), INGR(1000), POINT(500), LINK(1000),
       1        NI(500), PTIN(500), IN(1000), NO(500), PTON(500),
       2        ON(1000), IL(500), OL(500), TF(500), TNI
C       BOTH THE FOLLOWING ARRAYS CONTAIN ALPHAMERIC WORDS
        REAL LABEL(500), TITLE(10,500), ALPHA1, ALPHA2, ALPHA3, ALPHA4,
       1  SMALL, LARGE, LV, FLO
        COMMON TITLE
C       THE FOLLOWING ARE USED AS LOGIC TRACES
        LOGICAL SWITCH, ISORT, UNPACK, FORM
C       THE FOLLOWING WORDS ARE USED IN OUTPUT
        DATA SMALL, LARGE, LV, FLO/"SMALL ", "LARGE ",
       1  "LEVEL ", "FLOAT "/
C
C       DECLARATIONS FOR USE OF THE INPUT SCANNER
        LOGICAL END, FIXED, MATCH, STRING, NEXT
        COMMON/SCANER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
        EQUIVALENCE (IVALUE,VALUE)
C       DELETE THE ECHO PRINT OF THE SCANNER
        DATA PRI/"OFF"/
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
        CALL WDINIT(10,72,PRI)
C
C       READ THE OUTPUT UNIT IN
        KOUT = 6
        WRITE(KOUT,1005)
        CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
        IF(.NOT. MATCH("P",1)) GO TO 10
        KOUT = 1
        CALL SETOUT(KOUT)
C
C       SET THE ECHO
        WRITE(6,1015)
        CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
        IF(MATCH("YES",3)) CALL WDINIT(10,72,"ON    ")
C
C       INITIALIZE THE ARRAYS
     10 CALL INIT(NODE, INT, INGR, POINT, LINK, NI, PTIN, IN, NO,
       1         PTON, ON, IL, OL, TF, LABEL)
```

```
C
C      READ THE DATA FILE NAME IN
       K = 0
    20 WRITE(6,1000)
       CALL RDLINE
       ENTITY(NWD+1) = 6H.
       CLOSE(3)
       CHANGE(3,TITLE=ENTITY)
       CHANGE(4,TITLE=ENTITY)
C
C      CHECK FOR EXISTING DATAFILE
       INQUIRE(3, RESIDENT=ONPACK)
       IF(.NOT. ONPACK) GO TO 50
C
C      EXISTING FILE, DOUBLE CHECK THE NAME
       WRITE(6,1010)
       CALL RDLINE
       IF(MATCH("YES",3)) GO TO 65
       GO TO 20
C
C      READ NEW NETWORK DATA IN FROM THE TERMINAL
    50 SWITCH = .TRUE.
       CALL INPUT(SWITCH, NN, TNI, L, NODE, INT, LABEL, NI,
      1          POINT, INGR, LINK, LNN)
       GO TO 70
C
C      READ THE OLD NETWORK DATA FROM THE DATAFILE
    65 CALL OLDATA(NN, TNI, L, NODE, INT, LABEL, NI, POINT,
      1          INGR, LINK, LNN)
C
C      CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
       WRITE(6,1020)
       CALL RDLINE
       IF(.NOT.MATCH("YES",3)) GO TO 70
       SWITCH = .FALSE.
       CALL INPUT(SWITCH, NN, TNI, L, NODE, INT, LABEL, NI,
      1          POINT, INGR, LINK, LNN)
C
C      CONSTRUCT THE SIMPLIFIED INGREDIENTS LIST
    70 CALL INGRED(NN, TNI, NI, POINT, INGR, LINK, PTIN, IN)
C
C      CONSTRUCT THE SIMPLIFIED DEPENDENTS LIST
       CALL DEPEN(NN, TNI, NI, PTIN, IN, ND, PTDN, DN)
C
C      CALCULTE THE TOPOLOGIC LEVELS OF THE NETWORK
       CALL LEVEL(NN, TNI, NI, PTIN, ND, PTDN, DN, IL, OL, TF, LENGTH)
C
C      SET THE TRACE INDICATING THE NO SORT HAS BEEN DONE
       TSORT = .FALSE.
C
C      BEGIN SCANNING FOR OPERATING COMMANDS
C
       K = 0
   100 WRITE(6,1030)
       CALL RDLINE
   110 IF(END(X)) GO TO 100
       IF(MATCH("SORT",3)) GO TO 120
       IF(MATCH("INGRED",3)) GO TO 160
       IF(MATCH("DEPEND",3)) GO TO 190
       IF(MATCH("WRITE",3)) GO TO 220
       IF(MATCH("MODIFY",3)) GO TO 230
       IF(MATCH("STOP",3)) STOP
```

```
      IF(MATCH("NEXT",3)) GO TO 10
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 100
C
C     SORTING DONE HERE. FIND THE PARAMETERS
C
  120 TSORT = .TRUE.
C
C     FIND THE VALUE FOR FIRST PRIORITY SORTING
      WRITE(6,1040)
      CALL RDLINE
      M1 = 1
      ALPHA2 = FLO
      ALPHA4 = LV
      IF(MATCH("FLOAT",3)) GO TO 130
      M1 = 2
      ALPHA2 = LV
      ALPHA4 = FLO
      IF(MATCH("LEVEL",3)) GO TO 130
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 120
C
C     FIND THE MODE FOR FIRST PRIORITY SORTING
  130 WRITE(5,1050)
      CALL RDLINE
      M2 =1
      ALPHA1 = SMALL
      IF(MATCH("SMALL",3)) GO TO 140
      M2 = 2
      ALPHA1 = LARGE
      IF(MATCH("LARGE",3)) GO TO 140
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 130
C
C     FIND THE MODE FOR SECOND PRIORITY SORTING
  140 WRITE(6,1060)
      CALL RDLINE
      M3 = 1
      ALPHA3 = SMALL
      IF(MATCH("SMALL",3)) GO TO 150
      M3 = 2
      ALPHA3 = LARGE
      IF(MATCH("LARGE",3)) GO TO 150
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 140
C
C     CALL THE SORTING ROUTINE
  150 CALL SORTER(M1, M2, M3, NN, TN1, N1, PTIN, IN, ND, PTDN,
     1                DN, UL, IL, TF)
      K = 0
      GO TO 100
C
C     GLOBAL INGREDIENCE DESIRED HERE, FIND THE DESIRED ROOT
C
  160 FORM = .FALSE.
      IF(MATCH("TITLE",3)) FORM = .TRUE.
      WRITE(6,1070)
      CALL RDLINE
```

```
      IF(MATCH("COMPLE",3)) GO TO 170
      IF(FIXED(X)) GO TO 180
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 160
C
C     DO FOR THE COMPLETE NETWORK
  170 CALL SEARCH(NN, INI, NO, NI, PTIN, IN, OL, KOUNT, NODE,
     1     LABEL, TSORT, M1, M2, M3, TF, FORM)
      CALL OUTPUT(3, KOUT, 0, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
     1 ALPHA4, LENGTH, FORM)
      K = 0
      GO TO 100
C
C     DO FOR ONE NODE
  180 N = INT(IVALUE)
      CALL GLOBAL(N, NN, INI, NI, PTIN, IN, OL, 1, KOUNT,
     1             NODE, LABEL, FORM)
      CALL OUTPUT(1, KOUT, IVALUE, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3
     1   ALPHA4, LENGTH, FORM)
      K = 0
      GO TO 100
C
C     GLOBAL DEPENDENCE DESIRED HERE, FIND THE DESIRED ROOT
C
  190 FORM = .FALSE.
      IF(MATCH("TITLE",3)) FORM = .TRUE.
      WRITE(6,1070)
      CALL RDLINE
      IF(MATCH("COMPLE",3)) GO TO 200
      IF(FIXED(X)) GO TO 210
      K = K + 1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 190
C
C     DO FOR THE COMPLETE NETWORK
  200 CALL SEARCH(NN, INI, NI, NO, PTON, ON, IL, KOUNT, NODE,
     1     LABEL, TSORT, M1, M2, M3, TF, FORM)
      CALL OUTPUT(4, KOUT, 0, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
     1 ALPHA4, LENGTH, FORM)
      K = 0
      GO TO 100
C
C     DO FOR ONE NODE
  210 N = INT(IVALUE)
      CALL GLOBAL(N, NN, INI, NO, PTON, ON, IL, 1, KOUNT,
     1             NODE, LABEL, FORM)
      CALL OUTPUT(2, KOUT, IVALUE, KOUNT, TSORT, ALPHA1, ALPHA2, ALPHA3,
     1   ALPHA4, LENGTH, FORM)
      K = 0
      GO TO 100
C
C     WRITE OUT THE COMPLETE DATA
C
  220 CALL DATOUT(NN, INI, NODE, INI, LABEL, NI, PTIN,
     1             IN, NO, PTON, ON, IL, OL, TF)
      WRITE(KOUT,5000)
      K = 0
      GO TO 100
C
C     MODIFY THE BASIC INPUT DATA
C
```

```
  230 SWITCH = .FALSE.
      CALL INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL, NI,
     1          POINT, INGR, LINK, LNN)
      K = 0
C
      CALL ZERO(PTIN, IN, NO, PTON, ON, IL, OL, TF)
      GO TO 70
C
 1000 FORMAT(1H ,"ENTER THE DATA FILE NAME",/)
 1005 FORMAT(1H , "ENTER P FOR OUTPUT ON THE ON-SITE PRINTER",
     1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL",/)
 1010 FORMAT(1H ,"FILE EXISTS WITH THIS NAME. DO YOU WANT TO USE IT?",/)
 1015 FORMAT(1H ,"DO YOU WANT TO HAVE THE INPUT ECHOED ON OUTPUT",/)
 1020 FORMAT(1H ,"DO YOU WANT TO MODIFY THE EXISTING DATA?",/)
 1030 FORMAT(1H ,"ENTER A PROGRAM COMMAND",/)
 1040 FORMAT(1H ,"ENTER THE VALUE FOR FIRST PRIORTY SORTING",/)
 1050 FORMAT(1H ,"ENTER THE MODE FOR FIRST PRIORTY SORTING",/)
 1060 FORMAT(1H ,"ENTER THE MODE FOR SECOND PRIORTY SORTING",/)
 1070 FORMAT(1H ,"ENTER THE ROOT NODE NUMBER",
     1        "---OR THE WORD 'COMPLETE'",/)
 5000 FORMAT(1H1)
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INIT(NODE, INT, INGR, POINT, LINK, NI, PTIN, IN,
     1                ND, PTON, ON, IL, OL, TF, LABEL)
      INTEGER NODE(1), INT(1), INGR(1), POINT(1), LINK(1), NI(1),
     1        PTIN(1), IN(1), ND(1), PTON(1), ON(1), IL(1), OL(1), TF(1)
      REAL LABEL(1), TITLE(10,500)
      COMMON TITLE
      DATA BLANK/6H      /
      DO 10 I = 1, 500
   10 NODE(I) = 0
      DO 20 I = 1, 500
   20 INT(I) = 0
      DO 30 I = 1, 500
   30 POINT(I) = 0
      DO 40 I = 1, 500
   40 NI(I) = 0
      DO 50 I = 1, 500
   50 PTIN(I) = 0
      DO 60 I = 1, 500
   60 ND(I) = 0
      DO 70 I = 1, 500
   70 PTON(I) = 0
      DO 80 I = 1, 500
   80 IL(I) = 0
      DO 90 I = 1, 500
   90 OL(I) = 0
      DO 100 I = 1, 500
  100 TF(I) = 0
      DO 110 I = 1, 500
      DO 110 J = 1, 10
  110 TITLE(J,I) = BLANK
      DO 120 I = 1, 1000
  120 INGR(I) = 0
      DO 130 I = 1, 1000
  130 LINK(I) = 0
      DO 140 I = 1, 1000
  140 IN(I) = 0
      DO 150 I = 1, 1000
```

