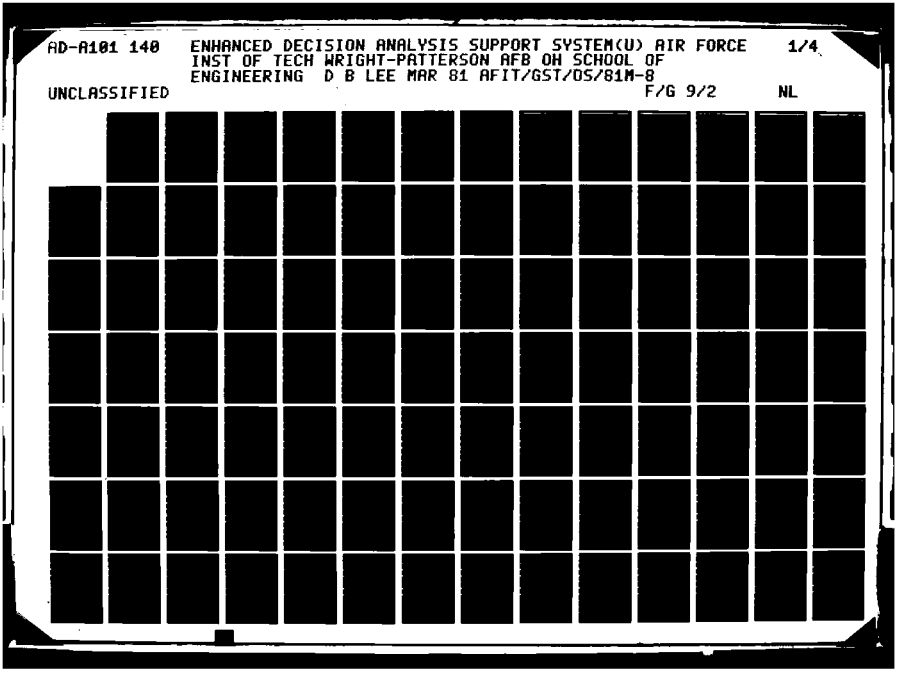
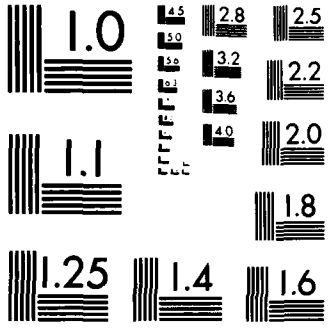

AD-A101 140 ENHANCED DECISION ANALYSIS SUPPORT SYSTEM(U) AIR FORCE 1/4
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING D B LEE MAR 81 AFIT/GST/OS/81M-8
UNCLASSIFIED F/G 9/2 NL



LEVEL II

AFIT/GST/OS/81M-8

28 MAY 19[illegible]

APPROVED FOR PUBLIC RELEASE AFR 1[illegible]

Fredric C. Lynch

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs

Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

AD A101140

DTIC FILE COPY

(6) ENHANCED DECISION ANALYSIS SUPPORT SYSTEM,

(11) May 81

(9) Master's THESIS,

(10) David B. Lee
Captain USAF

(12) 173

(14) AFIT/GST/OS/81M-8

DTIC
ELECTE
JUL 9 1981
S
D

Approved for public release, distribution unlimited

012225

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GST/OS/81M-8	2. GOVT ACCESSION NO. AD-A101140	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ENHANCED DECISION ANALYSIS SUPPORT SYSTEM		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David B. Lee Captain, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE March 1981 13. NUMBER OF PAGES 306
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

2 JUN 1981

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE AFR 190-17.

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer programs
Decision aids
Decision analysis
Decision making
Decisions
Hierarchies
Multiattribute utility

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The enhanced version of the Decision Analysis Support System (DASS) is a highly interactive computer-aided decision analysis tool. It automates the method for determining preferences when multiple and competing attributes are involved. Worth assessment is used as the model which evaluates a deterministic hierarchical tree structure, although risk can be evaluated by incorporating risk into the tree or by conducting sensitivity analyses.

The objective of this work was to incorporate the DASS (originally

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

developed by Bruce W. Morlan) onto a microcomputer using PASCAL. The second objective was to exploit the color graphics capability on the microcomputer system for both input and output displays. The final objective was to add capability to the DASS by adding additional sensitivity analysis modules.

All objectives were met in this work. Additional exploitation of the computer graphics, as well as some pre-processing of the tree structure are areas suggested for further exploration.

AFIT/GST/OS/81M-8

Thesis

ENHANCED DECISION

ANALYSIS SUPPORT SYSTEM

by

David B. Lee
Captain USAF

Prepared in partial
fulfillment of the
requirements for a
Masters Degree

March 1981

School of Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base
Ohio

Preface

I wish to thank my advisor, Major Daniel Fox, who provided me with professional guidance and the freedom which allowed me to complete this thesis. I also thank Captain Aaron DeWispelare, my reader, who gave good advice and council.

My classmates, whose sense of humor and professionalism made my stay here bearable, also deserve my thanks.

Abstract

The enhanced version of the Decision Analysis Support System (DASS) is a highly interactive computer-aided decision analysis tool. It automates the method for determining preferences when multiple and competing attributes are involved. Worth assessment is used as the model which evaluates a deterministic hierarchical tree structure, although risk can be evaluated by incorporating risk into the tree or by conducting sensitivity analyses.

The objective of this work was to incorporate the DASS (originally developed by Bruce W. Morlan) onto a microcomputer using PASCAL. The second objective was to exploit the color graphics capability on the microcomputer system for both input and output displays. The final objective was to add capability to the DASS by adding additional sensitivity analysis modules.

All objectives were met in this work. Additional exploitation of the computer graphics, as well as some pre-processing of the tree structure are areas suggested for further exploration.

Table of Contents

List of Figures

ENHANCED DECISION ANALYSIS SUPPORT SYSTEM

I. Background

We, as human beings, are faced with many decisions from the time we awake in the morning until we sleep at night. Many decisions are trivial in nature, such as which shirt to wear, or which route to take to work. Others are more complex; such as which next job to take or what type of house to buy.

Extrapolating from the personal to the corporate, decisions have to be made by individuals concerning a military service or a national government. Decisions such as whether or not to buy the MX missile system or whether or not to develop nuclear power are complex both in the number of and the variability of the factors in the decision. Such decisions can be made entirely on an ad hoc, seat of the pants basis, although decisions so based could be radically incorrect. Thus decision makers are seeking methods and techniques which enables them to reduce the uncertainty in their decisions as well as providing a comfortable structure to work in. One technique is decision analysis.

Decision Analysis

Decision analysis has been available as a tool for decision makers for about 10 years (Ref 8:4). Decision analysis is defined as:

> "... a quantitative method which permits the systematic evaluation of the costs or benefits accruing to courses of actions that might be taken in a decision problem. It entails identification of the alternative choices involved, the assignment of values (costs/benefits) of possible outcomes, and the expression of the probability of those outcomes being realised."

Decision analysis looks at a particular way a problem can be decomposed into identifiable elements, and systematically evaluates the elements in order for the decision maker to clearly understand the problem and to actively pursue a solution (Ref 10:vii).

In using decision analysis as a tool for decision making, a particular paradigm is developed for the problem. Credit for the paradigm, itself, is generally attributed to A. D. Hall of the Bell Telephone System, and consists of seven steps: problem definition, value system design, system synthesis, system analysis, optimization, decision making and planning for action.

In the problem definition, a general statement of the current situation is defined as is the future situation after decision implementation. In addition, the scope of the problem is examined with regard to the stakeholders, their specific needs, major constraints and

societal factors. The problem is then partitioned into relevant elements. These elements must be relevant to the stakeholders, that is the decision maker as well as those who fund the project. When the problem is thus defined, the value system design is initiated.

In the values system design, the objectives and measures for the objectives are developed. These objectives are organized into a

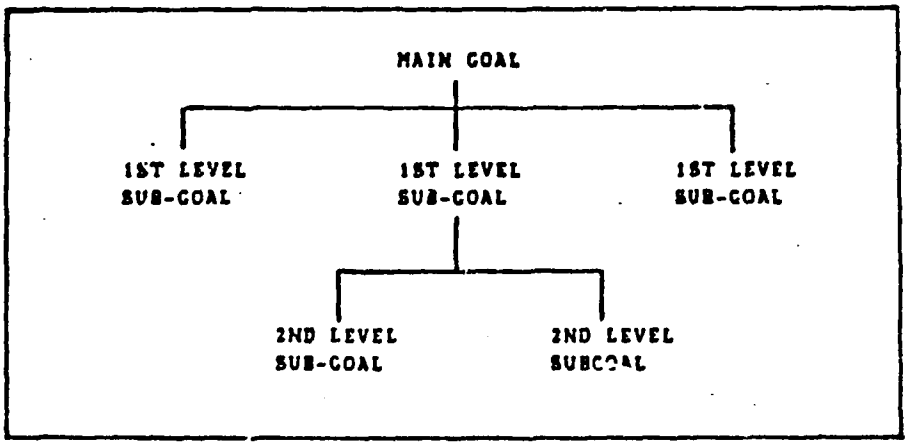


Figure 1 Objective Tree

hierarchical structure or objective tree with the main goal atop and supporting goals as various branches (Figure 1).

Going down the objective tree answers the question of how the main goal is to be accomplished. Going up the objective tree answers the question of why a particular goal needs to be met. Going laterally through the tree for the same level sub-goal determines what needs to be

accomplished to fulfill the next higher goal.

The remaining steps use various decision analysis techniques to arrive at a decision and then to implement the decision. One technique that is used for multiple objectives, which may not easily relate to specific criteria and may also be conflicting, is Multiple Criteria Decision Theory (MCDT). Multiple criteria decison theory is further divided into a Multiple Attribute Utility Theory (MAUT) and Multiple Objective Optimisation Theory (MOOT). Selection of MAUT or MOOT depends primarily on the nature of the problem, policies, and attributes.

Multiple Attribute Utility Theory is defined as:

> "A type of decision theory; requires the analyst to elicit preference information concerning the attributes of proposed alternative policy of the decision maker; utilizing the decision makers' preferences, the analyst forms a scalar choice function (SCF). The SCF is used to evaluate the outcomes of the alternatives, score, and subsequently rank the alternative policies for the decision making step." (Ref 3)

The advantage of MAUT is that the result provides a complete ranking of the alternatives; however, disadvantages include subjectivity in establishing the scalar choice function and the time required for implimentation.

Multiple Objective Optimisation Theory is defined as:

> "An optimisation method for enumerating 'optimal'

solutions for alternative acts which extremise a vector of performance indices. The purpose is to generate a non-dominated solution set we call pareto optimal which represents 'efficient' allocation of resources." (Ref 3)

The advantage of MOOT is that the only scoring functions with respect to performance indices are necessary. This results in some time saving over MAUT. However, MOOT only provides a partial or incomplete ranking of alternatives.

Focusing our attention to MAUT, we find that MAUT is further divided into two areas: certainty and risk. Under certainty, weights and values of the attributes are determined exactly. This form of the problem represents an easily solvable, closed form solution using the various decision tree techniques such as worth assessment.

Risk involves getting the decision maker's attitudes towards risk and establishing the values of attributes and alternatives; however, in this case, each value has an associated uncertainty factor. Solutions to this problem are much more complex, although they come much closer to repesenting the real world. Solutions often incorporate utility functions to measure the risk adverseness or proneness of a given decision maker to determine alternative ranking.

Current Activities in Decision Analysis

Decision analysis is slowly being accepted by the industrial and governmental community. Decision Support Systems (DSS) defined as:

> "...a computer-based system (say, a data base management system or a set of financial models) which is used personally on an ongoing basis by managers and their immediate staffs in direct support of managerial activities -- that is, decisions." (Ref 9:117)

have been growing both in the academic as well as industrial sectors. DSS are primarily designed to aid in decision making and decision implementation. Further, DSS focuses on the support of decision making and decision analysis rather than on the system of information flow and reports (Ref 18:61-63). Emphasis is placed on integrating the decision maker into the process of decision analysis. A model of the DSS system is shown in Figure 2.

Using the computer as a bookkeeper and display generator, the decision maker can determine the results of a particular decision or decisions. The objective of DSS is to provide the decision maker a very complete and flexible model which can be accessed from a terminal with out knowledge of programming or computers (Ref 11:38).

In order to meet the DSS objective many techniques using decision analysis methods are incorporated into the model base. As stated earlier, decision analysis methods break down a large problem into identifiable, smaller, more managable elements which can be dealt with on an individual basis. This decomposability coupled with the structured nature of decision analysis makes computerization of decision analysis

techniques highly attractive.

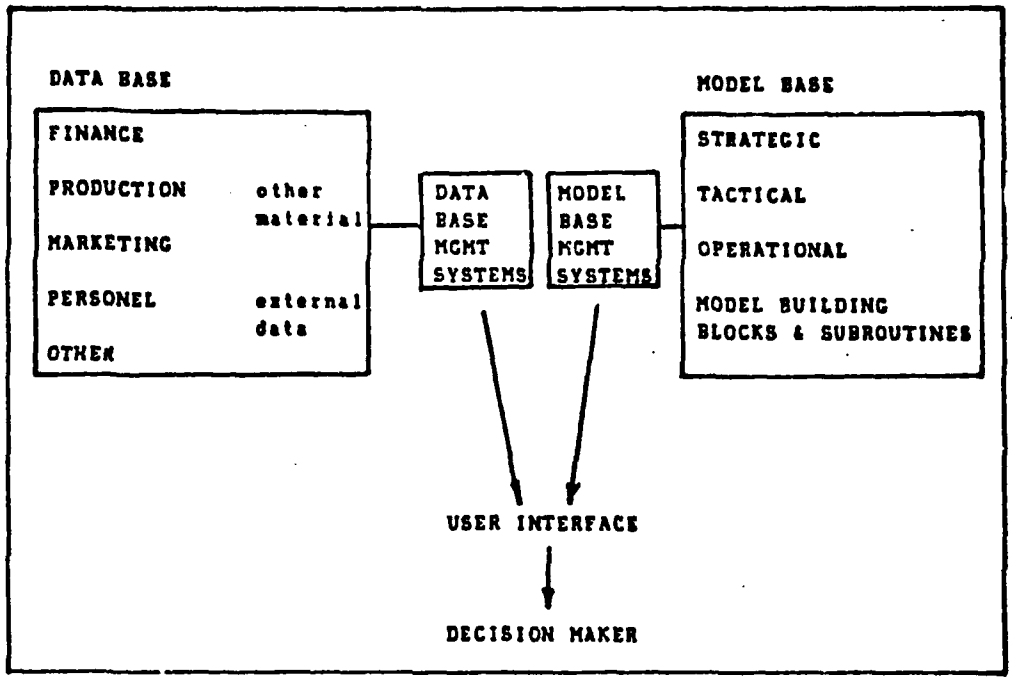


Figure 2 Decision Support Systems (Ref 18:64)

Recent work done by Decisions and Designs, Inc (DDI) for the Defense Advanced Research Projects Agency had demonstrated on-line real-time decision analysis models which could (1) perform hierarchically based probability assessments, (2) perform multi-attribute, linear additive value function analysis, and (3) work with standard decision tree structures (Ref 14:8). However, these

techniques were limited by the language that was used (APL), the documentation of the programs, and the size of the problem that could be analyzed (Ref 14:9). Thus a program package was desired which would provide a real-time, user oriented decision analysis system without the limitations encountered in the DDI work.

Current Research in Real-Time Decision Analysis Systems

A program package was developed by Captain Bruce W. Morlan as a Masters Thesis for the Air Force Institute of Technology. Known as the Decision Analysis Support System (DASS). The objectives of his thesis were to create a program which could:

1) be used interactively in non-specific heirarchical decision analysis

2) provide sensitivity analysis which could be used in an interactive dialog

3) demonstrate some display formats which could be used to get information to the user

4) act as a foundation upon which to build a decision analysis package

5) demonstrate some of the algorithms which can be used for manipulating hierarchical information

6) be documented sufficiently for other prospective users to implement and modify (Ref 14:4)

In addition to the above objectives, another co-objective was to demonstrate the program for use on a small microcomputer (Apple II). Microcomputers have the advantage of being highly portable and very low

in cost. Another advantage of microcomputers, and in particular the Apple II, is that the system is capable of color graphics which could greatly enhance the utility of displays. This capability is not readily available on larger machines except at very high cost. Unfortunately, due to unforseeable delays in the microcomputer acquisition, the bulk of the objectives were met using the Air Force Institute of Technology's CDC 6600 system using FORTRAN. However, a program was written for the Apple II, using Applesoft II BASIC to demonstrate feasibility, and many FORTRAN DASS options were included (including sensitivity). Many suggested improvements to DASS were suggested by Captain Morlan for the microcomputer system. One was incorporating color graphics to the Apple system through computer structuring (the current version was limited due to memory restrictions of graphics using Applesoft II BASIC). Introduction of graphics for displays and interactions would enhance the decision maker's interface with the DASS (Ref 18:66).

Another area of improvement for the microcomputer system is in the area of language. BASIC was selected in the demonstration for the Apple II primarily for convenience and language availability. However, the language suffers from not being very transferable to other systems. Since 1979, new standardized compiler languages have been made available for microcomputers, primarily USCD Pascal. Such a language has two advantages (1) being a compiler, the actual executable program can be run with a smaller memory, and (2) the language is standardized in the industry.

Motivation for Continued Research

Decision analysis requires tools that are flexible, adaptable, and easy to use (Ref 9:116). The flexibility and adaptability of the tool is often not measured in terms of the ease in creating the original structure, but rather in the ease of modification to that structure, primarily in the area of sensitivity analysis (Ref 12:113).

In addition, as stated earlier, the incorporation of graphical representations of decision analysis techniques greatly enhances the understanding of the impact of decisions on the problem structure. Also the understanding of the uncertainty of the attributes is enhanced.

The use of microcomputers can be of great service to decision analysis due to their low cost and graphics capability. In addition, they are highly portable and not dependent on larger machines for operation.

II The Model and Area of Potential Applications

This section will introduce the model that the Decision Analysis Support System (DASS) supports and the procedure that should be used in order to implement the DASS model.

The Model - Worth Assessment

The DASS models worth assessment which was introduced by J. R. Miller III in 1967 as a method for determining preferences when multiple and competing attributes are involved (Ref 10). The term worth can be equated to other terms such as value or utility, or any other words which have a definition of ". . . measure of the desirability of a thing . . ." (Ref 4:161). Worth is not a fixed value, but varies according to individual preferences. Thus worth ascribed by one individual may be quite different from the worth defined by a different individual.

Worth assessment decomposes the problem into an hierarchical tree structure of objectives and sub-objectives. The "root" or top-most node in the structure represents the main objective against which alternative solutions are evaluated. Within the tree structure, sub-objectives or goals are defined to meet the next higher objective, eventually leading to the main goal. At the lowest levels of a given branch of the tree resides the specific criteria or attributes which are either measured or directly assigned.

An example of such an hierarchical tree structure is shown in

Figure 4. The overall objective of a theatre air commander is to maintain control of enemy ground movement through the use of air power. The "root" node had three subgoals to aid in achieving the main objective and these, in turn, had sub-objectives. In this figure, the lowest sub-goals were further broken down into greater detail until the nodes were capable of being measured according to the section which follows.

Relevent features of problems which use worth assessment are:

1) There are multiple objectives and assessment sub-objectives to be considered and arranged in some organised form.

2) There are multiple factors whose attributes must be predicted.

3) There are multiple worth connections between the sub-objectives and the attributes.

4) There is physical interaction among the attributes.

5) There is often worth interdependence among the sub-objectives. (Ref 4:355)

At this point, it should be clear that this model only evaluates deterministic problems where the probability of the attribute or consequences occuring (or the value being true) is unity. No mechanism inherently exists for including the decision maker's attitude towards risk (Ref 16:355). However, several authors have implemented schemes in which uncertainty is factored into the weighting structure of the tree (Ref 4:170-1) or have created new criteria relating to risk itself.

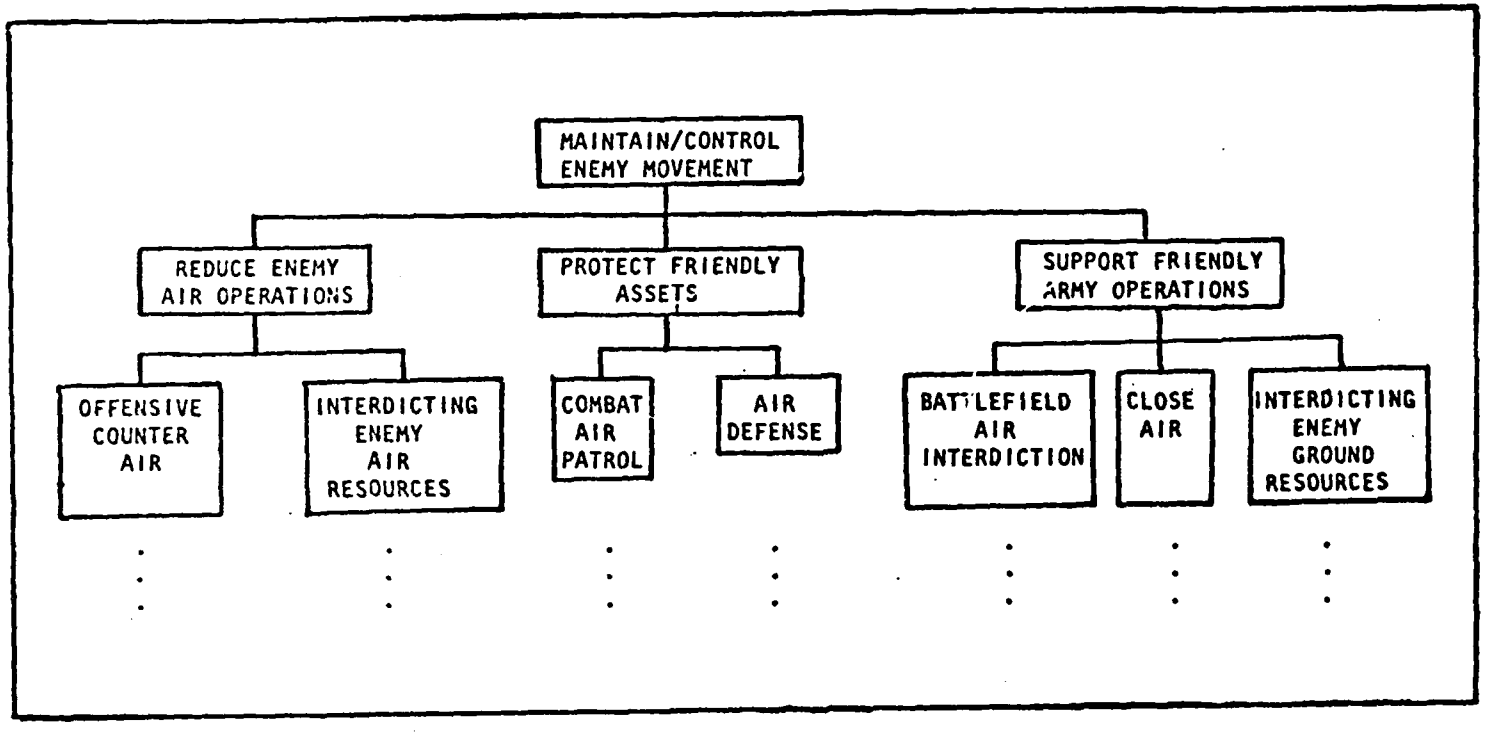


Figure 3 Hierarchical Tree Structure

Applying the Model

In applying the model, Farris & Sage suggest six steps (Ref 4:1163-1169). The DASS can be used to assist the decision maker in steps two, four, five, and six.

(1) List the overall performance objectives. The list should only contain those factors which are most important to the problem at hand. Such objectives can be obtained through many approaches (MacCrimmon - 1969). Approaches include examination of relevant literature, analytical study, and empirical empiricism. Examination of the literature can possibly reveal previous encounters with a similar problem. Analytical study includes buildin; a model of the system where variables of the problem can emerge. Empirical empiricism means to capure the "thought process" of people making similar decisions. Another method, which is gaining use in government and in private industry is to use a "panel of experts" to obtain the various lists of objectives (Ref 10:35).

(2) Construct a hierarchy of performance criteria. Based on the objectives enumerated in step one, determine which objectives are superior to, equal to, or subordinate to other objectives. The resulting tree structure will be developed with the overall goal or objective at the top of the subordinate objectives branching below. While constructing the tree, additional sub-objectives or criteria may be uncovered and added. Miller points out the purpose is to explicitly

state what is intended by or included by a particular objective (Ref 13:38). The result of this is to provide a view or pictoral map of the problem showing the interactions and relationships among the sub-objectives.

DABS can provide assistance by providing graphical representation of the decision tree (using a wiring diagram format). The decision maker can add or delete sub-objectives at will in order to achieve the desired tree structure.

(3) Select appropriate physical performance measures. In creating the tree structure, the decision maker will have a lowest-set of sub-objectives (also known as attributes or data nodes). These attirbutes should, as a group, have several properties. Attributes should be complete, operational, decomposable, non-redundant, and minimum size (Ref 10:50-52,14:16-17). In addition, these attributes should be able to have some physical characteristic assigned to them. In order to determine the appropriateness of a measure, Sage suggests that if changes in the state of the measure brings significant changes in the extent of goal satisfaction then the measure is appropriate (Ref 16:356).

(4) Define the relationship between attributes and physical performance measures, that is, deal with the scoring problem. The previous step dealt with the existence of attributes and not with the

particular values. Scoring functions must be established to transform the physical characteristic into a worth value. Miller uses a set of seven conventions that can be useful in establishing worth value functions (Ref 13:44-45). Note that this transformation is only done among the attributes and not the sub-objectives. This is because sub-objectives are related to the attributes or lower objectives and their values will be based on the contribution of the lower level objectives and attributes.

There are many scoring procedures that can be used to establish the worth scores (Rev 13:147-187). The DASS was designed to only use the resulting worth scores (measured from 0 to 100 as opposed to Miller's scale of 0 to 1) as inputs to the tree attributes.

(5) Establish relative importance within the suboriteria set. When the tree was originally established in step 2 of this procedure, many objectives were divided into immediate sub-objectives all contributing to the higher (or parent) objective. The problem now is to decide how much each sub-objective (or child) contributes to the parent objective. The resulting weights (a measure of relative importance) are assigned to each set of children such that the sum of the weights are unity. One method in assigning weights is to rank a particular set of children (Ref 16:357-358). Assign a temporary value of one to the most important child. Then estimate how much less important the next sub-objective is. For example, if the next sub-objective is three-fourths as important as

the most important sub-objective, then assign the temporary value of 0.75 to it. Then rate the third child against the second. If the third child is one-half as important as the second, then the third child is 0.375 as important to the first and the temporary value of 0.375 is assigned to the third child. The process continues until the last child is evaluated. At that time, all the weights are then rescaled such that their sum is unity. If we let b(p) be the temporary weight of the most important child (of p children), the normalised weight of any child i will be:

$$\text{Normalised Weight} = \frac{b(i)}{b(1) + b(2) + \ldots b(p)}$$

The DASS provides the user the capability to enter the importance of one node relative to the most important and automatically normalises the entered weights. The program also allows the user to modify these weights for a given structure at will.

(6) Adjust the weights to reflect confidence in the performance measures. After the weights and values have been established in the proceeding steps a cumulative weight can be established for each node by multiplying the weights of itself and all its parents. This cumulative weight reflects the node's contribution to the entire tree structure. By multiplying the worth scores for each attribute by its cumulative weight

and then summing the resulting products of all the attributes, a total worth score for a given alternative can be established.

As with all systems, all figures or values carry a certain degree of uncertainty. One method to deal with uncertainty is to adjust the cumulative weights of the attributes by a factor whose value is between zero (no confidence) to one (certainty). The cumulative weights are then re-normalised and the resultant new weights are then used in evaluating alternatives (Ref 16:358).

DAFS uses sensitivity analysis to deal with uncertainty. An advantage of sensitivity analysis over adjustments is that a particular sub-objective or attribute taking on a range of values can be examined to see if the alternative ranking of the overall objective changes. A disadvantage is that only one objective or attribute can be examined at a time.

III. Implementation

The main thrust of this thesis was to combine three different technologies to demonstrate both the feasibility and practicality of employing the DASS. These technologies were microcomputers, computer graphics, and sensitivity analysis. In this section, discussion will center about each technical area discussing both the objective and the method used to reach the objective.

Microcomputers

One objective pursued was to install the DASS on the Apple II microcomputer and to demonstrate the microcomputer's capability. As mentioned earlier, microcomputers have the advantage of being portable. In addition, microcomputers offer advantages of lower capital investment (typical systems are under $3000) and lower power costs (Ref 19:77).

The Apple II was selected because of its prior demonstrated performance using the BASIC version of DASS and its immediate accessibility. In addition, the Apple II had the capability of producing high resolution color graphics (280 x 191). These features coupled with the availability of the PASCAL compiler indicated than an expansion of the BASIC DASS program to be both feasible and doable.

PASCAL is a fairly new language. It is the first language to embody the concepts of structured programming defined by Edsger Dijkstra and C.

A. R. Hoare. The language was developed by Niklaus Wirth at Eidgenossische Technische Hochschule in Zurich and is a derivative of ALGOL 60 (Ref 7:iii). The main advantages of PASCAL are in its program structure and its data definition (Ref 15).

An implementation of PASCAL, specifically for small machines like the Apple II, is USCD PASCAL developed by the Institute of Information Sciences at the University of California at San Diego, under the direction of Kenneth L Bowles. USCD PASCAL differs from the PASCAL defined by Kathleen Jensen and Niklaus Wirth in their PASCAL USER MANUAL AND REPORT primarily in the areas of files and input/output operations. In the PASCAL version of DASS, one difference that was used in USCD PASCAL was the use of random access of data files (Ref 17:151). The largest difference, though, was in the use of segmented procedures.

Segmented procedures allow a large program to be divided into smaller parts which are then entered into the computer memory as they are needed. This process is similar to overlaying found in FORTRAN. This option allows for a computer compiled program to be several times larger than the intrinsic computer memory which results in more processing and display capabilities. In addition, segmented procedures are advantageous because the variables common to all procedures are immediately available, a fact which is not readily available in other schemes in program chaining (for example, BASIC).

In the development of the enhanced DASS, segment procedures were created along the lines of major program options such as creating the

tree structure (option SPA) and loading attribute values and sub-objective weights (option WVC). In all, six segment procedures were used out of the seven available which allows for some future expansion of the DASS.

Computer Color Graphics

The second objective which was pursued was to exploit color computer graphics, in particular, the color graphics available to the Apple II PASCAL system.