```
      150 DN(I) = 0
          RETURN
          END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
          SUBROUTINE INPUT(SWITCH, NN, INI, L, NODE, INT, LABEL,
         1                    NI, POINT, INGR, LINK, LNN)
C
C         ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
          INTEGER INI, NODE(1), INT(1), NI(1), POINT(1), INGR(1),
         1        LINK(1)
          REAL LABEL(1), TITLE(10,500), BLANK
          COMMON TITLE
          DATA BLANK/6H      /
          LOGICAL SWITCH, EXIT
C
C         DECLARATIONS FOR THE USE OF THE INPUT SCANNER
          LOGICAL END, FIXED, MATCH, STRING, NEXT
          COMMON/SCANER/ ENTITY(20), MODE, VALUE, NCHAR, NWD, NEXT
          EQUIVALENCE (IVALUE,VALUE)
C
C         CHECK THE SWITCH FOR NEW DATA OR MODIFICATION OF OLD DATA
          IF(SWITCH) GO TO 10
C
C         OLD DATA TO BE MODIFIED
          I = NN
          GO TO 15
C
C         NEW DATA, SET INITIAL CONDITIONS
       10 L = 0
          I = 0
          LNN = 0
C
C         SCAN FOR THE NODE NUMBER
       15 K = 0
       20 WRITE(6,1020)
          CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
       30 IF(FIXED(X)) GO TO 40
          EXIT = .FALSE.
          IF(MATCH("SAVE",3)) GO TO 150
          EXIT = .TRUE.
          IF(MATCH("END",3)) GO TO 150
C         IMPROPER INPUT DATA
          K = K + 1
          IF(K.EQ.10) CALL ERROR(1)
          WRITE(6,1000)
          CALL RDLINE
          GO TO 30
C
C         HAVE THE NODE NUMBER, CHECK FOR PREVIOUS ENTRY
       40 N = IVALUE
          IF(INT(N).NE.0) GO TO 90
C
C         NODE HAS NOT BEEN ENTERED BEFORE
          I = I + 1
          IF(LNN .LT. N) LNN = N
          INT(N) = I
          NODE(I) = N
          J = I
          M = 0
C
```

```
C       LOOK FOR LABEL, TITLE, AND INGREDIENTS
     50 IF(FIXED(X)) GO TO 60
        IF(MODE.EQ.3) GO TO 65
        IF(MODE.EQ.7) GO TO 70
        IF(END(X)) GO TO 55
C       IMPROPER INPUT DATA
        WRITE(6,1000)
        CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
        K = K + 1
        IF(K.EQ.10) CALL ERROR(1)
        GO TO 50
C
C       TEST FOR CONTINUATION OF DATA
     55 IF(MATCH(",",1)) GO TO 50
        GO TO 30
C
C       HERE WE HAVE AN INGREDIENT,  PUT IT IN THE LISTS
     60 CALL INGLIS(I, J, L, M, IVALUE, INT, NODE, NI, POINT, INGR, LINK)
        GO TO 50
C
C       HERE WE HAVE A LABEL
     65 LABEL(J) = ENTITY(1)
        CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
        GO TO 50
C
C       HERE WE HAVE A DESCRIPTION, PUT IT INTO THE TITLE ARRAY
     70 NWDS = (NCHAR + 5)/6
        DO 80 IJ = 1, NWDS
     80 TITLE(IJ,J) = ENTITY(IJ)
        CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
        GO TO 50
C
C       NODE HAS BEEN ENTERED BEFORE
     90 J = INT(N)
C
C       SCAN FOR NEW INGREDIENTS, LABELS, OR TITLES
    100 IF(FIXED(X)) GO TO 110
        IF(MODE.EQ.3) GO TO 125
        IF(MODE.EQ.7) GO TO 130
        IF(END(X)) GO TO 105
C       IMPROPER INPUT DATA
        WRITE(6,1000)
        CALL SCAN(ENTITY, MODE, VALUE, NCHAR)
        K = K + 1
        IF(K.EQ.10) CALL ERROR(1)
        GO TO 100
C
C       CHECK FOR CONTINUATION OF DATA
    105 IF(MATCH(",",1)) GO TO 120
        GO TO 30
C
C       HERE WE HAVE A REVISED LIST OF INGREDIENTS, CHECK THE FIRST ONE
    110 IF(IVALUE.NE.0) GO TO 115
C       CHANGE---THE NODE HAS NO INGREDIENTS
        NI(J) = 0
        POINT(J) = 0
        GO TO 100
C       ADD TO THE LIST OF INGREDIENTS
    115 M = 0
    120 CALL INGLIS(I, J, L, M, IVALUE, INT, NODE, NI, POINT, INGR, LINK)
C       CHECK FOR MORE INGREDIENTS
        IF(.NOT.FIXED(X)) GO TO 100
```

```
      GO TO 120
C
C     HERE WE HAVE A NEW LABEL
  125 LABEL(J) = ENTITY(1)
      CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
      GO TO 100
C
C     HERE WE HAVE A NEW DESCRIPTION
  130 DO 135 IJ = 1, 10
  135 TITLE(IJ,J) = BLANK
      NWDS = (NCHAR + 5)/6
      DO 140 IJ = 1, NWDS
  140 TITLE(IJ,J) = ENTITY(IJ)
      CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
      GO TO 100
C
C     END OF INPUT DATA, STORE ON DISC
C
  150 REWIND 4
      NN = I
      TNI = 0
      DO 160 II = 1, LNN
  160 TNI = TNI + NI(II)
      WRITE(4,1010) NN, TNI, L, LNN
      DO 170 I = 1, LNN
  170 WRITE(4,1010) NODE(I), INT(I), NI(I), POINT(I), LABEL(I)
      DO 180 I = 1, L
  180 WRITE(4,1010) INGR(I), LINK(I)
      DO 190 I = 1, LNN
  190 WRITE(4,1030) (TITLE(J,I),J=1,10)
      LOCK 4
      I = NN
      IF(.NOT. EXIT) GO TO 30
C
 1000 FORMAT(1H ,"INPUT ERROR---RE-ENTER ON A  NEW LINE",/)
 1010 FORMAT(1H , 4I6, A6)
 1020 FORMAT(1H ,"ENTER THE NODE NUMBERS AND ASSOCIATED DATA, ONE NODE
     1TO A LINE.",/," IF IT IS NECESSARY TO USE TWO LINES FOR THE"
     2" INGREDIENTS,",/," ENTER A COMMA AT THE BEGINNING OF THE ",
     3"SECOND LINE.",/" ENTER 'END' TO SIGNIFY THE END OF THE DATA.",/)
 1030 FORMAT(1H ,10A6)
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INGLIS(I, J, L, M, N, INT, NODE, NI, POINT, INGR, LINK)
C
C     ALL OF THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
      INTEGER INT(1), NODE(1), NI(1), POINT(1), INGR(1), LINK(1)
C
C     INCREMENT THE COUNTERS
      M = M + 1
      L = L + 1
      NI(J) = M
      POINT(J) = L
C
C     CHECK TO SEE IF THIS IS THE FIRST INGREDIENT
      IF(M.EQ.1) LINK(L) = 0
      IF(M.NE.1) LINK(L) = L - 1
C
C     CHECK TO SEE IF INGREDIENT NODE HAS BEEN ENTERED BEFORE
```

```
      IF(INT(N).EQ.0) GO TO 10
C
C     INGREDIENT NODE HAS BEEN ENTERED BEFORE
      INGR(L) = INT(N)
      GO TO 20
C
C     INGREDIENT NODE HAS NOT BEEN ENTERED BEFORE
   10 I = I + 1
      INT(N) = I
      NODE(I) = N
      INGR(L) = I
   20 RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE D-DATA(NN, TNI, L, NODE, INT, LABEL, NI,
     1                       POINT, INGR, LINK, LNN)
C
C     ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
      INTEGER TNI, NODE(1), INT(1), NI(1), POINT(1),
     1        INGR(1), LINK(1)
      REAL LABEL(1), TITLE(10,500)
      COMMON TITLE
C
C     FILL THE ARRAYS FROM FILE 3
      REWIND 3
      READ(3,1000) NN, TNI, L, LNN
      IF(LNN .LT. NN) LNN= NN
      DO 10 I = 1, LNN
   10 READ(3,1010) NODE(I), INT(I), NI(I), POINT(I), LABEL(I)
      DO 20 I = 1, L
   20 READ(3,1020) INGR(I), LINK(I)
      DO 30 I = 1, LNN
   30 READ(3,1030) (TITLE(J,I),J=1,10)
      REWIND 3
 1000 FORMAT(1H , 4I6)
 1010 FORMAT(1H , 4I6, A6)
 1020 FORMAT(1H , 2I6)
 1030 FORMAT(1H , 10A6)
      RETURN
      END
C
C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ZERO(PTIN, IN, ND, PTON, ON, IL, OL, TF)
      INTEGER PTIN(1), IN(1), ND(1), PTON(1), ON(1),
     1  IL(1), OL(1), TF(1)
      DO 10 I = 1, 500
      PTIN(I) = 0
      ND(I) = 0
      PTON(I) = 0
      IL(I) = 0
      OL(I) = 0
      TF(I) = 0
   10 CONTINUE
      DO 20 I = 1, 1000
      IN(I) = 0
      ON(I) = 0
   20 CONTINUE
      RETURN
      END
```

```
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INGRED(NN, INI, NI, POINT, INGR, LINK, PTIN, IN)
C
C     ALL THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
      INTEGER INI, NI(1), POINT(1), INGR(1), LINK(1), PTIN(1), IN(1)
C
C     CONSTRUCT THE POINTER PTIN FROM NI
      PTIN(1) = 1
      DO 10 I = 2, NN
   10 PTIN(I) = PTIN(I-1) + NI(I-1)
C
C     FILL THE SIMPLE INGREDIENT LIST IN
      K = 1
      DO 50 I = 1, NN
      IF(NI(I)-1) 50, 20, 30
   20 IN(K) = INGR(POINT(I))
      K = K + 1
      GO TO 50
   30 M = POINT(I)
      DO 40 J = 1, NI(I)
      IN(K) = INGR(M)
      K = K + 1
      M = LINK(M)
   40 CONTINUE
   50 CONTINUE
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE DEPEN(NN, INI, NI, PTIN, IN, ND, PTDN, DN)
      INTEGER INI, NI(1), PTIN(1), IN(1), ND(1), PTDN(1), DN(1)
C
C     CALCULATE THE NUMBER OF DEPENDENTS
      DO 10 I = 1, INI
   10 ND(IN(I)) = ND(IN(I)) + 1
C
C     CUNSTRUCI THE POINTER FOR THE DEPENDENT LIST
      PTDN(1) = 1
      DO 30 I = 2, NN
   30 PTDN(I) = PTDN(I - 1) + ND(I - 1)
C
C     CONSTRUCT THE DEPENDENT LIST
      DO 50 I = 1, NN
      IF (NI(I) .EQ. 0) GO TO 50
      IP1 = PTIN(I)
      IP2 = IP1 + NI(I) - 1
      DO 40 J = IP1, IP2
      DN(PTDN(IN(J))) = I
   40 PTDN(IN(J)) = PTDN(IN(J)) + 1
   50 CUNIINUE
C
C     CORRECT IHE POINTER FOR THE DEPENDENT LIST
      DO 60 I = 1, NN
   60 PTDN(I) = PTDN(I) - ND(I)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
```