The field of computer graphics is a new, rapidly developing field. Its chief advantage can be summed up by the old Chinese proverb of "one picture is worth a thousand words". Approximately eighty percent of information that is remembered is recieved through visual stimuli (Ref 1:106). In addition, the human mind is the best available pattern recognition computer. This coupled with the fact that graphic formats convey quantitative data as patterns in physical space, the human can assimulate large amounts of data to make decisions (Ref 6:182). In fact, graphical representations of data is used to clarify trends, identify the magnitude of trends, facilitate comparisons, aid in retention, and focus attention on the significant aspects of the information (Ref 2:26).

A two phase effort was used to achieve the objective of color computer graphics. The first was to enhance the graphical presentations in the original version of the DASS (Ref 14). These presentations were

primarily found in the display and sensitivity modules of the program. The second was to add new graphic displays. These displays were in the area of input, primarily the hierarchical tree itself and in assigning worth values to the attributes. Before discussing these phases in detail, some general comments need to be made concerning the color system used.

Colors available to the Apple II PASCAL system were orange, violet, blue, green, and white as well as the background color of black. In order to maintain consistency throughout the program, a unique color was assigned to each alternative. Thus the first alternative was assigned color orange, the next violet, and so on. A legend relating colors to alternatives appeared on each graphics figure which contained the alternatives. This provided the user with an immediate reference. A second use of the legend was to provide information when, in future versions of DASS, the graphics display could be printed directly to a black and white printer (Reasons for not using a color printer is primarily cost (Ref 6:189).

Improving Existing Displays. The original version of the DASS used alphanumeric characters in drawing the graphs (Ref 14). This was because the computer that the original version was run on did not have an available, interactive graphics capability. In addition to the use of characters in graphs, alternatives were labelled by alphabetical characters (that is an A represented the first alternative, B the second, and so on). In the enhanced version, the alphanumeric graphs

were replaced with lines and alternatives were colored and represented by lines or bars. An appreciation of the differences can be seen in Figures 4 and 5.

Regarding the display for the sensitivity analysis, very little was needed to improve the display. As can be seen in Figure 4, changes were to label which type of sensitivity analysis was used (only one type of analysis was possible in the original version. See the section on sensitivity analysis). In addition, note that the objective/attribute information is present in the newer display. The original version did not have this information with the graph; however, the analyses was normally done at a printing terminal which recorded all appropriate node information used in the sensitivity analysis.

While the display for the sensitivity analysis changed very little, the display for information regarding a node changed greatly (Figure 5). In the improved version, a wiring diagram was employed. The reason for using a wiring diagram was that decision makers, especially military decision makers are used to seeing hierarchical structures in wiring diagram formats as in organizational charts. In addition to the wiring diagram, individual color bars were used to denote the values of each alternative at each sub-objective. The bars provide, at a glance, a comparison among the aternatives. The bars were drawn vertically as opposed to any other direction because lines drawn vertically represent the most accurate and the most crisp representation of the data based on the Apple computer electronics. In addition, the wiring diagram (boxes

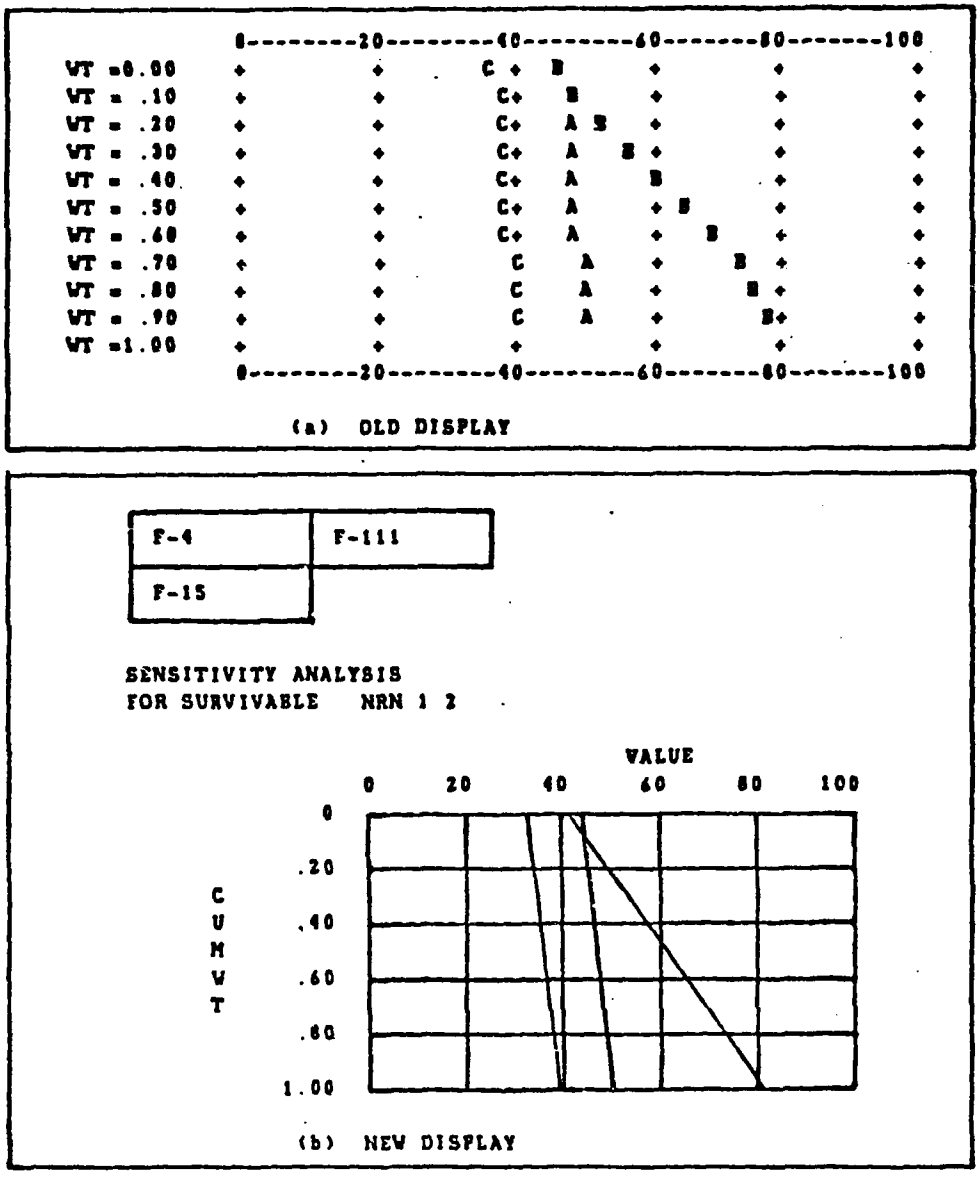


Figure 4 Old and Improved Sensitivity Display

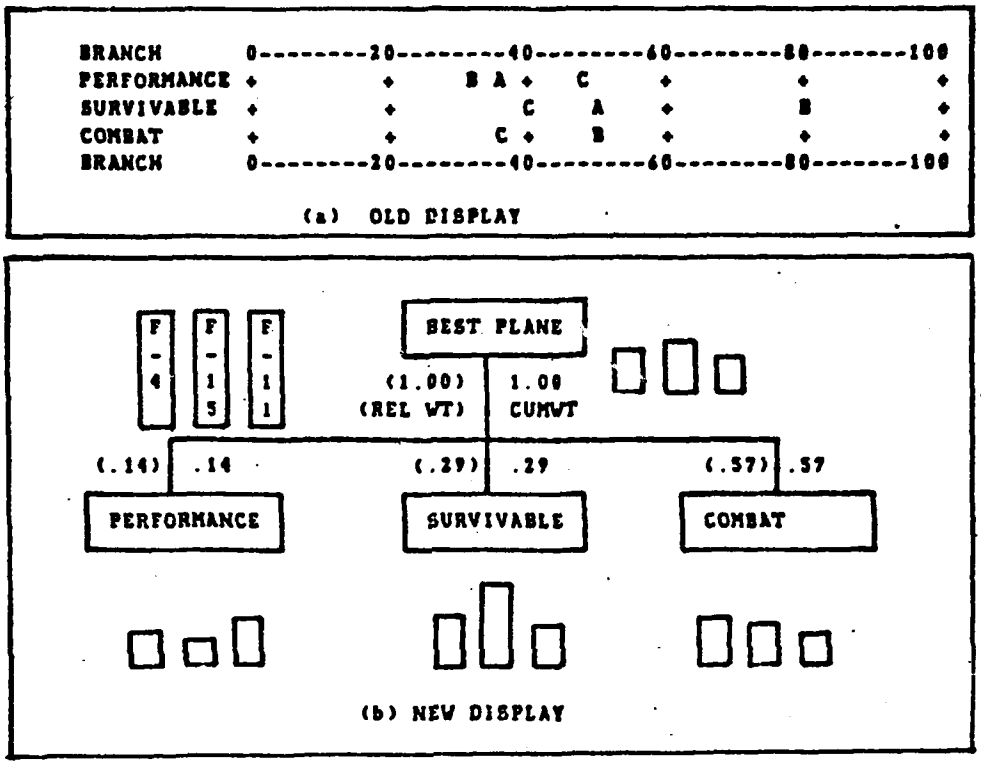


Figure 5 Old and Improved Node Display

and wires) was drawn in green in order to improve viewing.

Adding New Displays. Displays that were improved in the enhanced version of the DASS were oriented towards the output of the program. In addition, attention was directed towards exploiting color graphics on input. The challenge of using graphics for input is to be flexible and robust to users errors as well as pictorally displaying the input data. Two areas were identified as candidates for this type of display. They were the hierarchical structure itself and the worth values of the attributes.

In the original version of the DASS, the hierarchical tree was input one span at a time where the computer would display the title of the parent objective and the user would enter the immediate descendents. In the enhanced version, the same information is asked for, but through a tree diagram where the user inputs the title of the descendents (Figure 6). As can be seen in Figure 6a, the parent objective is at the top of the wiring diagram with a box for the first descendent. As immediate descendents are added, new boxes are drawn awaiting input (Figure 6b,c). When no more descendents are to be added, the program asks for descendents of the first sub-objective (Figure 6d), and so on. Again, the wiring diagram scheme was used to provide a setting that would be comfortable to the user.

For entering worth values, the original version of DASS appraised the user of the previous alternative values of the attribute and asked for the new inputs. This method of input is not bad, especially when

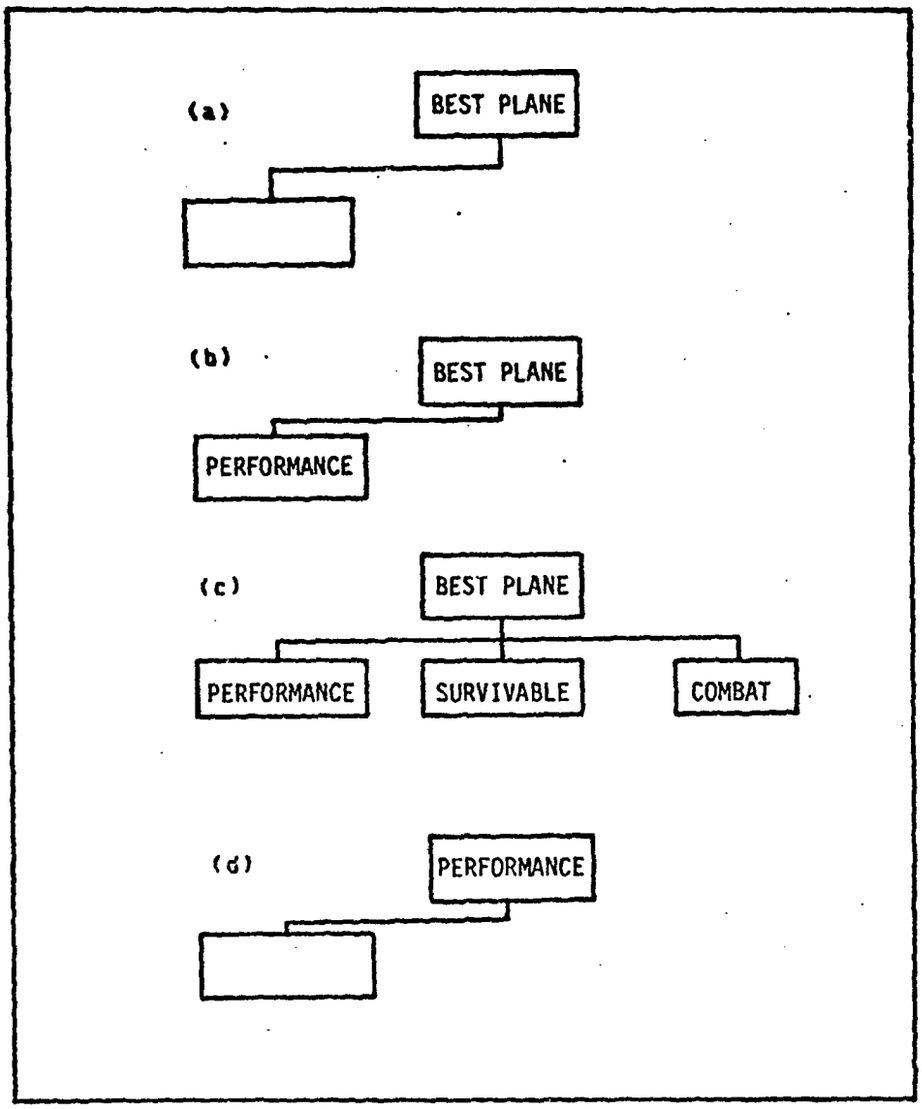


Figure 6 Hierarchical Tree Input Display

using worth functions to transform a measure such as dollars and hours into worth values. However, when a direct worth estimate needs to be made for an attribute, a visual comparison can be useful. Hence the display shown in Figure 7 was developed. As can be seen in the figure, the current value of the alternatives are presented both numerically and in bar graph format. The user then inputs the new value in the box at the bottom of the screen and upon completion, the computer converts the entry into a bar and places it under the NEW VALUES area, in the alternative's color. Thus the numerical input values are transformed and spatially displayed as an aid in value interpretation.

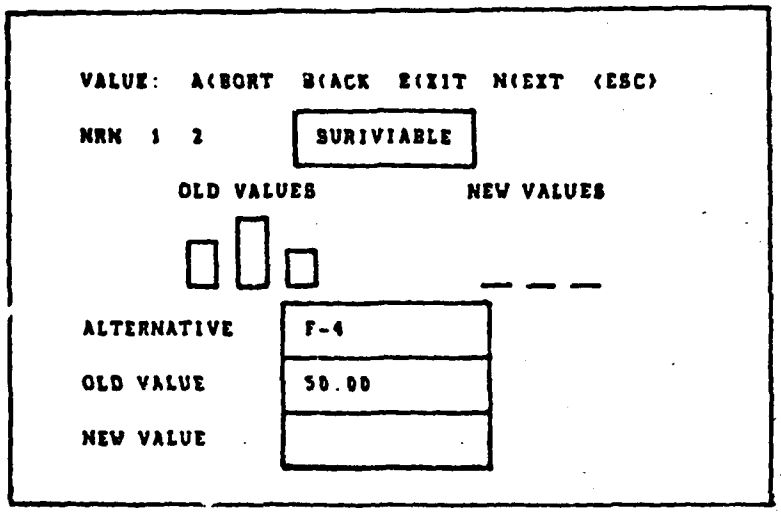


Figure 7 Attribute Value Input Display

Additional Sensitivity Modules

The third area incorporated into the DASS program was to provide additional capability in the interactive sensitivity analysis. The chief advantage of the original DASS was the fact the sensitivity analysis could be done in a real-time basis (that is, while the user is at the console). In fact, the major drive for all decision support systems is to give the user immediate answers to "what if" type questions (Ref: 6:182,5:54). In the original work, any objective or attribute node could be varied on the basis of its overall contribution to the hierarchical structure (Ref 14:Programmer's Manual:2-4). The resulting changes in the value of the "root" node for the alternatives provided an indication of how sensitive the incumbent best alternative would be if the overall "strength" of a particular objective varied.

As one can easily guess, the types of "what if" questions can be both infinite in number and extremely difficult in complexity. However, two additional "what if" areas were investigated in addition to the cumulative weight model presented in the original and enhanced versions of the DASS. They were (1) what happens to the alternative selection if the relative weight of a particular node changes value among its siblings?, and (2) what happens to the alternative selection if the value of an alternative changes for a particular attribute?

Sensitivity of Relative Weight. What we are examining is the change in alternative values at the "root" node based on a change of a relative

weight on a span anywhere in the tree. Note that the "root" node is the overall objective of the tree structure.

In performing the analysis, we need to look at how the values of a node is calculated. Note that for any node, the value of an alternative for that node is just the sum of the products of each immediate descendent alternative value and its relative weight or:

$$\text{VALUE(PARENT NODE)} = \sum_{\substack{\text{all} \\ \text{immediate} \\ \text{descendents}}} \text{RELATIVE WEIGHT(DESCENDENT)*VALUE(DESCENDENT)} \qquad [1]$$

Thus we can say that a change in the relative weight of any node will only affect the value of the parent node. In addition, note that the value of any node is only dependent on its own immediate descendents and not on the relative weight of the node itself. Therefore, in examining a change of the relative wieght of a node, the new alternative values of its parent need to be calculated and substituted for the incumbent values of the parent. This substitution will then affect the value of the "root" node.

Substitution of the new values into the "root" node, fortunately is straight-forward. Note that the value contribution of any node to the "root" node is just the product of its value multiplied by its cumulative weight or:

NODE'S CONTRIBUTION TO THE ROOT NODE = CUMULATIVE WEIGHT(NODE) * VALUE (NODE) [2]

Since the cumulative weight of the parent is unchanged, the change in the value of the parent can be added to the "root" node or:

NEW VALUE(ROOT NODE) = OLD VALUE(ROOT NODE) -
(CUMULATIVE WEIGHT(PARENT) * OLD VALUE(PARENT)) +
(CUMULATIVE WEIGHT(PARENT) * NEW VALUE(PARENT))
[3]

A question here arises as to how to distribute the remaining relative weight to the node's siblings. An arbitrary rule was made to keep the relative weights of the siblings at the same proportions as was in the incumbent situation. For example, let us have three objective nodes A, B, and C. Let each of these objectives have the relative weights of 0.7, 0.2, and 0.1 respectively. If we change the relative weight of A from 0.7 to 0.8, then node B will have a relative weight of 0.133 and node C will have a relative weight of 0.067. If we varied node A from 0.7 to 0.6, node B will have a relative weight of 0.267 and node C 0.133. Note that the ratio of the weight of B to C is unchanged in all cases.

Thus the overall procedure in examining the effects of a change in a relative weight of a given node on the "root" node is:

(1) Pick a new relative weight of the node to be examined.

(2) Redistribute the remaining relative weights (that is one minus the relative picked in step (1)) among the node's siblings.

(3) Recalculate the value of the node's parent using the new relative weights selected in steps (1) and (2) using equation [1]. Note that the values of the node and its siblings are unchanged.

(4) Substitute the new value of the parent node in place of its old value at the "root" node using equation [3].

Sensitivity of Attribute Value. The objective in this sensitivity analysis is to examine the effect of varying an attribute alternative value on the "root" node.

In this analysis note that varying a value of an alternative does not affect the values of any other alternatives (independence among alternatives). In addition, note that changing values does not affect the tree structure in either the cumulative or relative weights. Therefore, a change in the "root" node can occur only for the alternative varied and then by the amount of the cumulative weight of the attribute or:

$$\begin{aligned} \text{NEW VALUE(ALTERNATIVE,ROOT NODE)} = \text{OLD VALUE(ALTERNATIVE,ROOT NODE)} - \\ \text{CUMULATIVE WEIGHT(ATTRIBUTE)} * \text{INCUMBENT VALUE (ALTERNATIVE,ATTRIBUTE)} + \\ \text{CUMULATIVE WEIGHT(ATTRIBUTE)} * \text{NEW VALUE (ALTERNATIVE,ATTRIBUTE)} \end{aligned} \quad [4]$$

Note that varying the value of an alternative for an attribute only affects the value of the "root" node for that alternative and the values

of the "root" node for all other alternatives are unchanged.

Thus the procedure in examing the effects of changing an alternative value for a given attribute is:

(1) Select a new value of the attribute for a given alternative.

(2) Calculate the new value of the "root" node for the given alternative using equation [4]. All other alternative values for the "root" node will remain unchanged.

IV Examples

Two problems using worth assessment were evaluated using the DASS. The following few paragraphs discusses each of the applications.

Tritium Production Problem

The first problem dealt with determining the best production technique to develop tritium. The group studying the problem was unsure as to which of four alternatives (3 different modifications or a new facility) would be best based on several competing objectives.

After some discussion, the group decided that the problem could best be decided using the worth assessment procedure. They decided on an hierarchical structure shown in Figure 8. Based on this particular structure, the group developed weights and values for all applicable nodes. At that point, the hierarchical system was entered into the DASS, and evaluated.

The resulting value for the overall objective (BEST METHOD) is shown in Figure 9. The figure shows a consolidation of all nodes into the total line in the figure.

Based on these results, questions were raised regarding the sensitivity of each sub-objectives immediately descendent to the overall objective namely: cost (COST), production methods (PROD METH), technical risk (TECH RISK), time (TIME), and public reaction (PUB REACT).

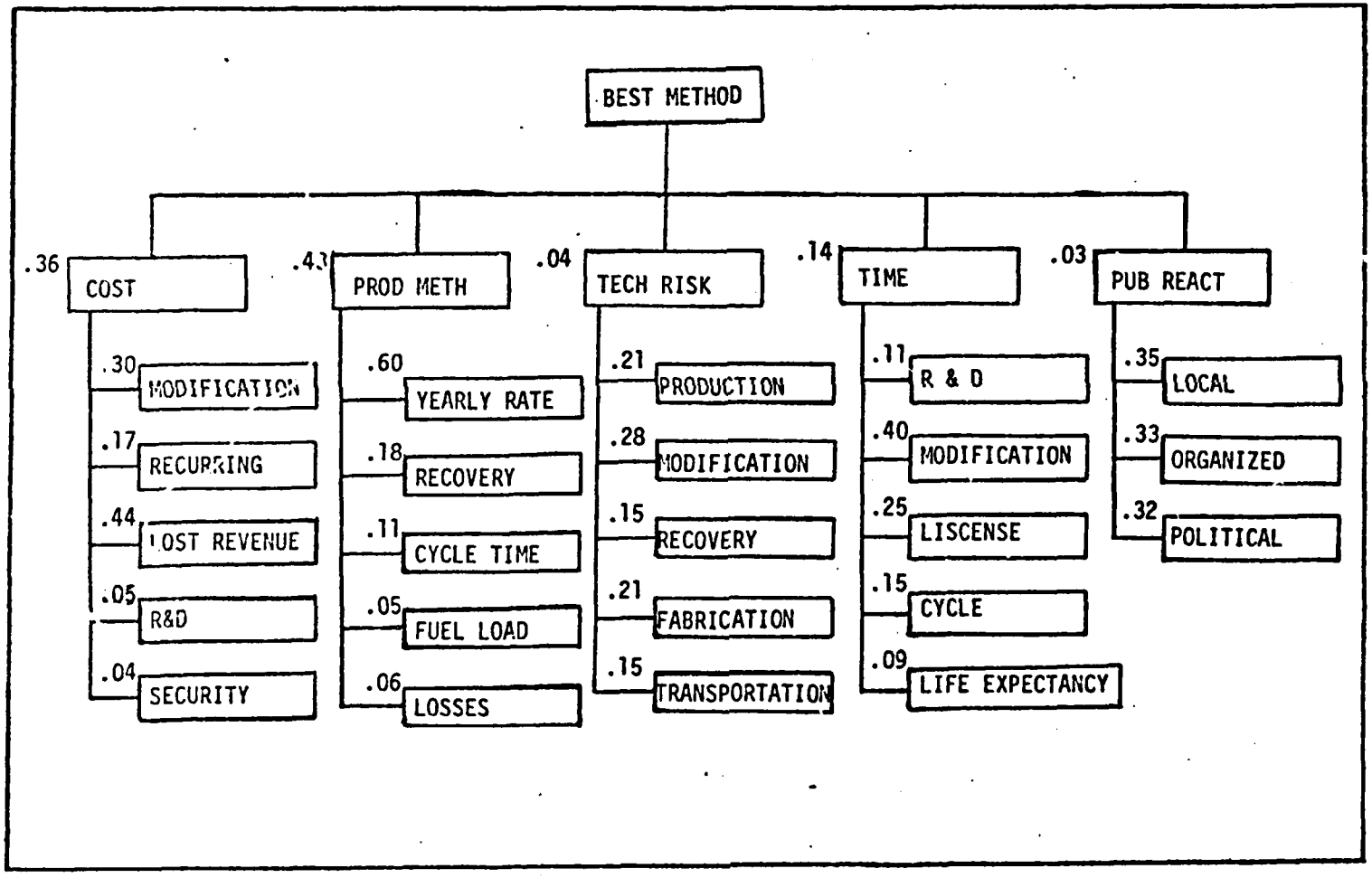


Figure 8 Tritium Production Tree

BEST METHOD FACTOR	IN-CORE	OUT-CORE	SHIELD	NEW PLANT
COST	77.00	86.03	87.45	65.08
PROD METH	53.00	53.60	33.45	100.00
TECH RISK	75.93	77.75	71.60	80.40
TIME	83.75	85.55	76.55	24.00
PUB REACT	43.20	43.20	63.95	58.10
TOTAL	66.56	70.40	61.36	74.71

Figure 9 Results for Tritium Production

Thus a sensitivity analysis on the cumulative weight were conducted using the DASS program for each of these nodes.

The results of the sensitivity analysis indicated that changing the emphasis on cost from 0.36 to 0.50; or production factors from 0.43 to 0.40; or time from 0.14 to 0.20 changed the result of the best alternative from a new plant to one of the modifications. However, changing the emphasis on technical risk or time over the entire range of cumulative weights for these sub-objectives did not alter the selection of the best alternative. Therefore, based on the sensitivity of the best solution, especially with regard to cost, production factors, and time, additional analytical investment should be made to insure the accuracy and stability of the entered weights and values.

Advance Weapon System Selection

The second problem dealt with determining the better of two new competing weapon systems for strategic attack. The team studying this

particular identified two competing sub-objectives and therefore decided to use worth assessment via the DASS.

After discussing the problem, the study team developed the hierarchical tree structure shown in Figure 10.

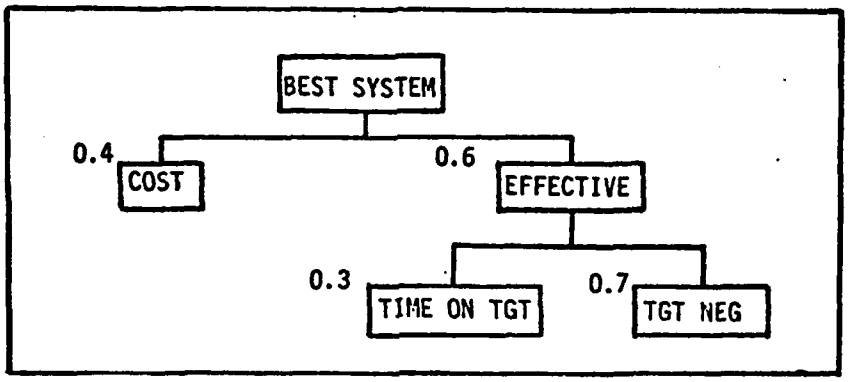


Figure 10 Advance Weapon System Tree

Based on this structure, the team then determined the appropriate weights and values and entered the data into the DASS program with the results shown in Figure 11.

BEST SYSTEM FACTOR	SYSTEM 1	SYSTEM 2
COST	85.00	55.00
EFFECTIVE	64.00	71.5
TOTAL	72.40	64.90

Figure 11 Results for Best Advanced System

Based on these results, questions again were raised as to the

sensitivity of the best alternative (System 1) if the weight of cost (COST) varied. Using the DASS, the sensitivity analysis showed that System 2 would be preferred if the cost were to be reduced in importance from 0.40 to 0.20. Thus additional effort to further define the weights used in this structure may not be fruitful.

An additional question was raised as to the effect of varying the relative importance of time on target (TIME ON TGT) to its sibling, number of targets negated (TGT NEG). The results, using the DASS, indicated that System 1 would be the preferred alternative regardless of what the relative importance was between these two nodes. Therefore, further refinement of these weights would not be meaningful.

V Conclusions and Recommendations

Many objectives were pursued in this research. These objectives were the implementation of the DASS on a microcomputer, the use of color computer graphics for input and output displays, and the addition of sensitivity analysis modules in the DASS itself. These objectives will be addressed individually. At the conclusion of these comments, a list of recommended extensions to this thesis will be presented.

The first conclusion that can be drawn is that the DASS can be adapted to a microcomputer including computer graphics. Prior to this work, DASS was implemented on a microcomputer, but was unable to use the microcomputer's graphics capability. This work implemented the DASS and was able to provide graphics support. In addition, options not available in the microcomputer program, displaying a node and pruning the hierarchical tree were installed under this enhanced version of the DASS. These accomplishments were primarily due to the use of a compiler language, PASCAL, which enabled the program to be executed in a much smaller memory than the previous version. In addition, the use of segmented procedures enabled the program to be executed successfully even though the program was much larger than the actual computer memory.

A second conclusion is that computer graphics does enhance the output of the DASS. Color graphics provided additional visual emphasis to those figures displayed in the original version and new insights to those graphics modules added. The use of colored bars in many displays

allow the user to interpret numbers, at a glance: a job graphics does best.

The third conclusion reached is that sensitivity analyses about a node's relative weight or an alternative attribute value can be accomplished. In fact, the sensitivity analyses in these areas were fairly straight forward. Facts that made the problem straight forward were the assumptions of the worth assessment model concerning linear additivity, attribute value independence, and constant marginal rate of substitution (Ref 14:19-20).

While this work made advancements in combining technologies relevent to decision analysis, much further work needs to be done. Five areas are recommended. They are:

* Creating worth value functions to transform measures into worth values. The current program only accepts worth values.
* Enhancing graphical displays. Additional graphics could be added to the program in the area of multi-node displays.
* Improving node reference nomenclature. This could be done by examining the merits of the node reference number system used to identify a particular node as opposed to possible user oriented method (such as the title of the nodes themselves).
* Extending the sensitivity analysis to two or more objectives/values and provide graphical support.
* Examine the merits of audio inputs and outputs in the DASS.

In conclusion, decision analysis is coming of age and new

technologies such as microcomputers and color graphics are becoming readily available. The future holds a double challenge for those who dare: to provide answers to questions which are complex, vague, and difficult, and to present those answers in the most meaningful way to the decision maker.