```
      SUBROUTINE LEVEL(NN, TNI, NI, PTIN, NO, PTON, DN, IL, OL, TF,
     1                 LENGTH)
C
C     ALL THE FOLLOWING ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
      INTEGER TNI, NI(1), PTIN(1), NO(1), PTON(1), DN(1), IL(1),
     1        OL(1), TF(1)
C
C     ORDER IS A LOCAL VARIABLE
      INTEGER ORDER(500)
C
C     INITIALIZE THE COUNTERS AND ORDER
      JL = 0
      JF = 0
      DO 10 I = 1, 500
   10 ORDER(I) = 0
C
C     BUILD AN ORDERED LIST OF THE NETWORK BY LOOKING FOR
C     STARTING NODES (THOSE WITH NO INGREDIENTS)
      DO 40 I = 1, NN
      IF(NI(I).NE.0) GO TO 40
C     HERE WE HAVE A NODE WITH NO REMAINING INGREDIENTS,
C     PUT IT IN THE LIST
      NI(I) = -1
      JL = JL + 1
      ORDER(JL) = I
C
C     PROCESS THE DEPENDENTS OF THIS NODE BY REDUCING THEIR
C     REMAINING INGREDIENTS BY ONE
   20 JF = JF + 1
      JJ = ORDER(JF)
      IF(NO(JJ).EQ.0) GO TO 35
      L1 = PTON(JJ)
      L2 = L1 - 1 + NO(JJ)
      DO 30 LI = L1, L2
      LL = DN(LI)
      NI(LL) = NI(LL) - 1
C
C     CHECK FOR MORE NODES WITH NO REMAINING INGREDIENTS
C     AMONG THESE DEPENDENTS
      IF(NI(LL).NE.0) GO TO 30
C     HERE WE HAVE A NODE WITH NO REMAINING INGREDIENTS,
C     PUT IT IN THE LIST
      NI(LL) = -1
      JL = JL + 1
      ORDER(JL) = LL
   30 CONTINUE
C     CHECK TO SEE IF ALL THE NODES NOW IN "ORDER" HAVE
C     HAD THEIR DEPENDENTS PROCESSED
   35 IF(JF.LT.JL) GO TO 20
   40 CONTINUE
C
C     CHECK NETWORK FOR ERRORS
      IF(JL.EQ.0) CALL ERROR(2)
      IF(JL.NE.NN) CALL ERROR(3)
C
C     CORRECT THE VALUES OF NI
      J = NN - 1
      DO 50 I = 1, J
   50 NI(I) = PTIN(I + 1) - PTIN(I)
      NI(NN) = TNI - PTIN(NN) + 1
C
C     CALCULATE THE EXTREME DISTANCE FROM INPUT FOR EACH NODE---
```

```
C      THIS IS SIMILAR TO THE EARLY START IN A CPM ROUTINE
       NF = ORDER(1)
       LENGTH = 0
       DO 80 I = 1, NN
       J = ORDER(I)
       IF(ND(J).EQ.0) GO TO 70
       L1 = PTON(J)
       L2 = L1 + ND(J) - 1
       DO 60 LI = L1, L2
       M = DN(LI)
       IF(IL(M).LT.(IL(J) + 1)) IL(M) = IL(J) + 1
    60 CONTINUE
    70 IF(LENGTH.GE.(IL(J) + 1)) GO TO 80
       NF = J
       LENGTH = IL(J) + 1
    80 CONTINUE
C
C      CALCULATE THE EXTREME DISTANCE FROM OUTPUT FOR EACH NODE---
C      THIS IS A MODIFICATION OF THE LATE FINISH IN A CPM ROUTINE
       K = NN
    90 J = ORDER(K)
       OL(J) = LENGTH
       IF(ND(J).EQ.0) GO TO 110
       L1 = PTON(J)
       L2 = L1 + ND(J) - 1
       DO 100 LI = L1, L2
       M = DN(LI)
       IF((OL(M) - 1).LT.OL(J)) OL(J) = OL(M) - 1
   100 CONTINUE
   110 K = K - 1
       IF(K.NE.0) GO TO 90
C
C      CALCULATE THE OUTPUT LEVEL AND THE TOTAL FLOAT
       DO 120 J = 1, NN
       OL(J) = LENGTH - OL(J)
   120 TF(J) = LENGTH - OL(J) - IL(J) - 1
       RETURN
       END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
       SUBROUTINE SORTER(M1, M2, M3, NN, TNI, NI, PTIN, IN, ND, PTON, DN,
      1OL, IL, TF)
C
C      ALL OF THE ARRAYS ARE DIMENSIONED  IN THE CALLING PROGRAM
       INTEGER TNI, NI(1), PTIN(1), IN(1), ND(1), PTON(1),
      1        DN(1), OL(1), IL(1), TF(1)
C
C      DETERMINE THE ORDER OF SORTING
       GO TO (10, 20) M1
C
C      SORTING BY FLOAT, THEN BY LEVEL
C      SORT THE INGREDIENTS
    10 CALL SORT(NN, TNI, NI, PTIN, IN, TF, OL, M2, M3)
C      SORT THE DEPENDENTS
       CALL SORT(NN, TNI, ND, PTON, DN, TF, IL, M2, M3)
       RETURN
C
C      SORTING BY LEVEL, THEN BY FLOAT
C      SORT THE INGREDIENTS
    20 CALL SORT(NN, TNI, NI, PTIN, IN, OL, TF, M2, M3)
C      SORT THE DEPENDENTS
```

```
      CALL SORT(NN, INI, ND, PTDN, DN, IL, TF, M2, M3)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE SORT(NN, INI, NR, PTRN, RN, A, B, M2, M3)
C
C     ALL OF THE ARRAYS ARE DIMENSIONED  IN THE CALLING PROGRAM
      INTEGER INI, TEMP, NR(1), PTRN(1), RN(1), A(1), B(1)
C
C     SET THE MULTIPLIERS FOR NEAREST AND FARTHEST MODES OF SORTING
      MULT1 = 1
      MULT2 = 1
      IF(M2.EQ.2) MULT1 = -1
      IF(M3.EQ.2) MULT2 = -1
C
C     ORDER THE BRANCHES AT EACH NODE BY A AND B
      DO 30 I = 1, NN
C
C     CHECK FOR MULTIPLE BRANCHES
      IF(NR(I).LT.2) GO TO 30
      N = NR(I) - 1
      DO 30 JJ = 1, N
      J = JJ - 1
      DO 30 K = JJ, N
C
C     SORT FIRST BY A
      IF(MULT1*(A(RN(PTRN(I)+J)) - A(RN(PTRN(I)+K)))) 30, 20, 10
C
C     SWITCH BRANCHES J AND K
   10 TEMP = RN(PTRN(I)+J)
      RN(PTRN(I)+J) = RN(PTRN(I)+K)
      RN(PTRN(I)+K) = TEMP
      GO TO 30
C
C     SORT ANY TIES BY B
   20 IF(MULT2*(B(RN(PTRN(I)+J))-B(RN(PTRN(I)+K)))) 30,30,10
   30 CONTINUE
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE SEARCH(NN, INI, NP, NR, PTRN, RN, LEV, KOUNT,
     1                 NODE, LABEL, TSORT, M1, M2, M3, TF, FORM)
C
C     THE FOLLOWING ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
      INTEGER INI, NP(1), NR(1), PTRN(1), RN(1), LEV(1),
     1          NODE(1), TF(1)
      REAL LABEL(1)
      LOGICAL TSORT, FORM
C
C     SN DOES NOT APPEAR IN THE MAIN PROGRAM AND IS DIMENSIONED HERE
      INTEGER SN(500), TB
C
C     CONNECT ALL THE STARTING NODES TO THE ARTIFICIAL
C     NODE N AND MODIFY NR, PTRN, AND RN ACCORDINGLY
      N = NN + 1
      NR(N) = 0
      I = 0
      DO 10 J = 1, NN
```

```
      IF(NP(J).NE.0) GO TO 10
      I = I + 1
      SN(I) = J
      NR(N) = NR(N) + 1
   10 CONTINUE
      TB = TNI + I
      PTRN(N) = TNI + 1
      DO 20 J = PTRN(N), TB
      I = J - TNI
   20 RN(J) = SN(I)
C
C     CHECK TO SEE IF THE NEW BRANCHES SHOULD BE SORTED
      IF(.NOT.TSORT) GO TO 50
C
C     SORT IS DESIRED, DETERMINE THE ORDER
      GO TO (30,40) M1
C     SORT BY FLOAT, THEN BY LEVEL
   30 CALL SORT(1, I, NR(N), PTRN(N), RN, TF, LEV, M2, M3)
      GO TO 50
C     SORT BY LEVEL, THEN BY FLOAT
   40 CALL SORT(1, I, NR(N), PTRN(N), RN, LEV, TF, M2, M3)
C
C     FIND THE GLOBAL INGREDIENCE (DEPENDENCE) OF NODE N
   50 NUM = N
      CALL GLOBAL(N, N, TB, NR, PTRN, RN, LEV, 0, KOUNT, NODE, LABEL,
     1            FORM)
C
C     CORRECT NR, PTRN, AND RN TO THEIR ORIGINAL VALUES
      NR(NUM) = 0
      DO 60 J = PTRN(NUM), TB
   60 RN(J) = 0
      PTRN(NUM) = 0
      RETURN
      END
C
C     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE GLOBAL(N, NN, TNI, NR, PTRN, RN, LEV, KEY, KOUNT,
     1                  NODE, LABEL, FORM)
C
C     THESE ARRAYS ARE DIMENSIONED IN THE MAIN PROGRAM
      INTEGER TNI, NODE(1), NR(1), PTRN(1), RN(1), LEV(1)
      REAL LABEL(1)
      LOGICAL FORM
C
C     GLOB, HIT AND STACK ARE VARIABLES WHICH DO NOT APPEAR
C     IN THE MAIN PROGRAM AND ARE DIMENSIONED HERE
      INTEGER GLOB(1000), HIT(500), STACK(20,2)
C
C     NR AND PTRN ARE BOTH DESTROYED IN THE ROUTINE
C     SO A DUPLICATE OF NR IS PRODUCED HERE
      INTEGER NNR(500)
      DO 10 I = 1, NN
   10 NNR(I) = NR(I)
C
C     INITIALIZE GLOB, HIT, AND STACK
      DO 20 I = 1, 1000
   20 GLOB(I) = 0
      DO 30 I = 1, NN
   30 HIT(I) = 0
      DO 40 I = 1, 20
      STACK(I,1) = 0
```

```
   40 STACK(I,2) = 0
C
C     BEGIN PROCESSING THE BRANCHES OF THE NETWORK IN PREORDER
      M = 1
      L = 0
      REWIND 2
      KOUNT = 0
C
C     INSERT THE ROOT NODE INTO THE OUTPUT LIST,
C     UNLESS IT IS IMAGINERY
      IF(KEY.EQ.1) CALL LIST(N, L, STACK, LEV, NNR(N), NODE(N),
     1      LABEL(N), FORM)
      IF(KEY.EQ.1) KOUNT = 1
   50 GLOB(M) = N
      IF( NR(N).EQ.0 ) GO TO 80
      L = L + 1
      STACK(L,1) = N
   60 NR(N) = NR(N) - 1
      STACK(L,2) = NR(N)
      M = M + 1
      J = RN(PTRN(N))
      PTRN(N) = PTRN(N) + 1
      IF( HIT(J).NE.0 ) GO TO 70
      HIT(J) = 1
      N = J
      CALL LIST(N, L, STACK, LEV, NNR(N), NODE(N), LABEL(N), FORM)
      KOUNT = KOUNT + 1
      GO TO 50
   70 GLOB(M) = -J
      CALL LIST(GLOB(M), L, STACK, LEV, NNR(J), NODE(J), LABEL(J), FORM)
      KOUNT = KOUNT + 1
   80 IF(L.EQ.0) GO TO 100
      IF(STACK(L,2).EQ.0) GO TO 90
      N = STACK(L,1)
      GO TO 60
   90 L = L - 1
      GO TO 80
C
C     CORRECT THE VALUES OF NR AND PTRN
  100 PTRN(1) = 1
      NR(1) = NNR(1)
      DO 110 I = 2, NN
      NR(I) = NNR(I)
  110 PTRN(I) = PTRN(I - 1) + NNR(I - 1)
      END FILE 2
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE LIST(N, L, STACK, LEV, NNR, NODE, LABEL, FORM)
C
C     STACK AND LEV ARE DIMENSIONED IN THE CALLING PROGRAM
      INTEGER STACK(20,2), LEV(1)
C
C     LINE IS THE VARIABLE WHICH CONTAINS A LINE OF OUTPUT
      REAL LINE(234), MINUS, LABEL
      DATA BLANK, COLON, DOT, MINUS, ASTER/1H ,1H:,1H.,1H-,1H*/
      LOGICAL FORM
C
C     INITILIZE THE LINE TO ALL BLANKS
      DO 10 I = 1, 234
```