Bibliography

1. _____ "One Picture Worth Many Words", Computer Decisions, 12:106-7 (May 1980)

2. _____. "Graphics for Everyone (Even for Executives)", Computer Decisions, 11:26+ (September 1979)

3. DeWispelare, A. R. Captain. Lectures in ST 6.31, Weapon Effectiveness/Tradeoff. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1980

4. Farris, D. R., and A. P. Sage, "On Decision Making and Worth Assessment", International Journal of System Sciences 16-2:1165-1178 (December 1975)

5. Feidelman, "Microcomputers Mean Business", Infosystem 26:104+ (June 1979)

6. Friend, David, "Color Graphics Information Systems Boost Productivity", Mini-Micro Systems, 13:181-182+ (May 1980)

7. Grogono, Peter. Programming in PASCAL. Reading: Addison-Wesley Publishing Company, Inc. (1978)

8. Howard, Ronald A. "An Assessment of Decision Analysis", Operations Research, 28:4-27 (January-Febuary 1980)

9. Keen, P. G. and G. R. Wagner. "DSS: An Executive Mind-Support System", Datamation, 25:117-122 (November 1979)

10. Keeney, Ralph I. and Howard Raiffa. Decisions With Multiple Objectives: Preference and Value Tradeoffs. New York: John Wiley & Sons (1976)

11. Mandell, M. "Computer Based Planning, The Once and Future Thing", Computer Decisions, 11:38-39+ (Febuary 1980)

12. McArthur, D. S. "Decision Scientists, Decision Makers, and the Gap", Interfaces, 10:110-13 (Febuary 1980)

13. Miller, J. R. III, "A Systematic Procedure for Assessing the Worth of Complex Alternatives", Mitre Corporation, Bedford, Mass., Contract AF 19(628) 5165, Electronic Systems Division, Air Force Systems Command, ESD TR 67-90 (AD 662 001) Nov 1967.

14. Morlan, Bruce W., Captain. A Computer-Based Decision Analysis Support System. MS thesis, Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1979 (AD A083796)

15. Mundie, David A., "In Praise of PASCAL", Byte, 3:8:110-116 (August 1978)

16. Sage, Andrew P. Methodology for Large-Scale Systems. New York: McGraw-Hill Book Company (1977)

17. Shillington, A., et al. Apple PASCAL Reference Manual, Apple Computer Inc (1979)

18. Sprague, R. H. Jr., and H. J. Watson "Bit by Bit; Toward Decision Support Systems", California Management Review, 22:60-68 (Fall 1979)

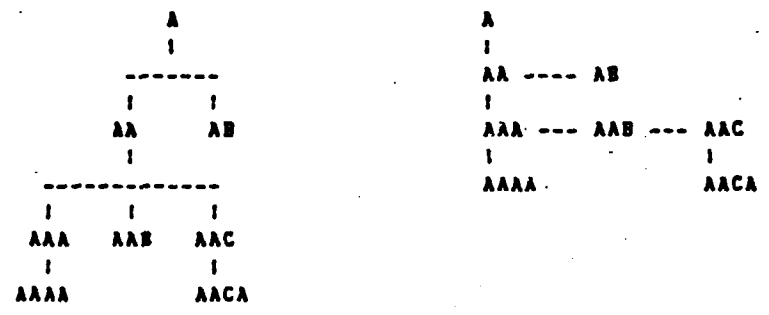
19. Wayne, F. P. "3000 Aros for 3000 Bucks: How to Justify a Personal Computer for Your Own OR/MS Department", Interfaces, 9:75-80 (August 1979)

APPENDIX A

GLOSSARY

Glossary

Note - the glossary will refer to the following example tree hierarchy to demonstrate some of the concepts defined.



(tree structure) (data structure)

The Sample Hierarchy

backlink - this is a data structure concept. The node which proceeds a node in the data structure is backlinked to that node. In the example, node (AA) is the backlink to node (AB) and node (A) is the backlink to node (AA).

branching node - a node (or objective) which has at least one descendent.

cell (or node record) - refers to the block of data associated with a node. DASS uses cells to store the pointers and the data for each node. The pointers stored are the downlink pointer, the crosslink pointer, and the backlink pointer. In addition

information relating to the node such as the node digit, the relative weight, the cumulative weight, and all alternative values are located in a cell.

children - this is a concept of hierarchies. The children of a node are those nodes which are one level down from that node. In the example, nodes (AA, AB) are children of node (A) and nodes (AAA,AAB,AAC) are children of node (AA).

crosslink - this is a data structure concept. The first sibling node which is added after a node is that node's crosslink node (crosslinked to that node). In the example, node (AB) is the crosslink to node (AA) and node (AAC) is the crosslink to node (AAB).

cumulative weight - this is the weight of a node, relative to the root node. This is a measure of the contribution of the node to the entire hierarchical structure. The cumulative weight is equal to the product of the relative weight of the node and the cumulative weight of the parent (the cumulative weight of the "root" node is unity).

data node (also attribute) - a node which has node descendents.

descendent - a node which comes after a given node in the hierarchy.

depth-first search - a synthesis of data and logic structure concepts. A depth-first search traverses a tree by:
(1) adding new levels first (if possible)
(2) then visiting the crosslink nodes
A depth-first traversal of the example would visit the nodes in the following order:
A AA AAA AAAA AAB AAC AACA AB

level - this is refers to the depth the node is from the "root" node. For example, node (A) is a level 1 node; node (AAAA) is a level 4 node.

node - this is an element in the hierarchical or data structure. All the

Glossary (cont)

elements in the example are nodes. With regard to worth assessment, nodes can be objectives or attributes depending on their position in the tree.

node digit - is the number which states the position of a node on its span. In the example, the node digit of node AAB is 2. The node digit of node AA is 1. The node digit of AACA is 1.

node reference number (NRN) - the vector which points the path through the tree from the top down to a particular node. Each node has a unique NRN. The NRN is made up of node digits. For each level down, an additional digit must be added to identify a node. For example node (AAAA) has a NRN of 1 1 1 1. Node (AACA) has an NRN of 1 1 3 1. Node (AB) has an NRN of 1 2.

parent node - this is a concept of hierarchies. The node directly in line one level above a node is the parent node to that node. In the example, node (A) is the parent of nodes (AA) and (AB). Node (AA) is the parent of nodes (AAA), (AAB) and (AAC).

relative weight - this is the importance of the node relative to its siblings. These weights have the value between zero and one, and the sum of all siblings of a particular parent is unity.

root (or "root" node) - this is a concept of hierarchies. The root node is the top most node in the hierarchy and represents the overall objective of the hierarchical structure.

sibling - this is a concept of hierarchies. Those nodes which have the same parent node are siblings. In the example, nodes (AA) and (AB) are siblings as are nodes (AAA), (AAB) and (AAC).

span - this is a comcept of hierarchies. This term is associated with any given set of siblings. For example, nodes (AA) and (AB) are on a span.

value - this is a concept of worth assessment. Value is the worth of a measure of a particular attribute for a given alternative.

APPENDIX B

USERS MANUAL FOR DASS

Table of Contents

List of Figures

OVERVIEW

The Decision Analysis Support System (DASS) was originally developed by Bruce W. Morlan to provide user oriented automated support to decision analyses where a deterministic, linear additive, and hierarchical decision structure exists. This version of the DASS was specifically written for use on microcomputers, in particular, the Apple II. Machine requirements for running the DASS is 64K memory (Apple II) and at least one 5 1/4 inch floppy disk drive. In addition to those functions available in the original DASS system, additional features have been added. These features include expanded sensitivity analysis capability and computer color graphic displays.

I. CAPABILITIES AND LIMITATIONS

Every software tool has a set of capabilities and limitations. The DASS is no exception.

Capabilities

The DASS is capable of analysing a decision analysis structure which is deterministic, linear additive, and hierarchical in nature.

The program allows for the use to input interactively the tree structure, relative weights and attribute values through the use of various program options. The input is enhanced through the user of computer generated graphics. Further, the program automatically calculates the cumulative weights of all nodes (also known as collasping the tree), and determines the composite value of the "root" or overall objective node as well as all intermediate nodes based on the attributes in the tree structure. The system also allows the user to interactively ascertain the status of any node. In addition, data are graphically displayed and incorporate color for quick data examination and evaluation.

The most important feature of the DASS is the ability to conduct interactive sensitivity analyses from a variety of perspectives. The first perspective is with regard to the changing of the cumulative weight of a node to the overall heirarchical structure. The second is to

evaluate the effect on the selection of the best alternative given pertebatic of the relative weight of a node among its siblings. The final pers, ·ctive is to examine the effect on alternative selection based on changes in value for a given system attribute.

Limitations

There are several limitations that are associated with this system. The limitations are associated with both the model being implemented and the machine on which the model is being implemented.

One model limitation is that the entries for the attribute values must be in terms of worth. That is, the mapping of the attribute measures to the worth value have already taken place.

Another model limitation is in the basic assumptions of the linear additive function used in the model. These assumptions are attribute value independence and marginal rate of substitution. These assumptions are especially critical when conducting sensitivity analyses when large deviations are made.

An additional limitation with regard to the attribute values is that they all must follow a "more is better" philosophy (that is, the higher the value the better) or a "less is better" philosophy (the lower the value the better).

Machine limitations are due to the memory constraints of the Apple system (or any microprocessor system for that matter). Thus the following r•strictions should be observed:

MAXIMUM NUMBER OF NODES: 100

MAXIMUM NUMBER OF LEVELS: 5

MAXIMUM NUMBER OF SYSTEMS: 5

MAXIMUM NUMBER OF NODES ON A SPAN: 5

In addition to these "operational" limitations, that is limitations when running the program, there are other limitations concerning the number of separate data sets. For the DASS program only three (3) data sets can reside concurrently with the DASS program when running a single disk system. If more than a single disk system is used, the second disk, which could be used as strictly a data disk, can contain up to 11 separate data sets, each set meeting the numerical limits specified above.

II. INPUT

The term "input" refers to all data which is put into the program. Input comes from two sources: (1) user's interactive responses to program questions (prompts), and (2) previously entered and stored data from disk files. When DASS requires any input from the user it will write a prompting message to the console.

Program Control

The program is controlled by user responses to questions or by user selection from a menu. Selections are made in one of several ways: (1) using a three character mnemonic, concluding by depressing the RETURN key (selecting a major option), (2) using a single letter to select from a table of sub-options, (3) by pressing the Y or N key in answering yes/no questions posed by various options.

Each major option is discussed briefly in the following flow section concurrent with a simple example. Complete discussions are found in the OPTIONS section.

Conventions

In the discussions to follow, all computer input and output is capitalized, except for output variable values, alternative listings, node reference numbers, and option names, in which case they are

separated by less-than and greater-than signs ((xxxxx)).

Terminology

This program has some specific terms with regard to the tree structure and elements within it. These terms are defined below. The reader is urged to refer to Appendix A of this thesis.

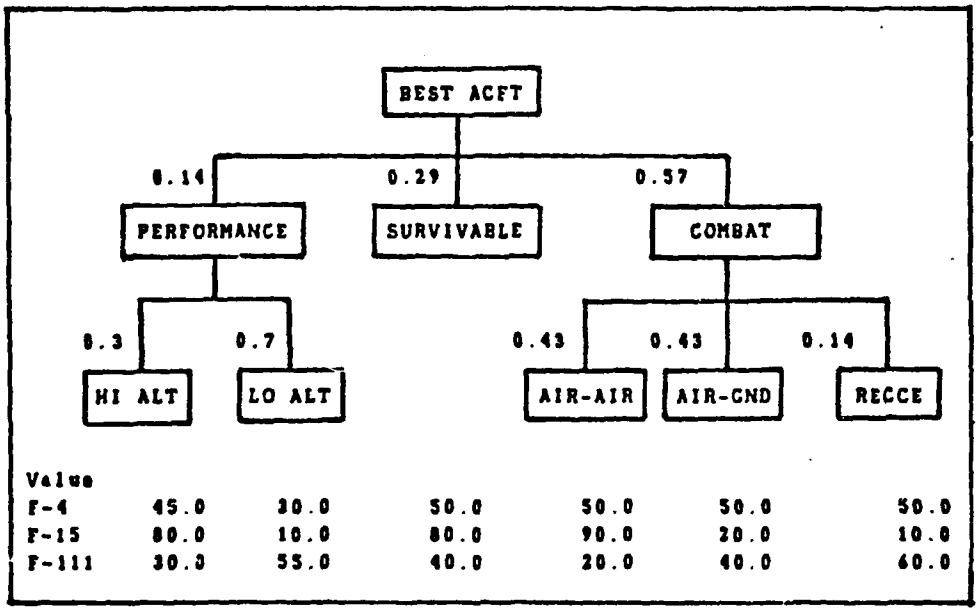
III. PROGRAM FLOW

The following discussion of the use of DASS is built around the hierarchy shown in Figure 1, which was also used in Captain Morlan's work.

In the heirarchy shown, the question that is being answered is "which of the available systems (F-4, F-15, and F-111) should we deploy to a forward base?". The attribute that we wish to determine is "the best plane", which has been described in terms of the remainder of the tree. At the point when we begin to use DASS, we have a structure (tree), tentative weights for each node and values for each alternative. The values for each alternative have been normalized to fall between the values of 0 and 100 for each attribute (the bottom-most node(s) on a given branch). In addition, the values are consistent, that is to say, that the extremes for all nodes mean the same thing (in this case, all data nodes with large values mean that the particular alternative is very good- more is better) (Figure 1).

When the DASS program begins executing, OPTION SEL (see (SECTION IV for details) is automatically executed. Typing NEW to the question as to whether the file name entered is a new file or not will result in OPTIONS ATT, TTL and SPA to be automatically executed.

Option ATT prompts the user to enter the characteristic that the attribute values will have. Any entry starting with an R will be treated



Value						
F-4	45.0	30.0	50.0	50.0	50.0	50.0
F-15	80.0	10.0	80.0	90.0	20.0	10.0
F-111	30.0	55.0	40.0	20.0	40.0	60.0

Figure 1 Sample Tree Structure

as "less is better". Otherwise a "more is better" situation among the attributes is assumed. The thing to remember is that all data entered must be consistent with the characteristic entered.

Option TTL asks the user to enter the title of the upcoming data structure. This title will be subquently used as headers for various displays that are available in the MAIN MENU part of the program.

Option SPA enables the user to then enter the tree structure. This option enables the user to quickly enter the tree on a node-by-node basis, entering descendents to a given node. The option uses graphical displays to aid the user when entering the descendents.

At the conclusion of the SPA option, the user should execute the

REV option to insure that the tree structure inserted under SPA (above paragraph) is what is desired. If various nodes need to be either inserted or deleted, the options MOD (for adding nodes) and/or PRU (for deleting nodes) should be used.

When the user is satisfied with the tree structure, the user should then execute option SYS in order to enter the alternatives (systems) that are to be considered by this tree structure.

Next, the user will need to use the WVC option to input the various relative weights of descendents on a given span, and to input the values of the data nodes. After entering the appropriate weights and values, the tree should then be calculated in order to determine the relative rankings of the alternatives under consideration.

After exiting the WVC option, the user will more than likely want to examine the results. The program has two main methods to accomplish this. These are options DIS and NUM.

Option NUM gives a numerical review of some or all nodes in the data structure. Along with the relative weights and cumulative weights of a given node (and node reference number), the values of the alternatives (either entered or calculated) will be presented. The review can be routed to either the printer or the console (monitor).

Option DIS gives a review, but only for a specified node and that node has to have descendents. Information given in option DIS is similar to that in option NUM; however, option DIS gives the added option in providing a graphical display of the node and its descendents.

One of the most important aspects of the DASS system is its capability to provide sensitivity analysis. The sensitivity analyses that are available with this version of DASS allow the variation of the cumulative weight for a particular node, the variation of the relative weight for a particular node with relation to its siblings, and the variation of the value of an alternative on a data node. It is through sensitivity analysis that portions of the tree structure can be singled out for further investigation. In addition, care must be taken in performing sensitivity analysis on the cumulative weights due to the nature of the heirarchy. That is, all parts are linked together. Changes in one area may impact assigned values in appearently unrelated areas.

Another option that is available to the user is STA. This option provides current data concerning the tree structure which is necessary to insure against exceeding the capacity of the various data files generated by this program.

The final option, and the most important, is the DON command which allows the user to elegantly exit the program. This program sequentially closes all open files and returns control to the disk system.

Deleting old files

Current limitations of the DASS system, only allows for three data sets to exist on a single disk (see LIMITATIONS). Thus the time will come that the user will want to delete old files to make room for new ones. In addition, given a very long length of time, the user may forget

the title of some files and would then have to resort to getting a new disk or re-entering the data files. The following gives a very brief but complete discussion as to how to remove or get the names of old files.

When in the Disk Operating System (DOS) mode, depress the F key. You will be in the DOS mode when you just turn on power to the machine with the DASS disk in the disk drive, or after exiting the DASS program via the DON option. This will invoke the FILER program which accomplishes all disk operations. When executed, the console will display a line of characters which are various options available to the FILER program.

The first option that can be executed is done by typing E. This invokes the directory list command which will display all diskfiles that are on the disk. After pressing the E key, the computer will respond with DIR LISTING OF WHAT VOL ?. At this time enter an asterisk (*) and depress the RETURN key. The results will be similar to that shown in Figure 2.

The first column indicates the names of files currently on the disk. Those files which end with SYS.DATA or NODE.DATA represent the data sets for various tree structures which begin with the first three letters preceding SYS.DATA or NODE.DATA. If the user were using file names when executing the DASS program less than three letters, the letters X will be used to fill in the space between the user entered filename and three characters.

In order to remove a data set, depress the letter R. The computer

will respond with REMOVE?. The user should then enter the file name that is to be removed (e.g. ALPSYS.DATA) and conclude by depressing the RETURN key. The computer will then respond with a verification message and the user should respond by depressing Y for yes or anything else for no. In either

```
LEE3:
SYSTEM.APPLE          32 26-Jul-79     6 Data
SYSTEM.PASCAL         36  4-May-79    38 Data
SYSTEM.MISCINFO        1  4-May-79    74 Data
SYSTEM.FILER          28 24-May-79    75 Code
SYSTEM.LIBRARY        46 10-Nov-80   103 Data
SYSTEM.CHARSET         2 14-Jun-79   149 Data
DASS.CODE             56  7-Jan-81   151 Code
ECHSYS.DATA            1 27-Oct-80   207 Data
ECHNODE.DATA          23  7-Jan-81   208 Data
ALPSYS.DATA            1  7-Jan-81   231 Data
ALPNODE.DATA          23  7-Jan-81   232 Data
BETSYS.DATA            1  7-Jan-81   255 Data
BETNODE.DATA          23  7-Jan-81   256 Data
< UNUSED >             1             279
13/13 files, 1 unused, 1 in largest
```

Figure 2. Directory Display

case, control will be returned to the FILER program. Note that if data sets are to be removed, both the SYS.DATA and the NODE.DATA must be removed (e.g. ALPSYS.DATA and ALPNODE.DATA). IN ADDITION, the user must execute the K option (compress the disk) in order to make room for subsequent files (the file structure for this computer dictates that an unused disk space bigger than the file size be available in order to be able to create a new files under DASS). The user can do this by

depressing the K key and respond to the subsequent question with an asterisk (*) and depressing the RETURN key. Another question will be asked as to whether all blocks are to be compressed and the answer is Yes. If the user inadvertently depresses another key, other messages will occur. The user at this time is urged to simply depress the RETURN key and return to the FILER menu.

After accomplishing the desired removal operations, the user can exit the FILER program by depressing the Q key which will return the user to the DOS menu. The user can then execute the DASS program by the methods described previously.

IV THE OPTIONS

The following sections deal with the major options and their sub-options from the user's standpoint and in explicit detail. The options are discussed in alphabetical order, each beginning on a new page. All options are discussed using the format shown in Figure 3.

```
OPTION: (option mnemonic)(option name)
USE: A general discussion of the use of the option

          ********** C A U T I O N S **********
            CAUTIONS ASSOCIATED WITH USE OF THIS OPTION
          ********** C A U T I O N S **********

PROMPT or MESSAGE  prompts and messages are displayed by the
                   the computer which may require some action
                   on the part of the user.
SITUATION            on the part of the user. This action attempts
                   to completelly describe the meaning of
                   the prompt and the various effects due to the prompt
REQ ACTION           specifies action(s)
                   required by the user in order to satisfy
                   program requirements.
```

Figure 3. Option Discussion Format

THE MAIN MENU

PROMPT 1: ATT DIS DON MOD NEW NUM PRU REV SEL
SEN SPA STA SYS TTL WVC
OPTION?

SITUATION: After the program has been initialised (i.e. executing option SEL), the main menu will be displayed indicating all options available under the current version of DASS.

REQ ACTION: Enter the three letter option and the RETURN key. The computer will respond with prompts associated with that option. Entering anything else will result in PROMPT 1 being repeated.

OPTION: ATT (ATTribute)

USE: This option is used to enter a label for the decision which is being used. Reasonable responses are 'VALUE' (more is better) and 'REGRET' (less is better). The response is later used to prompt the user for the values for the alternatives being considered and indicating which alternative is the best (highest or lowest) in the sensitivity analysis.

********** C A U T I O N **********

1. ANY ENTRY STARTING WITH THE LETTER R WILL BE TREATED AS REGRET (LESS IS BETTER) IN THE SENSITIVITY ANALYSIS. ALL CHARACTERS STARTING WITH ANY OTHER LETTER/NUMERAL/ OR CHARACTER WILL BE TREATED A VALUE (MORE IS BETTER) IN THE SENSITIVITY ANALYSIS.

********** C A U T I O N **********

PROMPT 1: ENTER ATTIRBUTE CHARACTERISTIC (REGRET OR VALUE)?

REQ ACTION: Enter attribute characteristic, ending with the RETURN key. The entry can be any length; however, only the first ten characters will be retained.

OPTION: DIS (DISplay a node with descendents)

USE: This option will display the information at a node in either tabular or graphical format.

********** C A U T I O N S **********

1. DIRECTING THE DISPLAY TO GO TO THE PRINTER WHEN THE PRINTER IS NOT AVAILABLE WILL RESULT IN A PROGRAM FAILURE.

********** C A U T I O N S **********

PROMPT 1: ENTER NODE TO BE DISPLAYED...
ENTER...NRN?

SITUATION: The user is asked to enter the node reference number, for a node which has descendents, and for which the display is to take place.

ACTION REQ: Enter the NRN and conclude by depressing the return key. Entering a NRN which is non-existent will return control back to the MAIN MENU.

PROMPT 1-A: NODE IS A DATA NODE AND CANNOT BE DISPLAYED
DEPRESS RETURN TO CONTINUE

SITUATION: The NRN entered in response to PROMPT 1 is not a node with descendents.

ACTION REQ: Depress the RETURN key to continue the program.

PROMPT 2: WARNING...
TREE NEEDS TO BE (RE)CALULATED
OUTPUT VALUES MAY BE INCORRECT

PRESS ANY KEY TO CONTINUE

SITUATION: The tree structure has been modified by either adding or deleting nodes, or adding or deleting alternatives and has not been re-computed since. This means that the weights and values for some or all nodes may be inaccurate. The display, however, will not be inhibited nor can it be aborted.

ACTION REQ: Depress any key.

PROMPT 3: T(ABULAR G(RAPHIC

SITUATION: The user is now asked as to whether a tabular or graphical presentation of the data is desired. The tabular format is shown in Figure 4, and shows the node entered in PROMPT 1 as well as its immediate descendents. In addition, numeric values of each alternative for all nodes are listed as well as the cumulative and relative weights.

The graphical format provides similar information; however, the display takes on the appearance illustrated in Figure 4 with the numeric values for the relative and cumulative weights being displayed and the values of the alternatives, in bar graph format, being displayed underneath the descendent nodes and to the right of the parent node. An alternative/color legend is displayed on the upper left hand corner.

ACTION REQ: Press the T or G key for the option desired. If T is depressed, PROMPT 4 will occur.

If G is depressed, Figure 5 will be displayed. Depressing any other key will result in PROMPT 3 being repeated.

PROMPT 4: C(ONSOLE P(RINTER

SITUAION: This prompt will occur if the T key was depressed in response to PROMPT 3. This prompt determines if the review is to sent to the console or the printer.

ACTION REQ: Depress the C or P key depending on the option desired. The output will appear as shown in Figure 4. If the P key is depressed, control is returned to the MAIN MENU after the data is sent to the printer. Depressing any other key will result in PROMPT 4 being repeated.

PROMPT 5: PRESS ANY KEY TO CONTINUE

SITUATION: This prompt occurs after the tabular display is produced on the console. This causes a temporary pause so that the the user can examine the data.

ACTION REQ: After examining the display, press any key. Control will be returned to the MAIN MENU.

```
1 3
COMBAT     -
  FACTOR    F-4    F-15   F-111
AIR-AIR *   50.00  90.00  20.00
AIR-GND *   50.00  20.00  40.00
RECCE   *   50.00  10.00  40.00
   TOTAL    50.00  48.57  34.28

  FACTOR    (WT)  CUMWT
AIR-AIR *  (.42)  0.24
AIR-GND *  (.42)  0.24
RECCE   *  (.14)   .08
   TOTAL          0.57
```

The above format will be seen if T is depressed in response to PROMPT 3

Line 1 (1 3) the NRN for the node entered in PROMPT 1

Line 2 (COMBAT -) the label for the node entered in PROMPT 1

Line 3 (FACTOR F-4 F-15 F-111) lists the header for all immediate descendent nodes and lists horizontally the alternative labels truncated to five characters.

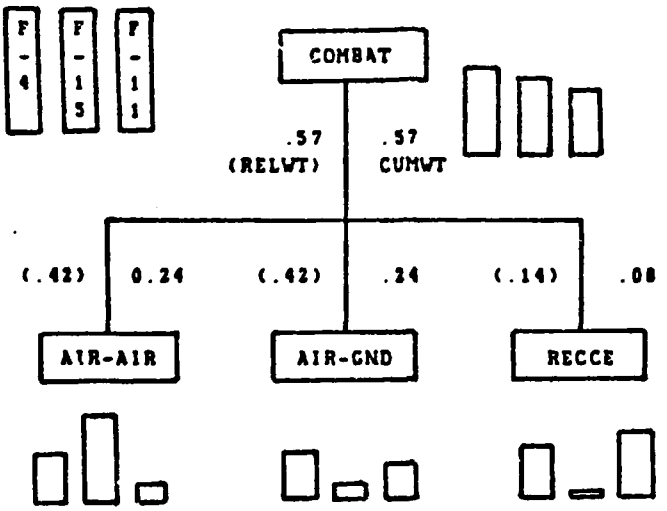
Lines 4-6 lists the labels of the immediate descendent nodes and lists values (inputted or calculated) for each alternative. The asterisk (*) listed to the right of the label indicates the node as a data node (the bottom-most node for that branch).

Line 7 (TOTAL 50.00 48.57 34.28) lists the value of each altrnative for the node inputted in response to PROMPT 1.

Line 8 (FACTOR (WT) CUMWT) lists the header, again for the immediate descendents. The (WT) is the relative weight of each sibling (lines 9-11) against each other. The Cum-T is the contribution of each sibling (lines 9-11) to the overall decision structure.

Line 12 (TOTAL 0.57) indicates the amount (relative to 1.00) that the node inputted in response to PROMPT 1 contributes to the overall decision structure.

Figure 4 DIS Tabular Format



The above format will be seen if G is depresed in response to PROMPT 3.

The parent node is listed at the top with the values calculated on its right. For each descendent node, its set of values are listed in bar graph format underneath. Nodes labeled with asterisks (*) are data nodes. Each bar is colored uniquely to a particular alternative legend in the upper left hand corner.

(REL WT) values in parentheses are values of each sibling against each other. For COMBAT (the parent node), its relative weight is in relation to other nodes (not shown in the display) at the parent's level.

CUMWT indicates the overall contribution of the node to the overall decision structure.

Figure 5 DIS Graphical Display

OPTION: DON (DONe)

USE: This is the last option used. This signals that the user is finished with the program and the data files. The program will close (save) the files including appropriate information of the tree structure (number or nodes, number of levels, number of alternatives, and tree accuracy). The program will terminate and the DISC OPERATING SYSTEM menu will be displayed.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

PROMPTS: none

OPTION: MOD (MODify existing tree structure)

USE: This option is used to modify the tree one node at a time. Each node addressed by node reference number (NRN) is: (1) created, if necessary, along with its predecessors as necessary, (2) changed in label only if it already exists.

********** C A U T I O N S **********

1. ALL NRNs ENTERED MUST BEGIN WITH 1. IF ANY OTHER NUMBER IS ENCOUNTERED FIRST, THE TREE STRUCTURE (AS KNOWN BEFORE THIS MODIFICATION) WILL BE PERMANENTLY LOST.

2. ADDING MORE THAN FIVE NODES ON A SPAN IS POSSIBLE IF ONE OF THE SPANS CONTAINED IN THE NRN ALREADY HAS FIVE NODES. THE RESULTS OF THIS ARE UNPREDICTABLE AND MAY PRODUCE UNEXPECTED OR UNDESIRABLE RESULTS.

3. ANY NUMBER CAN BE USED IN ENTERING THE NRN REQUESTED BY PROMPT 1; HOWEVER, IF A GIVEN SPAN DOES NOT EXIST FOR A GIVEN NRN LEVEL, THE NRN NUMBER 1 WILL BE ASSIGNED FOR THAT LEVEL.

4. THERE ARE NO CHECKS TO GUARD AGAINST THE OVERALL SIZE REQUIREMENTS OF THE TREE STRUCTURE GIVEN EARLIER IN THIS TEXT. THE USER IS URGED TO USE OPTION STA AT THE CONCLUSION OF THIS OPTION TO INSURE AGAINST INADVERTENT SIZE VIOLATIONS.

********** C A U T I O N S **********

PROMPT 1: ENTER...NRN?

SITUATION: The user is asked to enter the node reference number where the additional node is to be added, or the label of the node is to be changed.

ACTION REQ: Enter the NRN and conclude by depressing the RETURN key. Depressing the RETURN key without any entry will result in control being returned to the MAIN MENU.

PROMPT 2: LABEL?

SITUATION: The user is asked to enter the label of the node(s) directed by PROMPT 1. If the node already exists, just the label of that node will be changed. If the node directed by the NRN does not exist, all new nodes created by the NRN entered in PROMPT 1 will contain the entered label.

ACTION REQ: Enter the label of the node(s) and conclude by depressing the RETURN key. The node label may be of any length; however, only the first ten (10) characters will be retained. Entering DONE and depressing the RETURN key or depressing the RETURN key without any entry will abort this option. Control will be returned to the MAIN MENU. If a label was inputted in response to PROMPT 2, PROMPT 1 will be repeated.