```
   10 LINE(I) = BLANK
      IF(L.EQ.0) GO TO 25
C
C     PUT COLONS BELOW ALL "FATHERS" OF THE NODE
      DO 20 I = 1, L
      J = LEV(STACK(I,1))
      IF(STACK(I,2).EQ.0 .AND. I.NE.L) GO TO 20
      LINE(10*J + 4) = COLON
   20 CONTINUE
C
C     PUT DOTS FROM THE LAST COLON TO THE NODE
   25 LN = LEV(IABS(N))
      I1 = 10*J + 5
      I2 = 10*LN + 3
      IF(LN.EQ.0) GO TO 35
      DO 30 I = I1, I2
   30 LINE(I) = DOT
C
C     BREAK THE EXTERNAL NUMBER OF THE NODE INTO DIGITS AND ENTER
C     THEM IN THE APPROPRIATE SPACES IN THE LINE
   35 N100 = IABS(NODE)/100
      N10 = IABS(NODE)/10 - 10*N100
      N1 = IABS(NODE) - 10*N10 - 100*N100
      IF(N1.EQ.0) N1=10
      IF(N100.NE.0) GO TO 40
      IF(N10.NE.0) GO TO 50
      GO TO 60
   40 CALL NUMBER(LINE(I2-1),N100)
      IF(N10.EQ.0) N10 = 10
   50 CALL NUMBER(LINE(I2),N10)
   60 CALL NUMBER(LINE(I2+1),N1)
C
C     ENTER A NEGATIVE SIGN IF THE NODE HAS BEEN PRINTED BEFORE
      IF(N.LT.0 .AND. N100.NE.0 .AND. N10.NE.0) LINE(I2-2) = MINUS
      IF(N.LT.0 .AND. N100.EQ.0 .AND. N10.NE.0) LINE(I2-1) = MINUS
      IF(N.LT.0 .AND. N100.EQ.0 .AND. N10.EQ.0) LINE(I2) = MINUS
      IF(N.LT.0 .AND. NNR.NE.0) LINE(I2+2) = ASTER
C
C     PUT THE LABEL OR TITLE OF THE NODE IN THE LINE
      IF(.NOT. FORM) CALL WORD(LABEL, LINE, I2)
      IF(FORM) CALL WORD1(LINE, I2, ABS(N))
C
C     WRITE THE LINE ON THE DISK
      WRITE(2,70) LINE
   70 FORMAT(1H ,234A1)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE NUMBER(LINEI ,NUMB)
C
C     THIS ROUTINE IS USED TO ENTER THE PROPER DIGIT IN THE
C     PROPER SPACE IN THE LINE
      REAL LINEI,NINE
      DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,NOUGHT/
     1"1","2","3","4","5","6","7","8","9","0"/
      GO TO(1,2,3,4,5,6,7,8,9,10) NUMB
    1 LINEI = ONE
      RETURN
    2 LINEI = TWO
      RETURN
```

```
    3 LINEI = THREE
      RETURN
    4 LINEI = FOUR
      RETURN
    5 LINEI = FIVE
      RETURN
    6 LINEI = SIX
      RETURN
    7 LINEI = SEVEN
      RETURN
    8 LINEI = EIGHT
      RETURN
    9 LINEI = NINE
      RETURN
   10 LINEI = NOUGHT
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE WORD(LABEL, LINE, I2)
C
C     ALL ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
      REAL LABEL, LINE(1)
C
C     SET THE COLUMNS IN WHICH THE LABEL IS TO BE PRINTED
      I4 = I2 + 4
      I9 = I2 + 9
C
C     CONVERT THE LABEL FROM A WORD TO SIX CHARACTERS
      WRITE(2,100) LABEL
      BACKSPACE 2
      READ(2,110) (LINE(I),I=I4,I9)
      BACKSPACE 2
C
  100 FORMAT(A6)
  110 FORMAT(6A1)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE WORD1(LINE, I2, N)
C
C     ALL ARRAYS ARE DIMENSIONED IN THE CALLING PROGRAM
      REAL TITLE(10,500), LINE(1)
      COMMON TITLE
C
C     SET THE COLUMNS IN WHICH THE TITLE IS TO BE PRINTED
      I4 = I2 + 4
      IX = I2 + 63
C
C     CONVERT FROM WORDS TO CHARACTERS
      WRITE(2,100) (TITLE(I,N),I=1,10)
      BACKSPACE 2
      READ(2,110) (LINE(I),I=I4,IX)
      BACKSPACE 2
C
  100 FORMAT(10A6)
  110 FORMAT(60A1)
      RETURN
      END
```

```
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OUTPUT(MARK, KOUT, N, KOUNT, TSORT, ALPHA1, ALPHA2,
     1  ALPHA3, ALPHA4, LENGTH, FORM)
      REAL LINE(39), ALPHA1, ALPHA2, ALPHA3, ALPHA4
      LOGICAL TSORT, FORM
C
C     SET THE LIMITS FOR THE WIDTH OF OUTPUT
      I1 = 0
      I2 = LENGTH - 1
      I3 = 1
      NL = 12
      NW = 22
      IF(KOUT .EQ. 6) NL = 7
      IF(KOUT .EQ. 6) NW = 13
      I4 = NW
      IX = 13
      IF(FORM) IX = 67
      NP = 1 + (10*(I2) + IX)/(6*NW)
      IF(NP .EQ. 1) GO TO 5
C
C     RESET THE WIDTH LIMITS FOR A PARTIONED OUTPPUT
      I2 = NL
      J = 0
    2 I1 = I1 + J*NL
      I2 = I2 + J*NL
      I3 = I3 + J*NW
      I4 = I4 + J*NW
C     BEGIN THE OUTPUT
    5 REWIND 2
C
C     WRITE THE PROPER PAGE HEADING
   10 IF(TSORT) GO TO 12
      WRITE(KOUT, 980)
      GO TO 16
   12 WRITE(KOUT, 990) ALPHA1, ALPHA2, ALPHA3, ALPHA4
   16 GO TO (20, 30, 40, 50) MARK
   20 WRITE(KOUT,1000) N
      GO TO 60
   30 WRITE(KOUT,1010) N
      GO TO 60
   40 WRITE(KOUT,1020)
      GO TO 50
   50 WRITE(KOUT,1030)
      GO TO 70
C
C     WRITE THE SECOND AND THIRD LINES OF THE PAGE HEADING
   60 WRITE(KOUT,1040)
      GO TO 80
   70 WRITE(KOUT,1050)
   80 WRITE(KOUT,1060) (I,I=I1,I2)
C
C     READ IN A LINE OF OUTPUT FROM THE DISK AND PRINT IT OUT
   90 READ(2,1070) (LINE(I),I=1,I4)
      WRITE(KOUT,1090)
      WRITE(KOUT,1080) (LINE(I),I=I3,I4)
C
C     DECREMENT THE TOTAL NUMBER OF LINES AND CHECK
      KOUNT = KOUNT - 1
      IF(KOUNT.GT.0) GO TO 90
C
```

```
C       CHECK FOR MORE PAGES
        J = J + 1
        NP = NP - 1
        IF(NP) 100, 100, 2
C
    100 RETURN
C
    980 FORMAT(1H1 "UNSORTED")
    990 FORMAT(1H1 "SORTED FIRST BY ", 2A6, "AND THEN BY ", 2A6)
   1000 FORMAT(1H0,"GLOBAL INGREDIENCE OF NODE",I3)
   1010 FORMAT(1H0,"GLOBAL DEPENDENCE OF NODE",I3)
   1020 FORMAT(1H0,"GLOBAL INGREDIENCE OF COMPLETE NETWORK")
   1030 FORMAT(1H0,"GLOBAL DEPENDENCE OF COMPLETE NETWORK")
   1040 FORMAT(1H0,"EXTREME LEVEL FROM OUTPUT")
   1050 FORMAT(1H0,"EXTREME LEVEL FROM INPUT")
   1060 FORMAT(1H0, I4, 10(5X,I2)/)
   1070 FORMAT(39A6)
   1080 FORMAT(1H+,39A6)
   1090 FORMAT(5H      )
        END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
        SUBROUTINE DATOUT(NN, TNI, NODE, INT, LABEL, NI,
      1                      PTIN, IN, ND, PTON, ON, IL, OL, TF)
C
C       ALL OF THE ARRAYS ARE DIMENSIONED IN THE MAIN PROGAM EXCEPT DASH
        INTEGER TNI, NODE(1), INT(1), NI(1), PTIN(1), IN(1), ND(1),
      1         PTON(1), ON(1), IL(1), OL(1), TF(1)
        REAL LABEL(1), TITLE(10,500), DASH(22)
        COMMON TITLE
        DATA DASH(1)/6H------/
        DO 10 J = 2, 22
     10 DASH(J) = DASH(1)
C
C       WRITE THE PAGE HEADING
        I = 0
     20 WRITE(1,1000) DASH
C
C       INCREMENT THE NODE AND WRITE OUT A LINE OF DATA
     30 I = I + 1
        IF(I.GT.NN) RETURN
        N = INT(I)
        WRITE(1,1010) I, LABEL(N), (TITLE(J,N),J=1,10),
      1   IL(N), OL(N), TF(N)
C
C       FIND THE NUMBER OF INGREDIENTS AND DEPENDENTS
        K = NI(N)
        L = ND(N)
C
C       DETERMINE THE NUMBER OF LINES REQUIRED
        KL = (K+3)/4
        LL = (L+3)/4
        IF((KL.GT.1).OR.(LL.GT.1)) GO TO 50
C
C       ONLY ONE LINE REQUIRED
        IF(K.EQ.0) GO TO 35
        K1 = PTIN(N)
        K2 = K1 + K - 1
        WRITE(1,1020) (NODE(IN(J)),J=K1,K2)
     35 IF(L.EQ.0) GO TO 30
        L1 = PTON(N)
```

```
      L2 = L1 + L - 1
      WRITE(1,1030) (NODE(DN(J)),J=L1,L2)
      GO TO 30
C
C     MORE THAN ONE LINE REQUIRED, PROCESS ONE LINE AT A TIME
   50 M = 0
   60 K1 = PTIN(N) + 4*M
      IF(KL-1) 100, 80, 70
   70 K2 = K1 + 3
      GO TO 90
   80 K2 = PTIN(N) + K - 1
C     WRITE THE INGREDIENTS
   90 WRITE(1,1020) (NODE(IN(J)),J=K1,K2)
  100 L1 = PTON(N) + 4*M
      IF(LL-1) 140, 120, 110
  110 L2 = L1 + 3
      GO TO 130
  120 L2 = PTON(N) + L - 1
C     WRITE THE DEPENDENTS
  130 WRITE(1,1030) (NODE(DN(J)),J=L1,L2)
C
C     CHECK TO SEE IF MORE LINES ARE REQUIRED
  140 IF((KL.LE.1).AND.(LL.LE.1)) GO TO 30
C
C     INCREMENT THE COUNTERS
      M = M + 1
      KL = KL - 1
      LL = LL - 1
      WRITE(1,1040)
      GO TO 60
C
 1000 FORMAT(1H1,"DATA    DATA",T115, "INPUT OUTPUT TOTAL",/
     1       " NO.     LABEL", 19X, "DATA DESCRIPTION",
     2       T78, "INGREDIENTS", 9X, "DEPENDENTS", 7X,
     3       "LEVEL  LEVEL FLOAT",/ 22A6)
 1010 FORMAT(1H ,I3, 3X, A6, 3X, 10A6, 41X, 2(I2, 4X), I2)
 1020 FORMAT(1H+, 75X, 4I4)
 1030 FORMAT(1H+, 95X, 4I4)
 1040 FORMAT(1H )
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ERROR(N)
      GO TO (1, 2, 3) N
C
    1 WRITE(6,101)
      STOP
C
    2 WRITE(6,102)
      STOP
C
    3 WRITE(6,103)
      STOP
C
  101 FORMAT(1H ,"PROGRAM STOPPED---INCORRECT INPUT")
  102 FORMAT(1H ,"PROGRAM STOPPED---NO STARTING NODE IN NETWORK")
  103 FORMAT(1H ,"PROGRAM STOPPED---NETWORK CONTAINS A LOOP")
      RETURN
      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 hierarchial 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 alr. dy 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.