OPTION: NEW (NEW data file)

USE: This option is used to generate a new tree structure, either with th present file (if file name is identical) or a new file name. If the file name request by the prompts is identical to the file currently in use, the file will be destroyed.

SEE OPTION SEL FOR THE PROMPTS AND REQUIRED ACTIONS

OPTION: NUM (NUMeric review)

USE: This option is used to generate a display of the tree structure, including the node reference number, node label, relative weight, cumulate weight, input values or calculated values for each node and alternative.

********** C A U T I O N **********

1. IF THE CTRL AND A KEY ARE DEPRESSED SIMULTANEOUSLY, THE OTHER HALF OF THE SCREEN WILL BE DISPLAYED. WHILE NOT CATASTROPHIC, A BLANK SCREEN MAY OCCUR WHERE DATA OR MESSAGES OUGHT TO BE DISPLAYED. IF IN DOUBT, DEPRESS THE CTRL AND A KEY SIMULTANEOUSLY.

2. DIRECTING THE NUMERIC REVIEW TO GO TO THE PRINTER WHEN THE THE PRINTER IS NOT AVAILABLE WILL RESULT IN A PROGRAM FAILURE.

********** C A U T I O N **********

PROMPT 1: A(LL S(ELECT

SITUATION: This prompt determines if the entire tree is to be reviewed (ALL) or if only a selected node is to be reviewed (SELECT)

ACTION REQ: Depress the A or S key for the option desired. If the the A key is depressed, the program will respond with PROMPT 2. If the S key is depressed, PROMPT 1-A will appear. Any other key will result in PROMPT 1 being repeated.

PROMPT 1-A: ENTER...NRN?

SITUATION: The S key has been depressed in response to PROMPT 1 and the user is now asked to input the node reference number (NRN) for which the numeric review is to take place.

ACTION REQ: Enter the node reference number and conclude with the RETURN key.

PROMPT 2: C(ONSOLE P(RINTER

SITUATION: This prompt determines if the review is to sent to the console or the printer.

ACTION REQ: Depress the C or P key depending on the option desired. If C is depressed, the output will appear as in Figure 6. If P is depressed, the output will appear as in Figure 7.

PROMPT 3: WARNING...
TREE NEEDS TO BE (RE)CALCULATED
OUTPUT VALUES MAY BE INCORRECT

PRESS ANY KEY TO CONTINUE

SITUATION: The tree structure has been modified by either adding or deleting nodes; adding or deleting alternatives; or changing node weights or values and has not been calculated since. This means that the weights and values for some or all nodes may be inaccurate. The review, however, is not inhibited.

ACTION REQ: Depress any key

PROMPT 4: <ANY KEY> CONTINUE <ESC> EXIT

SITUATION: When a full page of output is produced, either on the printer or the console, the output is paused so that the user can examine the data. Pressing any key will continue the output. Pressing the ESC key will abort the numeric review and return control to the MAIN MENU.

ACTION REQ: Depress any key or the ESC key depending on the action required.

PROMPT 5: PRESS ANY KEY TO CONTINUE

SITUATION: Display is completed. This is written in order to allow the user time to peruse the last page of the numeric review.

ACTION REQ: Press any key to continue the program. The program will return control to the MAIN MENU.

R E V I E W

BEST AIRCRAFT

NODE	REF NUMBER	LABEL	REL WT	CUMWT	F-4	F-15	F-111
1		BEST PLANE	1.00	1.00	47.79	55.04	37.81
1	1	AERO	.14	.14	34.50	31.00	47.50
1	1 1	HIGH ALT	.30	.04	45.00	80.00	30.00
1	1 2	LOW ALT	.70	.10	30.00	10.00	55.00
1	2	SURVIVABLE	.29	.29	50.00	80.00	40.00
1	3	COMBAT	.57	.57	50.00	48.57	34.29
1	3 1	AIR-AIR	.43	.24	50.00	90.00	20.00
1	3 2	AIR-GND	.43	.24	50.00	20.00	40.00
1	3 3	RECCE	.14	.08	50.00	10.00	60.00

The above format will be seen in P is depressed in response to PROMPT 2. Each line in the table contains the following information (the third line is used as an example).

1) NODE REF NUMBER LABEL (1 1 1 HIGH ALT). The node reference number (NRN) is given for each node, along with that node's label.

2) REL WT (.30) The wieght of this node, relative to its siblings (1 1 2 LOW ALT) is given.

3) CUM WT (.04) The cumulative weight of this node relative to the entire tree (its contribution to the top node calculated values) is given.

4) ALTERNATIVE LABELS- F-4 F-15 F-111 (45.00 80.00 30.00). The values for the alternatives are given. These are either the inputs values (if the node is a data node) or calculated values (if the node is an inner, branching node). These calculated values are those which the user would enter if the branching node were to be replaced with a data node, and the values entered as user inputs.

Figure 6. NUM Format for the Line Printer

R E V I E W

BEST AIRCRAFT

NODE	REF NUMBER	LABEL	REL WT	CUMWT		F-4	F-15	F-111
1		BEST PLANE	1.00	1.00	BEST PLANE	47.79	55.04	37.81
1	1	AERO	.14	.14	AERO	34.50	31.00	47.50
1	1 1	HIGH ALT	.30	.04	HIGH ALT	45.00	80.00	30.00
1	1 2	LOW ALT	.70	.10	LOW ALT	30.00	10.00	55.00
1	2	SURVIVABLE	.29	.29	SURVIVABLE	50.00	80.00	40.00
1	3	COMBAT	.57	.57	COMBAT	50.00	48.57	34.29
1	3 1	AIR-AIR	.43	.24	AIR-AIR	50.00	90.00	20.00
1	3 2	AIR-GND	.43	.24	AIR-GND	50.00	20.00	40.00
1	3 3	RECCE	.14	.08	RECCE	50.00	10.00	60.00

The data on the left side of the dashed line will be seen if the C key is depressed in response to PROMPT 2. In order to see the data on the right side, depress the CTRL and A key at the same time. Depressing the CTRL and A key again will return the display back to the left side.

See Figure 6 for further information.

Figure 7. NUM Format for the Console

OPTION: PRU (PRUne tree)

USE: Prune is used to eliminate selected branches from the tree. The user is given two options for pruning: (1) prune a node and all its descendents, or (2) prune only the descendents of a node.

****************** C A U T I O N S ********************

1. ENTERING AN INCORRECT OR NON-EXISTANT NODE REFERENCE NUMBER CAN HAVE CATASTROPHIC EFFECTS ON THE TREE STRUCTURE. USE OPTION REV OR NUM TO INSURE PROPER NRN ENTRY.

2. A FORM OF TEMPORARY PRUNING, WHICH DOES NOT ALTER THE TREE STRUCTURE IS TO USE WEIGHTS OF 0.0 FOR THOSE BRANCHES WHICH WOULD BE PRUNED. USE OPTION WVC.

****************** C A U T I O N S ********************

PROMPT 1: N(ODE+DOWN D(OWN ONLY E(XIT

SITUATION: The user is asked to selected the particular method of pruning the tree structure.

NODE+DOWN prunes the node entered in response to PROMPT 2 as well as all descendent nodes.

DOWN ONLY prunes only the descendents of the node entered in response to PROMPT 2.

EXIT exits the option and returns the control to the MAIN MENU.

ACTION REQ: Depress the N, D, or E key for the desired option.

Depressing the N or D key will result in PROMPT 2 and on to be displayed. Depressing the E key will result in the exit of this procedure and return to the MAIN MENU. Depressing any other key will result in PROMPT 1 being repeated.

PROMPT 2: ENTER...NRN?

SITUATION: The user is now asked to enter the node reference number for the node on which the pruning is to take place.

ACTION REQ: Enter the NRN and conclude with the RETURN key.

PROMPT 2-A: YOU CANNOT PRUNE A DATA NODE WITH
A DOWN ONLY OPTION. . .
PRUNE TERMINATED . . .
<ANY KEY> CONTINUE

SITUATION: The NRN entered in response to PROMPT 2 was a data node and the D key was depressed in response to PROMPT 1.

ACTION REQ: Depress any key. The program will return control to the MAIN MENU.

MESSAGE 1: PRUNING...

SITUATION: This message will indicate that various pruning operations are underway. Additional periods (.) will be displayed as more pruning operations are initiated and completed. At the end of the pruning, control of the program will

automatically be transferred to the MAIN MENU.

OPTION: REV (REView of structure)

USE: This option is used to generate a display of the tree stru ure. The node reference number and node label are given for each node reviewed, which will be the user selected node and all its descendents.

********** C A U T I O N **********

1. DIRECTING THE REVIEW TO GO TO THE PRINTER WHEN THE THE PRINTER IS NOT AVAILABLE WILL RESULT IN A PROGRAM FAILURE.

********** C A U T I O N **********

PROMPT 1: A(LL S(ELECT

SITUATION: This prompt determines if the entire tree is to be reviewed (ALL) or if only a selected node is to be reviewed (SELECT)

ACTION REQ: Depress the A or S key for the option desired. If the the A key is depressed, the program will respond with PROMPT 2. If the S key is depressed, PROMPT 1-A will appear. Any other key will result in PROMPT 1 being repeated.

PROMPT 1-A: ENTER...NRN?

SITUATION: The S key has been depressed in response to PROMPT 1 and the user is now asked to input the node reference number (NRN) for which the numeric review is to take place.

ACTION REQ: Enter the node reference number and conclude with the RETURN key.

PROMPT 2: C(ONSOLE P(RINTER

SITUATION: This prompt determines if the review is to be sent to the console or the printer.

ACTION REQ: Depress the C or P key depending on the option desired. The output will appear as in Figure 8.

PROMPT 3:
```
WARNING...
TREE NEEDS TO BE (RE)CALCULATED
OUTPUT VALUES MAY BE INCORRECT

PRESS ANY KEY TO CONTINUE
```

SITUATION: The tree structure has been modified by either adding or deleting nodes; adding or deleting alternatives; or changing weights and values of nodes and has not been calculated since. This means that the weights and values for some or all nodes may be inaccurate. The review, however, is not inhibited.

ACTION REQ: Depress any key

PROMPT 4: (ANY KEY) CONTINUE (ESC) EXIT

SITUATION: When a full page of output is produced, either on the printer or the console, the output is paused so that the user can examine the data. Pressing any key will continue the output. Pressing the ESC key will abort

the review and return control to the MAIN MENU.

ACTION REQ: Depress any key or the ESC key depending on the action required.

PROMPT 5: PRESS ANY KEY TO CONTINUE

SITUATION: Display is completed. This is written in order to allow the user time to peruse the last page of the review.

ACTION REQ: Press any key to continue the program. The program will return control to the MAIN MENU.

```
1 BEST PLANE
1 1 AERO
1 1 1 HIGH ALT
1 1 2 LOW ALT
1 2 SURVIVABLE
1 3 COMBAT
1 3 1 AIR-AIR
1 3 2 AIR-GND
1 3 3 RECCE
```

Figure 8. REV Format for the Console or Line Printer

OPTION: SEL (SELect data file)

USE: This option is used to select a new data file

********** C A U T I O N **********

1. ENSURE DISK NAME ENTRIES CONCLUDE WITH A COLON (:)

2. IF CREATING A NEW FILE AND THE FILENAME ENTERED IS THE SAME AS ONE THAT ALREADY EXISTS, THE OLD DISK FILE WILL BE DESTROYED.

3. IF THE FILE ASKED FOR IN PROMPT 1 AND 2 DOES NOT EXIST, A RESULTING COMPUTER ERROR WILL OCCUR. YOU MUST THEN DEPRESS THE SPACE BAR AND RE-EXECUTE THE DASS PROGRAM.

********** C A U T I O N **********

PROMPT 1: ENTER...DISK NAME TO BE USED
(E.G. APPLE0.)

SITUATION: Computer data files for the DASS program are catalogued by a disk name (volume name) and file name. The computer at this point is asking for a disk name.

ACTION REQ: Enter the name of the disk Conclude entry with a colon (:) and the RETURN key. If the file is to be on the same disk with the DASS program, the RETURN key by itself will suffice.

PROMPT 2: PLEASE INSERT (DISK NAME) AND DEPRESS RETURN

SITUATION: The computer is asking that the user insures that the dick specified in PROMPT 1 is in the disk drive.

ACTION REQ: Make sure that the disk specified in PROMPT 1 is in the

disk drive, then depress the RETURN key.

PROMPT 3: ENTER...FILE NAME TO BE ACCESSED

SITUATION: The computer is asking for the particular file name.

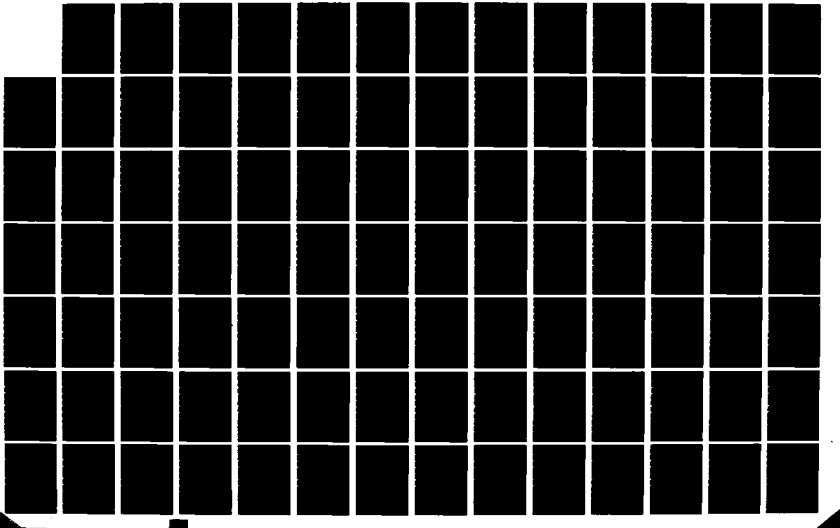
ACTION REQ: Enter the file name of the desired data, conclude with the RETURN key. The file name can consist of alphanumerics (letters or numerical digits) and the special characters minus sign (-), right slash (/), back slash (\), and underline (_). The file name can be any length; however, only the first three characters will be significant and used by the program.

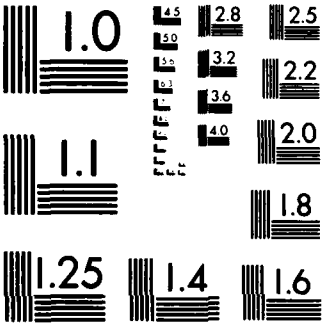
PROMPT 4: (CR) OR NEW?

SITUATION: Asking if the file already exists. The file exists if the DASS program has been stopped by the DON or SEL option after entering the file name entered in PROMPTs 1 and 2.

ACTION REQ: If the file described in PROMPT 1 is to be a new file, type NEW and conclude with the RETURN key. The program will then execute options ATT, TTL, and SPA. If the file described in PROMPT 1 and 2 is an old file, press the RETURN key. The program will display the MAIN MENU.

AD-A101 140 ENHANCED DECISION ANALYSIS SUPPORT SYSTEM(U) AIR FORCE 2/4
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING D B LEE MAR 81 AFIT/GST/OS/81M-8
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

OPTION: SEN (SENsitivity analysis)

USE: Used to conduct sensitivity analyses. The selected node's cumulative weight, relative weight, or in the case of a data node, the value of an alternative, is allowed to range between a user selected minimum and maximum values. The tree is collapsed with the perturbed appropriate weight (or value), and the values of the top node are printed or plotted.