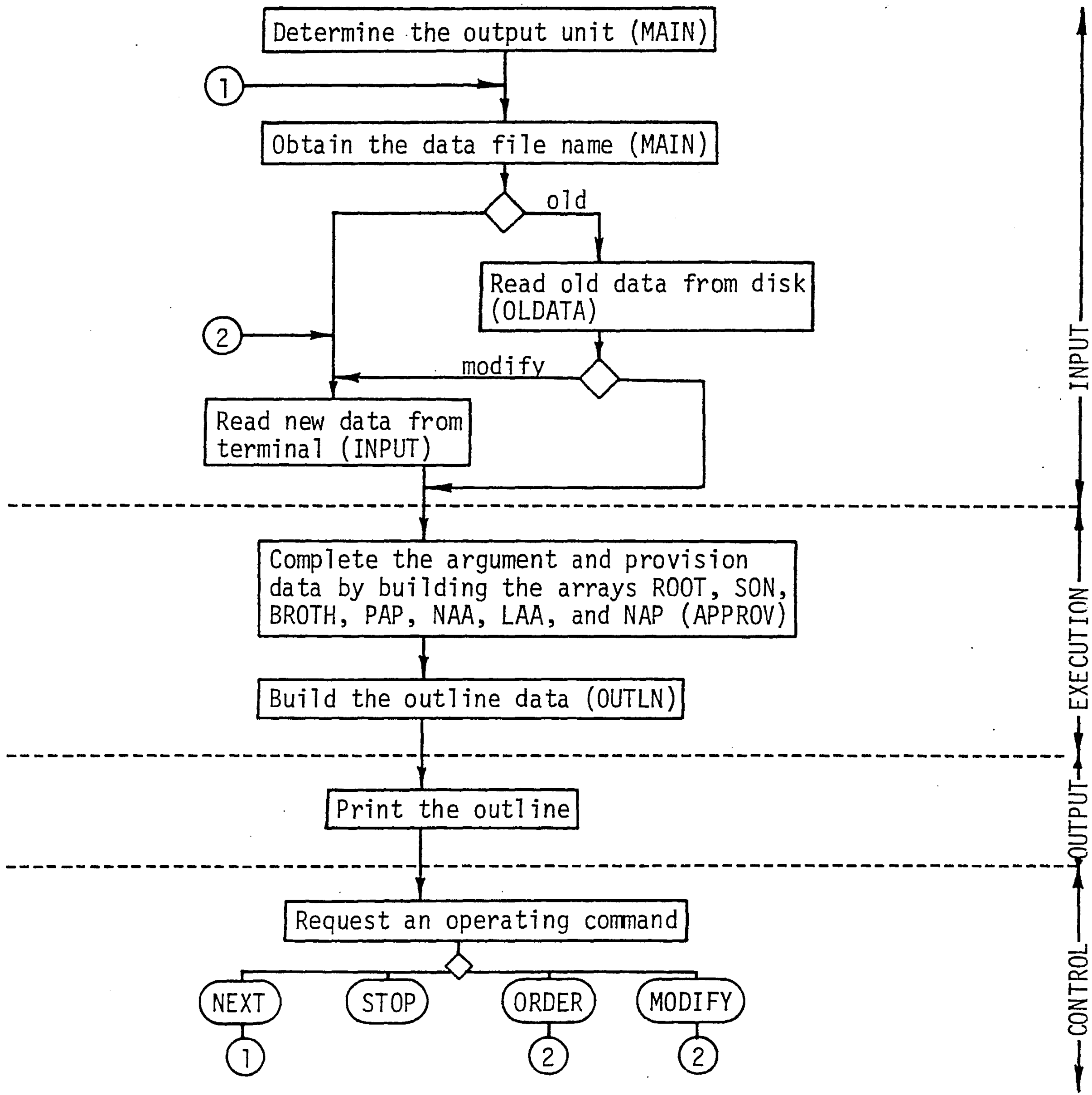
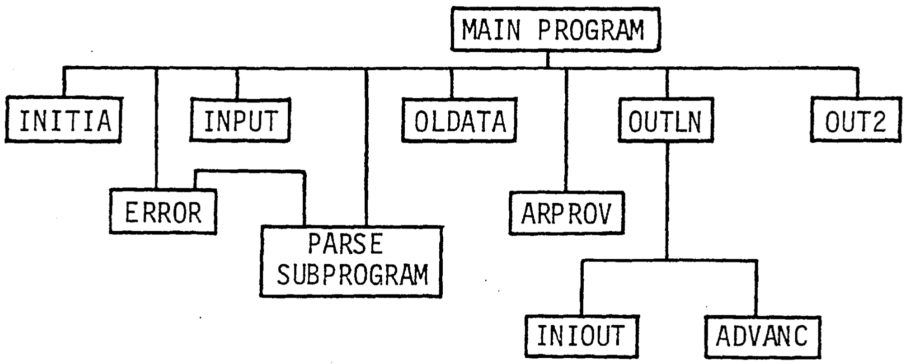


Figure 3.1 Block Diagram for the Outline Program



NOTES:

1. PARSE contains several function and subroutines, including PARSE, WDINIT, SCAN, SETOUT, RDLINE, 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)

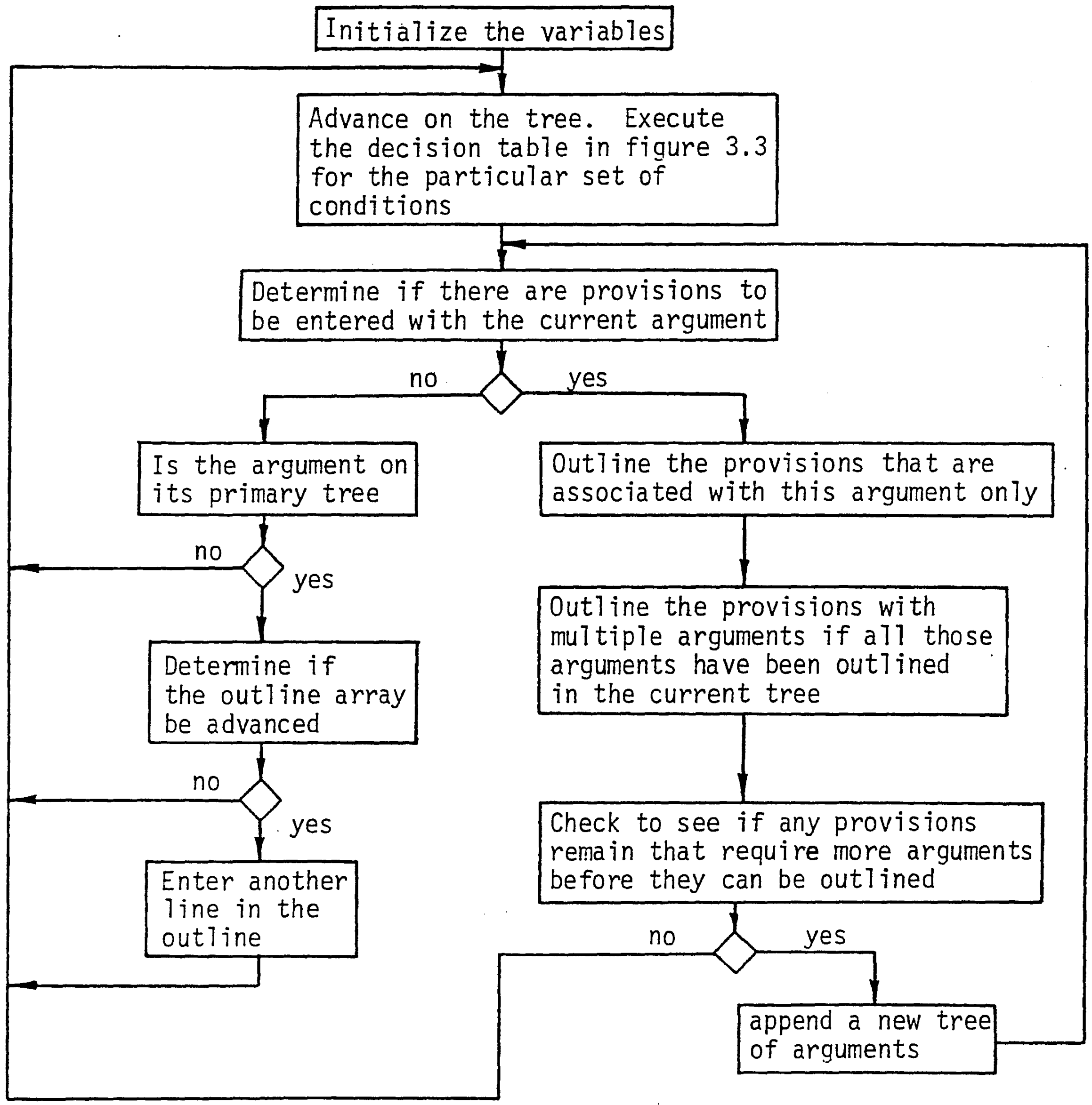


Figure 3.3 Subroutine OUTLN (OUTLINE)

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	F
2	Initial root, LO = 0	*	•	T	F	F	F	F	F	F	F	F	F
3	Initial =	*	•	•	T	T	T	T	F	F	F	F	F
4	Argument has son	*	•	•	T	F	F	F	•	•	•	•	•
5	Argument has brother	*	•	•	•	T	T	F	T	F	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 = LO	*		X	X	X	X		X		X	X	
10	LO = LO+1 (Incr. indent.)	*		X	X		X						
11	LO = LO-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

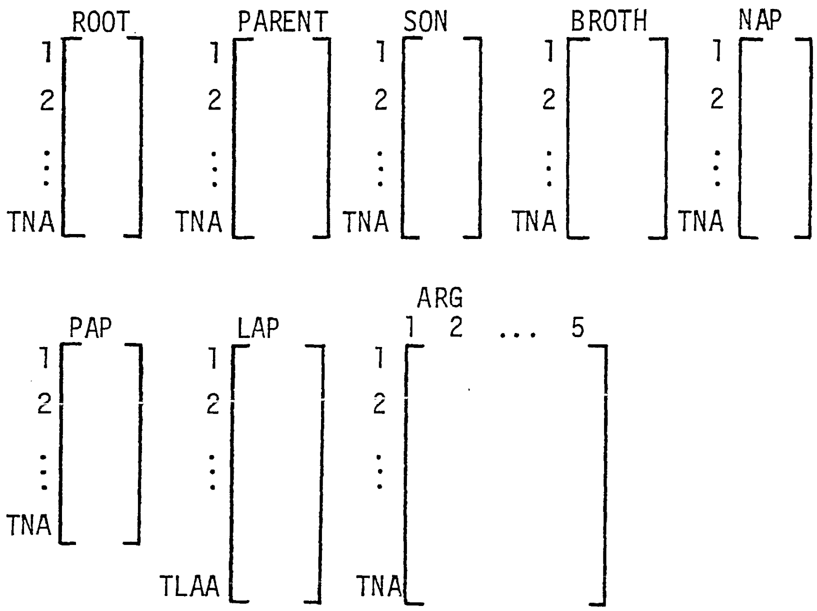
DERIVED DECISION NETWORK

```
C1 + + + R1
 -
 - - - - C2 + + + R2
          -
          - - - - C3 + + + C4 + + + R3
                   -        -
                   -        - - - - C5 + + + C6 + + + R5
                   -                 -        -
                   -                 -        - - - - R4
                   -                 -
                   -                 - - - - R6
                   -
                   - - - - C5 + + + R7
                            -
                            - - - - C7 + + + C8 + + + C9 + + + R10
                                     -        -        -
                                     -        -        - - - - R11
                                     -        -
                                     -        - - - - R9
                                     -
                                     - - - - R8
```

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.

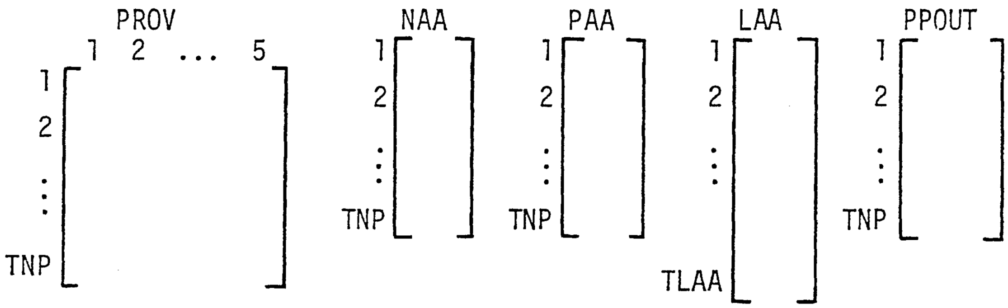


TNA is the total number of arguments
TLAA is the total number of associated arguments

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 alphameric description of the argument

Figure 3.5 Argument Data (OUTLINE)

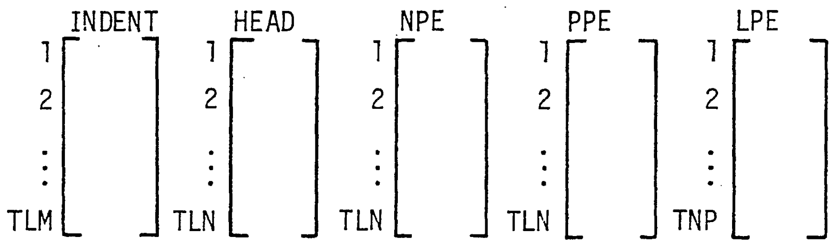


TNP is the total number of provisons
TLAA is the total number of associated arguments

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

PROV - the alphameric 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)

3.4 GLOSSARY

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

SYMBOL	:	DESCRIPTION
A	:	ARGUMENT NUMBER
ADV	:	LOGICAL VARIABLE THAT IS FALSE IF THE ARGUMENT IS NOT ENTERED IN THE OUTLINE
ADVANC	:	A SUBROUTINE
AN	:	ARGUMENT NUMBER
ARPROV	:	A SUBROUTINE
ARG	:	ARRAY CONTAINING THE DESCRIPTIVE TITLE OF THE ARGUMENTS
BL	:	A BLANK SYMBOL (4 CHARACTERS)
BLANK	:	A BLANK SYMBOL (6 CHARACTERS)
BROTH	:	VECTOR CONTAINING THE FIRST BROTHER OF EACH ARGUMENT
DF	:	NUMBER OF THE DATA FILE
DOT	:	THE DOT SYMBOL
ENDIN	:	LOGICAL VARIABLE USED FOR DETECTING THE END OF PROVISION DATA OF ARGUMENT DATA IN THE INPUT ROUTINE
HEAD	:	VECTOR CONTAINING THE ARGUMENT NUMBER FOR EACH HEADING IN THE OUTLINE
INC	:	NUMBER OF ARGUMENTS THAT ARE NOT IN THE STACK
INDENT	:	VECTOR CONTAINING THE INDENTATION OF EACH LINE IN THE OUTLINE
INIOUT	:	A SUBROUTINE
INIT	:	LOGICAL VARIABLE THAT IS FALSE IF SUBROUTINE ADVANC IS BEING PROCESSED A SECOND TIME WITHOUT RETURN
INITIA	:	A SUBROUTINE
INPUT	:	A SUBROUTINE

```
                 :
    LA           :    DUMMY NAME FOR LAA
                 :
    LAA          :    VECTOR CONTAINING THE LIST OF ARGUMENTS ASSOCIATED WITH
                 :    THE PROVISIONS
                 :
    LAP          :    VECTOR CONTAINING THE LIST OF PROVISIONS ASSOCIATED WITH
                 :    THE ARGUMENTS
                 :
    LT           :    INDENTATION OF THE HEADING
                 :
    LIST         :    VECTOR CONTAINING THE LIST OF PROVISIONS ELIGIBLE TO
                 :    BE OUTLINED AT THIS HEADING
                 :
    LN           :    LINE NUMBER OF THE OUTLINE
                 :
    LO           :    INDENTATION LEVEL OF THE NEXT HEADING
                 :
    LPF          :    VECTOR CONTAINING THE LIST OF PROVISION NUMBERS IN THE
                 :    THE ORDER THEY APPEAR IN THE OUTLINE
                 :
    MISS         :    THE NUMBER OF AN ASSOCIATED ARGUMENT THAT IS NOT IN THE
                 :    CURRENT ARGUMENT NETWORK
                 :
    MODIFY       :    LOGICAL VARIABLE THAT IS TRUE WHEN MODIFICATIONS TO THE
                 :    DATA ARE BEING MADE
                 :
    NAA          :    VECTOR CONTAINING THE NUMBER OF ARGUMENTS ASSOCIATED
                 :    WITH EACH PROVISION
                 :
    NAP          :    VECTOR CONTAINING THE NUMBER OF PROVISIONS THAT EACH
                 :    ARGUMENT IS ASSOCIATED WITH
                 :
    NP           :    NUMBER OF PROVISIONS ENTERED AT THE CURRENT HEADING
                 :
    NPF          :    VECTOR CONTAINING THE NUMBER OF PROVISIONS ENTERED AT
                 :    LINE (HEADING) OF THE OUTLINE
                 :
    NPP          :    NUMBER OF PROVISIONS REMAINING TO BE OUTLINED
                 :
    NRU          :    RULE NUMBER IN THE DECISION TABLE FOR SUBROUTINE ADVALC
                 :
    NST          :    NUMBER OF THE STEP IN THE FLOW DIAGRAM FOR SUB. OUTLN
                 :
    NUA          :    VECTOR CONTAINING THE NUMBER OF ASSOCIATED ARGUMENTS
                 :    FOR EACH PROVISION
                 :
    NUFA         :    NUMBER OF ASSOCIATED ARGUMENTS
                 :
    OLDAT        :    LOGICAL VARIABLE THAT IS TRUE FOR OLD DATA
                 :
    OLDATA       :    A SUBROUTINE
                 :
    ORD          :    LOGICAL VARIABLE ASSOCIATED WITH THE COMMAND ORDER
                 :
    ORDER        :    VECTOR CONTAINING THE ROOTS OF THE ARGUMENT TREES IN
                 :    THE ORDER THEY ARE TO BE USED
                 :
    OUTCOM       :    FILE CONTAINING THE COMMON DECLARATIONS FOR THE PROGRAM
                 :
    OUTLN        :    A SUBROUTINE
                 :
```

```
OUT2      :  A SUBROUTINE
          :
P         :  PROVISION NUMBER
          :
PAA       :  VECTOR CONTAINING THE ROW NUMBER OF LAA WHERE THE FIRST
          :  ARGUMENT ASSOCIATED WITH EACH PROVISION IS STORED
          :
PAP       :  VECTOR CONTAINING THE ROW NUMBER OF LAP WHERE THE FIRST
          :  PROVISION EACH ARGUMENT IS ASSOCIATED WITH IS STORED
          :
PARENT    :  VECTOR CONTAINING THE PARENT OF EACH ARGUMENT
          :
PN        :  PROVISION NUMBER
          :
POINT     :  POINTER TO THE PROVISION BEING ENTERED IN THE OUTLINE
          :
PPF       :  VECTOR CONTAINING THE ROW NUMBER IN LPE WHERE THE FIRST
          :  PROVISION FOR EACH LINE (HEADING) IS STORED
          :
PPOUT     :  VECTOR CONTAINING THE LINE NUMBER WHERE EACH PROVISION
          :  IS TO BE ENTERED IN THE OUTLINE
          :
PROV      :  ARRAY CONTAINING THE DESCRIPTIVE TITLE OF THE PROVISIONS
          :
ROOT      :  VECTOR CONTAINING THE ROOT OF THE ARGUMENT TREE FOR
          :  EACH ARGUMENT
          :
SON       :  VECTOR CONTAINING THE SON OF EACH AGUMENT
          :
STACK     :  VECTOR CONTAINING A STACK OF ARGUMENT NUMBERS THAT HAVE
          :  PROVISIONS YET TO BE OUTLINED IN THE CURRENT TREE
          :
SUPP      :  LOGICAL VARIABLE THAT IS TRUE IF THE ARGUMENT IS NOT
          :  ENTERED IN THE OUTLINE
          :
TLAA      :  TOTAL NUMBER OF SSOCIATIONS BETWEEN PROVISIONS AND
          :  ARGUMENTS---THE LENGTH OF LAA AND LAP
          :
TLN       :  TOTAL NUMBER OF LINES IN THE OUTLINE
          :
TNA       :  TOTAL NUMBER OF ARGUMENTS
          :
TNP       :  TOTAL NUMBER OF PROVISIONS
          :
Y         :  SUBSCRIPT OF ORDER
          :
YMAX      :  TOTAL NUMBER OF ROOTS (NUMBER OF TREES OF ARGUMENTS)
          :
Z         :  NUMBER OF THE FILE USED FOR OUTPUT
```

```
  3.5 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  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     *SON(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 PROV(30,5),ARG(30,5)
 C
 C      LOGICAL ENLIN,ORD,INIT,SUPP,ADV,OLDAT,MODIFY,ONPACK
 C
 C      COMMON/INT/DF,PN,TNP,TNA,TLAA,AN,XMAX,LN,NPR,LI,LO,
 C     *TLN,NAA,FAA,LAA,ROOT,PARENT,SON,BROTH,NAP,PAP,
 C     *LAP,ORDER,INDENT,HEAD,NPE,PPE,LPE,PPOUT,STACK,LIST,
 C     *MISS,INC,NP,POINT,X,K,LA,A,P,NUA,Z
 C
 C      COMMON/REAL/PROV,ARG,BLANK,BL,OUT
 C      COMMON/LOG/ENLIN,ORD,INIT,SUPP,ADV,OLDAT,MODIFY,ONPACK
 C
 C      END OF OUTCOM
 $INCLUDE 'PARCOM'
 C
 C
 C      THIS FILE LISTS THE DECLARATIONS FOR USE OF THE INPUT SCANNER
 C
 C      LOGICAL END, FIXED, MATCH, STRING, NEXT
 C      COMMON/SCANER/ 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
        CALL WDINIT(10,72,PRI)
 C
 C      READ THE OUTPUT UNIT IN
        Z = 6
```

```
      WRITE(Z,1005)
      CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
      IF(MATCH("P",1)) Z = 1
      CALL SETOUT(Z)
      NEXT= .TRUE.
C     READ THE DATA FILE NAME IN
      I = 0
   10 WRITE(6,1200)
      CALL SCAN(ENTITY,MODE,VALUE,NCHAR)
      IF(MATCH("YES",3)) CALL WDINIT(10,72,"ON      ")
C
C     INITIALIZE THE ARRAYS
   30 CALL INITIA
   40 WRITE(6,1000)
      CALL RDLINE
      ENTITY(NWD+1)=6H.
      CLOSE (3)
      CHANGE(3,TITLE=ENTITY)
      DF=3
C
C     CHECK FOR EXISTING DATAFILE
      I = 0
      INQUIRE(3,RESIDENT=UNPACK)
      IF(.NOT. UNPACK) GO TO 50
C
C     EXISTING FILE, DOUBLE CHECK THE NAME
      WRITE(6,1010)
      CALL RDLINE
      IF(MATCH("YES",3)) GO TO 63
      GO TO 40
C
C     READ NEW DATA IN FROM THE TERMINAL
   50 CALL INPUT
      GO TO 70
C
C     READ THE OLD DATA FROM THE DATAFILE
   63 CALL OLDATA
C
C     CHECK TO SEE IF MODIFICATIONS TO THE DATA ARE DESIRED
      WRITE(6,1020)
      CALL RDLINE
      IF(.NOT.MATCH("YES",3)) GO TO 100
      MODIFY=.TRUE.
   65 CALL INPUT
   70 CALL ARPROV
      CALL OUTLN
C          PRINT OUT THE OUTLINE
C      CALL OUT1
      CALL OUT2
  100 WRITE(6,1030)
      CALL RDLINE
  110 IF(END(X)) GO TO 100
      IF(MATCH("NEXT",3)) GO TO 10
      IF(MATCH("STOP",3)) STOP
      IF(MATCH("ORDER",3)) GO TO 150
      WRITE(6,1100)
      I = I  + 1
      IF(I.EQ.10) CALL ERROR(1)
      GO TO 100
  150 CALL INITIA
      CALL OLDATA
      ORD=.TRUE.
```

```
      GO TO 65
C
 1000 FORMAT(1H ,"ENTER THE DATA FILE NAME")
 1005 FORMAT(1H , "ENTER P FOR OUTPUT ON THE ON-SITE PRINTER,",
     1 /, " OTHERWISE THE OUTPUT WILL BE ON THE REMOTE TERMINAL")
 1010 FORMAT(1H ,"DATA FILE WITH THIS NAME EXISTS.  ",
     * "DO YOUWANT TO USE IT?",/)
 1020 FORMAT(1H ,"DO YOU WANT TO MODIFY THE EXISTING DATA?")
 1030 FORMAT(1H ,"ENTER A PROGRAM COMMAND")
 1100 FORMAT(" INPUT ERROR --- RE-ENTER ON A NEW LINE")
 1200 FORMAT(" DO YOU WANT THE PRINTOUT OF THE INPUT")
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INITIA
$INCLUDE 'OUTCOM'
      DO 40 I=1,30
      DO 10 J=1,5
      ARG(I,J)=BLANK
   10 PROV(I,J)=BLANK
      NAA(I)=0
      PAA(I)=0
      ROOT(I)=0
      PARENT(I)=0
      SON(I)=0
      BROTH(I)=0
      NAP(I)=0
      PAP(I)=0
      INDENT(I)=0
      HEAD(I)=0
      NPE(I)=0
      PPE(I)=0
      LPE(I)=0
   40 PPOUT(I)=0
      DO 50 I=1,10
      STACK(I)=0
   50 LIST(I)=0
      ORD=.FALSE.
      ENLIN=.FALSE.
      MODIFY=.FALSE.
      TNA=0
      TNP=0
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE INPUT
CDEBUG MONITOR(1) ORD,P,PN,TNP,NWO1,PROV,II,LA,NUA,ENLIN,AN
CDEBUG MONITOR(1) ORDER,XMAX
C
C
C  THIS SUBROUTINE READS THE PROVISIONS WITH THEIR ASSOCIATE
C  D ARGUMENTS, THE LIST OR ARGUMENTS WITH THEIR PARENTS AND
C  BUILDS THE VECTORS LAA,NAA, AND NAP
C
$ INCLUDE 'OUTCOM'
$ INCLUDE 'PARCOM'
       IF (ORD) GO TO 110
      IF(.NOT.MODIFY) GO TO 9
      DO 6 K=1,TNP
```

```
    6 P(K)=K
      DO 8 K=1,TNA
    8 A(K)=K
    9 WRITE(6,1000)
      K=0
C
C BEGIN SCANNING FOR INPUT
C
      CALL PARSE(ENTITY,MODE,VALUE,NCHAR)
   10 IF(MATCH("PROVIS",3)) GO TO 20
   15 IF(MATCH("END",3)) GO TO 160
      IF(MATCH("ARGUME",3)) GO TO 60
      IF(MATCH("ORDER",3)) GO TO 110
C
C  IMPROPER INPUT DATA
C
      WRITE(6,1100)
      CALL RDLINE
      K=K+1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 10
C
C  HERE WE HAVE THE LIST OF PROVISIONS AND ASSOCIATED ARG.
C
   20 WRITE(6,1060)
C             PN= COUNTER OF PROVISIONS
      PN=0
      K=0
      CALL RDLINE
   26 IF(.NOT.FIXED(X)) GO TO 28
      PN=PN+1
      IF(MODIFY) PN=IVALUE
      P(PN)=IVALUE
      TNP=MAX0(TNP,P(PN))
      IF(STRING(X)) GO TO 30
   27 IF(FIXED(X)) GO TO 40
   28 IF(END(X)) GO TO 50
       IF(ENLIN) GOTO 29
C  IMPROPER INPUT DATA
      WRITE(6,1100)
      CALL RDLINE
      K=K+1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 26
   29 CONTINUE
C  NEW LINE
      GO TO 15
C
C  READ TITLE OF PROVISION
C
C             II: COUNTER OF ASSOCIATED ARGUMENTS OF PROVISION
   30 II=0
C
      NWD1=(NCHAR+5)/6
      NWD1=MIN0(NWD1,5)
      IF(.NOT.MODIFY) GO TO 32
      DO 31 J=1,5
   31 PROV(P(PN),J)=BLANK
   32 DO 35 J=1,NWD1
   35 PROV(P(PN),J)=ENTITY(J)
      GO TO 27
   40 II=II+1
```

```
      LA(P(PN),II)=IVALUE
      NUA(P(PN))=II
      ENLIN=.FALSE.
      GO TO 27
   50 ENLIN=.TRUE.
      GO TO 26
C
C
C
C            AN: COUNTER OF ARGUMENTS
   60 WRITE(6,1070)
      AN=0
      K=0
      CALL RDLINE
      ENLIN=.FALSE.
   65 IF(.NOT.FIXED(X)) GO TO 90
      AN=AN+1
      A(AN)=IVALUE
      IF(MODIFY) AN=IVALUE
      IF(.NOT.STRING(X)) GO TO 80
      NWD1=(NCHAR+5)/6
      NWD1=MIN0(5,NWD1)
C
C
C
C
      IF(.NOT.MODIFY) GO TO 68
      DO 67 J=1,5
   67 ARG(A(AN),J)=BLANK
   68 DO 70 J=1,NWD1
   70 ARG(A(AN),J)=ENTITY(J)
   80 IF(.NOT.FIXED(X)) GO TO 90
      PARENT(A(AN))=IVALUE
      ENLIN=.FALSE.
   90 IF(END(X)) GO TO 95
C
C   ENLIN.-    WHEN THE ENTITY IS NOT STRING,INTEGER, OR END
C              OF LINE, ENLIN IS USED TO SPECIFY THAT THIS
C              IS A NEW LINE IF ITS VALUE IS TRUE.   BUT, IF
C              ITS VALUE IS FALSE, THERE IS AN ERROR.
C
      IF(ENLIN) GO TO 100
      WRITE(6,1100)
      CALL RDLINE
      K=K+1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 60
C  NEW LINE
  100 TNA=AN
      GO TO 10
   95 ENLIN=.TRUE.
      GO TO 25
  110 I=0
      WRITE(6,1080)
      K=0
  120 IF(.NOT.FIXED(X)) GO TO 130
      I=I+1
      ORDER(I)=IVALUE
      ENLIN=.FALSE.
       GO TO 120
  130 IF(.NOT.END(X)) GO TO 140
      ENLIN=.TRUE.
```

```
C
      GO TO 120
  140 IF(ENLIN) GO TO 150
      WRITE(6,1100)
      CALL RDLINE
      K=K+1
      IF(K.EQ.10) CALL ERROR(1)
      GO TO 10
  150 XMAX=I
      GO TO 10
  160 IF(.NOT.ORD) GO TO 165
      ORD=.FALSE.
      RETURN
  165 REWIND DF
      WRITE(DF=1,1040) TNP,TNA
      DO 170 I=1,TNP
      NI=P(I)
      NUMA=NUA(NI)
      NPI=NI+1
  170 WRITE(DF=NPI,1050)NI,(PROV(NI,J),J=1,5),NUMA,(LA(NI,II),II=1,NUMA
      DO 180 I=1,TNA
      NI=A(I)
      NAI=A(I)+31
  180 WRITE(DF=NAI,1050)NI,(ARG(NI,J),J=1,5),PARENT(NI)
      END FILE DF
       LOCK DF
 1000 FORMAT(" BEGIN INPUT INSTRUCTIONS.",
     *"  ENTER THE WORD END WHEN FINISHED.")
 1100 FORMAT(" INPUT ERROR --- RE-ENTER ON A NEW LINE")
 1040 FORMAT(1H ,2I2)
 1050 FORMAT(1H ,I2,5A6,6I3)
 1060 FORMAT(" ENTER THE LIST OF PROVISIONS AND ASSOCIATED ARGUMENTS",
     */," ONE LINE FOR EACH PROVISION")
 1070 FORMAT(" ENTER THE LIST OF ARGUMENTS AND THEIR PARENTS",
     */," ONE LINE FOR EACH ARGUMENT")
 1080 FORMAT(" ENTER THE WORD END WHEN FINISHED")
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OLDATA
$ INCLUDE 'OUTCOM'
      REWIND DF
      READ(DF=1,1040) TNP,TNA
      DO 170 I=1,TNP
      N=I+1
      READ(DF=N,1050)(PROV(I,J),J=1,5),NUMA,(LA(I,II),II=1,NUMA)
  170 NUA(I)=NUMA
      DO 180 I=1,TNA
      NAI=I+31
  180 READ(DF=NAI,1050)(ARG(I,J),J=1,5),PARENT(I)
      REWIND DF
      OLDAT=.TRUE.
 1040 FORMAT(1H ,2I2)
 1050 FORMAT(1H ,2X,5A6,6I3)
      RETURN
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE ARPROV
```

```
CEBUG MONITOR(1) ROOT,SON,BROTH,PAP,LAP,PAA,NUMA,NAA,LAA,NAP
CEBUG MONITOR(1) PARENT
$ INCLUDE 'OUTCOM'
      INTEGER RROOT,PPAP
C
C  THIS SUBROUTINE COMPLETES THE ARGUMENT AND PROVISION
C  DATA BY BUILDING THE ARRAYS ROOT,SON,BROTH,PAP,NAA,LAA,NAP
C
      I=0
      DO 9 K=1,TNP
      NUMA=NUA(K)
      NAA(K)=NUMA
      DO 5 J=1,NUMA
      I=I+1
      LAA(I)=LA(K,J)
    5 NAP(LAA(I))=NAP(LAA(I))+1
    9 CONTINUE
      TLAA=I
      PPAP=NAP(1)+1
      DO 60 I=1,TNA
      PARENT(I)=PARENT(I)
      IF(PARENT(I).EQ.0) GO TO 20
      IF(I.EQ.TNA) GO TO 10
      IF(PARENT(I).EQ.PARENT(I+1)) GO TO 10
      ROOT(I)=RROOT
      IF(PARENT(I+1).NE.0) GO TO 30
      SON(I)=0
      GO TO 35
      GO TO 30
   10 ROOT(I)=RROOT
      BROTH(I)=I+1
      GO TO 40
   20 RROOT=PARENT(I+1)
      ROOT(I)=I
      BROTH(I-1)=0
   30 SON(I)=I+1
   35 BROTH(I)=0
   40 IF(NAP(I).NE.0) GO TO 50
      PAP(I)=0
      GO TO 60
   50 PAP(I)=NAP(I)+PPAP
      PPAP=PAP(I)
   60 CONTINUE
      BROTH(TNA)=0
C           BUILD PAA
      PAA(1)=1
      DO 70 I=2,TNP
   70 PAA(I)=PAA(I-1)+NAA(I-1)
C           BUILD LAP
      II=TNP
      DO 90 M=1,TNP
      I=PAA(II)
      J=I+NAA(II)-1
      DO 80 K=I,J
      AN=LAA(K)
      PAP(AN)=PAP(AN)-1
      L=PAP(AN)
   80 LAP(L)=II
      II=II-1
   90 CONTINUE
      RETURN
      END
```

```
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OUTLN
CERUG MONITOR(1) STACK,AN,ADV,INIT,SUPP,LI,LO,NP,LIST,NPR,INC
CERUG MONITOR(1) MISS,NPE,POINT,PPE,LN,NAP,TLN,ROOT,SON,BROTH,PAP
CERUG MONITOR(1) LAP,PPOUT,K,X,N1,N2,INDENT,HEAD,LPE,ORDER
$ INCLUDE 'OUTCOM'
C
C  1. INITALIZE OUTLINING VARIABLES
C
C
      CALL INIOUT
C
C  2. ADVANCE ON THE TREE (SEE TABLE 1. DECISION TABLE FOR
C     ADVANCE)
C
   20 NST=2
C     WRITE(1,300) NST
      CALL ADVANC
      IF(NPR.EQ.0) RETURN
  300 FORMAT(" ",T95,"STEP=",I2)
C
C  3. DETERMINE IF THERE ARE PROVISIONS TO BE ENTERED UNDER
C     THE HEADING OF THE CURRENT ARGUMENT
C
   30 NST=3
C     WRITE(1,300) NST
      IF(NAP(AN).EQ.0) GO TO 100
      NP=0
      N1=PAP(AN)
      N2=N1+NAP(AN)-1
      DO 31 J=N1,N2
      PPOUT(LAP(J))=PPOUT(LAP(J))
      IF(PPOUT(LAP(J)).NE.0) GO TO 31
      ROOT(LAA(PAA(LAP(J))))=ROOT(LAA(PAA(LAP(J))))
      IF (ROOT(LAA(PAA(LAP(J)))).NE.STACK(1)) GO TO 31
      NP=NP+1
      LIST(NP)=LAP(J)
   31 CONTINUE
C
C  THE PROVISION(S) IS(ARE) ENTERED WHEN THE ARGUMENT FIRST
C  ON ITS LIST OF ARGUMENTS IS IN THE HIERARCHICALLY FIRST
C  ARGUMENT TREE OF THE STACK
C
      NP=NP
      IF (NP.EQ.0) GO TO 100
C
C  4. ADVANCE OUTLINE
C
      NST=4
C     WRITE(1,300) NST
      ADV=ADV
      IF(.NOT.ADV) GO TO 41
      LN=LN+1
      INDENT(LN)=LO
      HEAD(LN)=AN
      SUPP=.FALSE.
      NPE(LN)=0
      PPE(LN)=POINT
   41 INC=0
C
```

```
C   5. ENTERING PROVISIONS
C          CHECK IF PROVISIONS CAN BE ENTERED.  IF THEY ARE
C          COMPLETE (ALL THEIR ARGUMENTS APPEAR IN THE CURRENT
C          STACK OF ARGUMENTS) THEY MAY BE ENTERED
C
       NST=5
C      WRITE(1,300) NST
    50 DO 59 J=1,NP
       N1=PAA(LIST(J))
       N2=N1-1+ NAA(LIST(J))
C
C   6. COMPLETE?
C
       NST=6
C      WRITE(1,300) NST
       DO 58 JJ=N1,N2
       DO 57 KK=1,K
       STACK(KK)=STACK(KK)
       IF(STACK(KK).EQ.LAA(JJ)) GO TO 58
    57 CONTINUE
       MISS=LAA(JJ)
       INC=INC+1
       GO TO 59
    58 CONTINUE
C
C   ALL ITS ARGUMENTS ARE IN THE CURRENT STACK, THEREFORE
C   PROVISION IS COMPLETE
C
C   7. ENTER PROVISION
       NST=7
C      WRITE(1,300) NST
       NPE(LN)=NPE(LN)+1
       LPE(POINT)=LIST(J)
       POINT=POINT+1
       PPOUT(LIST(J))=LN
       NPR=NPR-1
    59 CONTINUE
C
C   8. ANY INCOMPLETE PROVISION.   A NEW TREE OF ARGUMENTS IS
C      APPENDED AT THE ROOT AS THE ELDEST SON OF THE CURRENT
C      ARGUMENT
C
       NST=8
C      WRITE(1,300) NST
       IF(INC.EQ.0) GO TO 20
C
C   9. APPEND TREE
C
       NST=9
C      WRITE(1,300) NST
       K=K+1
       STACK(K)=ROOT(MISS)
       IF(STACK(K).NE.0) GO TO 95
       STACK(K)=ROOT(MISS+1)
    95 ADV=.TRUE.
       LI=LO
       LO=LO+1
       INIT=.TRUE.
       AN=STACK(K)
C
       GO TO 30
C
```

```
          NPR=NPR
          IF(NPR.EQ.0) RETURN
C
C  10.PRIMARY TREE.    NO PROVISIONS FOR POTENCIAL ENTRIES ARE
C         ENCOUNTERED AT A HEADING
C
  100 IF(ROOT(AN).EQ.STACK(1)) GO TO 105
           NST=10
C         WRITE(1,300) NST
          SUPP=.TRUE.
          LU=LI
          GO TO 20
C
C
  105 SUPP=.FALSE.
          ADV=ADV
          NST=10
C         WRITE(1,300) NST
          IF(.NOT.ADV) GO TO 20
          LN=LN+1
          INDENT(LN)=LO
          HEAD(LN)=AN
          GO TO 20
C
          END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
           SUBROUTINE INIOUT
$INCLUDE 'OUTCOM'
C  THIS SUBROUTINE INITIALIZES OUTILING VARIABLES
C
           LN=0
          LI=0
          LO=0
          K=1
          X=1
          STACK(K)=ORDER(X)
          SUPP=.FALSE.
          NPR=TNP
          POINT=1
          AN=STACK(K)
          PPE(I)=1
          RETURN
          END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
           SUBROUTINE ADVANC
CEBUG MONITOR(1) STACK,AN,ADV,INIT,SUPP,LI,LO,NP,LIST,NPR,INC
CEBUG MONITOR(1) MISS,NPE,POINT,PPE,LN,TLN,NAP,ROOT,SON,BROTH,PAP
CEBUG MONITOR(1) LAP,PPOUT,K,X
$ INCLUDE 'OUTCOM'
C  THIS SUBROUTINE DOES THE ADVANCE ON THE TREE (SEE TABLE 1
C  DECISION TABLE FOR ADVANCE)
C
C  R U L E  1.  NPR=0  MEANS: OUTLINING FINISHED
          NPR=NPR
          IF(NPR.NE.0) GO TO 2
          NRU=1
C         WRITE(1,200) NRU
  200 FORMAT("0 ",T70,"RULE ",I2)
```

```
      TLN=LN
      RETURN
C   R U L E  2.  LO=0   INITIAL PRIMARY TREE
C
    2 IF(LO.NE.0) GO TO 3
      NRU=2
C     WRITE(1,200) NRU
      IF(LO.EQ.0) GO TO 110
    3 IF(.NOT.INIT) GO TO 7
      IF(SON(AN).EQ.0) GO TO 5
C
C   R U L E  3.  SON IS NEW ARGUMENT
C
      NRU=3
C     WRITE(1,200) NRU
      K=K+1
      STACK(K)=SON(AN)
      AN=STACK(K)
      GO TO 110
    5 IF(BROTH(AN).EQ.0) GO TO 66
      K=K+1
      STACK(K)=BROTH(AN)
      AN=STACK(K)
      NRU=4
      IF(SUPP) NRU=5
C     WRITE(1,200) NRU
      IF(SUPP) GO TO 110
C
C   R U L E  4.  SUPP=.FALSE.  BROTHER IS NEW ARGUMENT
C   R U L E  5.  SUPP=.TRUE.   BROTHER IS NEW ARGUMENT
C
      GO TO 105
C   R U L E S  6 AND 8    REENTER WITH PARENT
C
   66 NRU=6
C     WRITE(1,200) NRU
   67 K=K
      STACK(K)=STACK(K)
      PARENT(AN)=PARENT(AN)
      IF(STACK(K).EQ.PARENT(AN)) GO TO 69
      K=K-1
      GO TO 67
   69 INIT=.FALSE.
      LO=LO-1
      AN=STACK(K)
      GO TO 3
    7 IF(BROTH(AN).EQ.0) GO TO 70
C   R U L E  7.  BROTHER IS NEW ARGUMENT
C
      NRU=7
C     WRITE(1,200) NRU
      K=K+1
      STACK(K)=BROTH(AN)
      AN=STACK(K)
      GO TO 105
      IF(PARENT(AN).NE.0) NRU=8
C     WRITE(1,200) NRU
   70 IF(PARENT(AN).NE.0) GO TO 68
C
C   R U L E  8.   PARENT .NE. 0  REENTER WITH PARENT
C
      IF(K.EQ.1) GO TO 100
```

```
      C
      C  R U L E  9. PROCEED TO APEND ANOTHER TREE
            NRU=9
      C     WRITE(1,200) NRU
            K=K-1
            AN=STACK(K)
            ADV=.FALSE.
            LI=LO
            GO TO 120
        100 X=X
            XMAX=XMAX
            IF(X.LT.XMAX) GO TO 102
              CALL ERROR(2)
      C
      C  R U L E  10.  NEW PRIMARY TREE
        102 X=X+1
            NRU=10
      C     WRITE(1,200) NRU
            STACK(1)=ORDER(X)
            AN=STACK(1)
        105 LI=LO
            ADV=.TRUE.
            GO TO 120
        110 LI=LO
            LO=LO+1
            ADV=.TRUE.
        120 INIT=.TRUE.
            RETURN
            END
      C
      C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            SUBROUTINE OUT1
      $ INCLUDE 'OUTCOM'
            WRITE(1,100)
            DO 50 I=1,TLN
         50 WRITE(1,200) I,INDENT(I),HEAD(I),NPE(I),PPE(I),LPE(I)
      C
      C
        100 FORMAT("1",T5,"SUBSCRIPT",T20,"LO  P.ARG",T35,"N.ENTRIES",
           *T45,"P.PROV",T55,"PROV.")
        200 FORMAT(" ",T7,I3,T20,I2,5X,I2,T37,I3,T47,I3,T57,I3)
            RETURN
            END
      C
      C - - - - - - - - - - - - - - - - - - - - - - - - - - -
      C
            SUBROUTINE ERROR(N)
            GO TO (1,2),N
          1 WRITE(6,100)
            WRITE(1,100)
            STOP
          2 WRITE(6,200)
            WRITE(1,200)
            STOP
        100 FORMAT(" INCORRECT INPUT--- PROGRAM EXECUTION STOPPED.")
        200 FORMAT(" PROGRAM STOPPED--- SOME PROVISIONS HAVE NOT
           *BEEN OUTLINED")
            RETURN
            END
      C
      C - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      C
```

```
      BLOCK DATA
$ INCLUDE 'OUTCOM'
      DATA BLANK/6H      /,BL/4H    /,DOT/4H.   /
      END
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      SUBROUTINE OUT2
$INCLUDE "OUTCOM"
      WRITE(Z,1050)
      WRITE (1,1000)
      WRITE(Z,1060)
      DO 60 I=1,TLN
      NBL=INDENT(I)
      WRITE(Z,2000)(BL,K=1,NBL)
      GO TO (10,20,30,40,50),NBL
   10 WRITE(Z,100) (ARG(HEAD(I),J),J=1,5)
      GO TO 55
   20 WRITE(Z,200) (ARG(HEAD(I),J),J=1,5)
      GO TO 55
   30 WRITE(Z,300) (ARG(HEAD(I),J),J=1,5)
      GO TO 55
   40 WRITE(Z,400) (ARG(HEAD(I),J),J=1,5)
      GO TO 55
   50 WRITE(Z,500) (ARG(HEAD(I),J),J=1,5)
C
   55 J=PPE(I)
      IF(J.EQ.0) GO TO 60
      K=J+NPE(I)-1
      DO 56 L=J,K
      NBL1=NBL+1
      WRITE(Z,2000) (BL,N=1,NBL),(DOT,M=NBL1,10)
   56 WRITE(Z,4000) LPE(L),(PROV(LPE(L),M),M=1,5)
   60 CONTINUE
      WRITE(Z,4500)
 1050 FORMAT("1",T6,"O   U   T   L   I   N   E")
 1000 FORMAT("0",T6, "H E A D I N G",T40,"P R O V I S I O N")
 1060 FORMAT(" ",T6,13(1H-),T40,17(1H-),///)
 2000 FORMAT(" ",30A4)
 4000 FORMAT("+",T40,I2,2X,5A6)
 4500 FORMAT("1")
  100 FORMAT("+",T6 ,5A6)
  200 FORMAT("+",T10 ,5A6)
  300 FORMAT("+",T14,5A6)
  400 FORMAT("+",T18 ,5A6)
  500 FORMAT("+",T22 ,5A6)
C
C
      RETURN
      END
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.