********** C A U T I O N **********

1. USING MIN AND MAX VALUES FAR FROM THE 'CORRECT' VALUE WILL GIVE MATHEMATICALLY CORRECT NUMBERS, HOWEVER THE VALIDITY OF THESE RESULTS MAY BE SUSPECT DUE TO THE SYNERGISTIC INTERACTIONS OF WEIGHTS WITH WEIGHTS AND WEIGHTS WITH VALUES.

********** C A U T I O N **********

MESSAGE 0: WARNING . . .
TREE NEEDS TO BE (RE)CALCULATED
REPORTED VALUES MAY BE INCORRECT!
(ANY KEY) CONTINUE

SITUATION: The tree structure has been modified by either adding or deleting nodes; adding or deleting alternatives; or changing node weights and values. The sensitivity analysis will not be inhibited; however, values calculated will not be valid.

ACTION REQ: Press any key to continue the program and respond to PROMPT 1 by pressing the E key.

PROMPT 1: SENSITIVITY: C)UMWT R)ELWT V)ALUE E)XIT

SITUATION: This prompt is to ascertain the type of sensitivity analysis that is to be accomplished.

CUMWT - vary the cumulative weight of a node.

RELWT - vary the relative weight of a node.

VALUE - vary the value of an alternative at a data node.

EXIT - exit sensitivity

ACTION REQ: Press the C, R, V, or E key depending on the option desired. Pressing any other key will result in PROMPT 1 being repeated.

MESSAGE 1: SENSITIVITY ANALYSIS FOLLOWS:

PROMPT 2: NRN FOR WHICH <OPTION> IS
TO BE PRETURBED
ENTER...NRN?

SITUATION: <OPTION> indicates the option selected by the user in PROMPT 1 The computer is asking for the Node Reference Number (NRN) for the node where the sensitivity analysis is to take place.

ACTION REQ: Enter the node reference number (NRN) for the node where the sensitivity is to be accomplished. Any character not a number will be ignored. End the NRN entry with a carriage return (RETURN key).

PROMPT 2A: NODE DOES NOT EXIST...
(ANY KEY) CONTINUE

SITUATION: The NRN entered by the user does not exist in the present tree structure.

ACTION REQ: Press any key to continue program. The computer will respond with PROMPT 2.

PROMPT 2B: NODE NOT DATA NODE...
(ANY KEY) CONTINUE

SITUATION: Message occurs when the user has selected the VALUE option and the NRN selected by the user is not a data node (i.e. the bottom-most node for that branch).

ACTION REQ: Press any key to continue program. The computer will respond with PROMPT 2.

PROMPT 3: SYSTEMS AVAILABLE
(LIST OF ALTERNATIVES)

ENTER SYSTEM FOR WHICH VALUE IS
TO BE PRETURBED. . .

SITUATION: (LIST OF ALTERNATIVES) Alternatives that are contained in the present tree structure. This prompt occurs only when the VALUE option is requested.

ACTION REQ: Enter one of the alternatives listed in the (LIST OF ALTERNATIVES) ending with a carriage return (RETURN key).

PROMPT 3A: SYSTEM ENTERED NOT VALID
(ANY KEY) CONTINUE

SITUATION: The system entered by the user in response to PROMPT 3 was not the same are listed in the (LIST OF ALTERNATIVES) (see PROMPT 3) above.

ACTION REQ: Press any key to continue the program. The computer will repeat with PROMPT 3.

MESSAGE 2: (NRN) (NODELABEL)
CURRENT NODE (OPTION) IS (VALUE)

SITUATION: (NRN) Node referenced entered by user in PROMPT 2

(NODE LABEL) The title of the node identified by the node reference number in PROMPT 2

(OPTION) Will be either CUMWT, RELWT or VALUE dependent upon the choice made in PROMPT 1.

(VALUE) The present value of the cumulative weight, the relative weight, or the value of the alternative for the node identified in the NRN in PROMPT 2.

PROMPT 4: MINIMUM (OPTION) (0-(MAX VALUE ALLOWED)) IS?

SITUATION: (OPTION) Will be either CUMWT, RELWT or VALUE depending upon the choice made in

PROMPT 1.

(MAX VALUE ALLOWED) 1 for options CUMWT or RELWT

100 for option VALUE

ACTION REQ: Enter a value which will be used as the lower limit in the sensitivity analysis for the option selected by PROMPT 1. Conclude the entry with a carriage return (RETURN key). Entered values below zero or higher than the (MAX VALUE ALLOWED) will result in PROMPT 4 being repeated.

PROMPT 5: MAXIMUM (OPTION) ((MIN VALUE) - (MAX VALUE ALLOWED)) IS?

SITUATION: (OPTION) Option selected by PROMPT 1

(MIN VALUE) User input value in response to PROMPT 4

(MAX VALUE ALLOWED) Same as PROMPT 4

ACTION REQ: The user is to enter a value which will be used as the upper limit in the sensitivity analysis for the option selected by PROMPT 1. Conclude the entry with a carriage return (RETURN key). Entered values below (MIN VALUE) or higher than the (MAX VALUE ALLOWED) will result in PROMPT 5 being repeated.

PROMPT 6: (OPTION): T)ABULAR G)RAPHICAL E)XIT

SITUATION: (OPTION) Either CUMWT, RELWT, or VALUE based on the option selected in PROMPT 1

The sensitivity analysis is completed. The computer

is now asking the user to select the form in which the analysis is to be displayed.

TABULAR - Tabulated values for each system, at the top node, for the two extremes and for 9 equally spaced points internal to the interval (Figure 9).

GRAPHICAL - The graphical representation of the analysis with the two extremes of the sensitivity interval along the Y-axis and the values for each system, at the top node, along the X-axis. Each alternative is color coded at the top of the display (Figure 10).

EXIT - Leave the senstivity analysis routine.

ACTION REQ: User is to press the T, G, or E key to display the analysis in the desired node. Pressing any other key will result in PROMPT 6 being repeated.

PROMPT 6A: TABULAR: C(ONSOLE P(RINTER

SITUATION: This prompt will be in response to the T key being depressed in PROMPT 6. The computer is now asking as to whether the sensitivity analysis is to go to the console or the line printer.

ACTION REQ: Press the C or P key to send the sensitivity analysis output to either the console or the line printer, respectively. Pressing any other

key will result in PROMPT 6 being repeated.

PROMPT 6B: GRAPHIC: N(ORMAL E(XPANDED

SITUATION: This prompt will be in response to the G key being depressed in PROMPT 6. The computer has accepted the command to provide a graphical display of the sensitivity analysis and is now asking for the format the graph is to take.

NORMAL - the X-axis will be labelled from 0 to 100 which is the minimum and maximum values which the top node can take.

EXPANDED - the X-axis will be labelled from the lowest value obtained in the sensitivity analysis (to the nearest 20) to the highest value (to the nearest 20). This option is normally used to separate alternative lines for clarity.

ACTION REQ: Press the N or E key based on the graphical display desired. Pressing any other key will result in PROMPT 6B being repeated. Upon completion of the graphical display, depressing any key will result in PROMPT 6 being repeated.

PROMPT 7: (ANY KEY) CONTINUE

SITUATION: This occurs at the conclusion of the tabular printout at either the line printer or the console.

ACTION REQ: Press any key to continue the program. The program will

respond with PROMPT 6.

CUMWT SENSITIVITY ANALYSIS
FOR SURVIVABLE NRN: 1 2

CUMWT:	F-4	F-15	F-111
0.00	46.9*	45.1	36.9
0.10	47.2	48.6*	37.2
0.20	47.5	52.0*	37.5
0.30	47.8	55.5*	37.8
0.40	48.1	59.0*	38.2
0.50	48.5	62.5*	38.5
0.60	48.8	66.0*	38.8
0.70	49.1	69.5*	39.1
0.80	49.4	73.0*	39.4
0.90	49.7	76.5*	39.7
1.00	50.0	80.0*	40.0

Figure 9 SEN Tabular Display

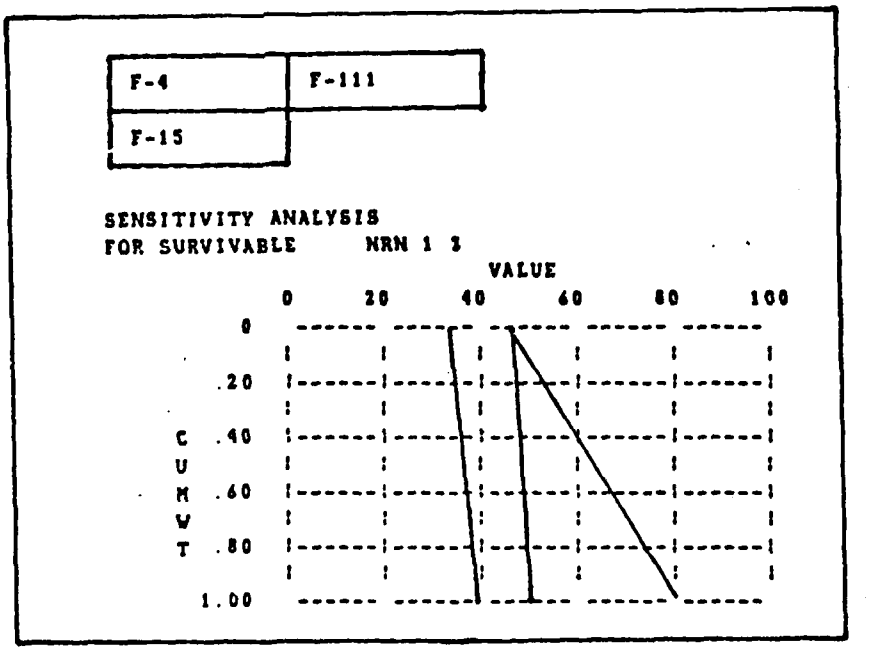


Figure 10 SEN Graphical Display

OPTION: SPA (SPAn on a given node)

USE: This option is used to create new tree structures by entering the immediate descendents (children) to nodes. The user enters the node reference number (NRN) for the top node to be processed, and NRNs are automatically generated for all additional nodes as the user enters labels for the new nodes. The user is given the option of creating descendents for each new node created.

********** C A U T I O N S **********

1. ALL OLD DESCENDENTS AT A NODE ARE LOST IF EVEN ONE NEW DESCENDENT IS ADDED. NOTE THAT IT IS POSSIBLE TO EXIT FROM A NEWLY PROMPTED NODE BY ENTERING DONE OR EXIT OR DEPRESSING THE RETURN KEY.

2. A YES RESPONSE TO PROMPT 1-A WILL DESTROY ALL PREVIOUS TREE STRUCTURE DATA.

3. A MAXIMUM OF 100 NODES CAN BE USED. THE PROGRAM WILL NOT CHECK IF MORE ARE ADDED. UNEXPECTED RESULTS MAY OCCUR.

4. SPANNING A NODE WITH DESCENDENTS WILL NOT ONLY RESULTS IN ALL THE DESCENDENTS BEING DESTROYED, BUT ALSO REDUCES THE TOTAL NUMBER OF NODES AVAILABLE FOR THE TREE STRUCTURE. IN OTHER WORDS, THE TREE STRUCTURE WILL ONLY BE ABLE TO CONTAIN SOME NUMBER OF NODES LESS THAN 100 IN IT. IF SPANNING ON A NODE THAT HAS DESCENDENTS, PRUNE (USING THE PRU OPTION) THE DESCENDENTS BEFORE SPANNING. THIS WILL INSURE THAT ALL 100 NODES ARE AVAILABLE IN THE STRUCTURE.

********** C A U T I O N S **********

PROMPT 1: SPAN NODES . . . A(LL S(ELECT

SITUATION: This prompt determines where a new structure is to be constructed (ALL) or new descendents are to be defined for a specific node (SELECT).

ACTION REQ: Depress the A or S key for the option desired. If A(LL is selected, PROMPT 1-A will occur, else PROMPT 1-B. Depressing any other key will result in PROMPT 1 being repeated.

PROMPT 1-A: DO YOU WANT TO BUILD A NEW TREE? (Y/N)

SITUATION: This prompt is in response to the A key being depressed in response to PROMPT 1. This is a safeguard to prevent inadvertent destruction of a tree structure. A yes response will start building the tree. A no response will result in exiting this option and a return to the MAIN MENU.

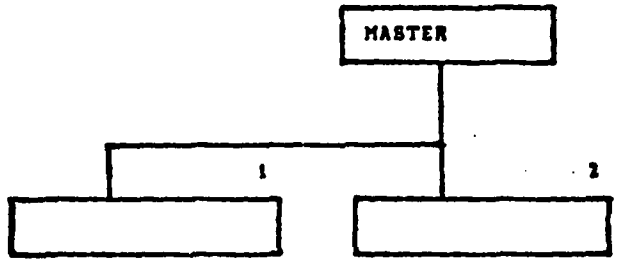
ACTION REQ: Depress the Y or the N key in response to the prompt. Depressing any other key will result in PROMPT 1-A being repeated. A Y response will result in SCREEN DISPLAY 1.

PROMPT 1-B: ENTER...NRN?

SITUATION: This prompt is in response to the S key being depressed in response to PROMPT 1. The program is asking for the specific node reference number to which the descendents are to be added.

ACTION REQ: Enter the node reference number and conclude with the RETURN key. A valid NRN will result in SCREEN DISPLAY 2. Entering an invalid NRN will cause program control to return to the MAIN MENU.

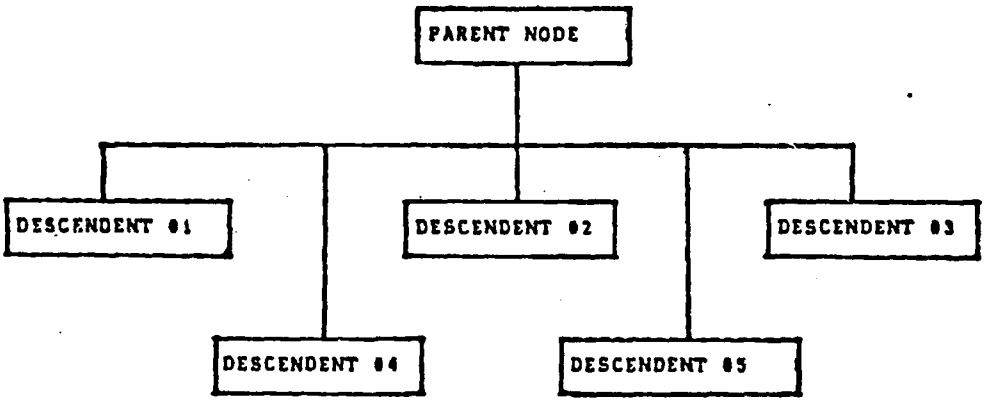
SCREEN DISPLAY 1:



SITUATION: This display indicates that a new structure is to be developed.

ACTION REQ: Enter the label of the root node in box number 1. If the label is less than 10 characters, depress the RETURN key. When box number 2 is drawn, respond by depressing the RETURN key.

SCREEN DISPLAY 2:



CURRENT NUMBER OF NODES: nn

SITUATION: The parent node is listed at the top of the structure. The program will then draw the wire and the box for descendent 1 and wait until an entry is made. After which the program will draw the box for descendent 2 and so on. CURRENT NUMBER OF NODES: nn, gives the current number of nodes presently in the structure and is incremented as nodes are added to the struct e.

ACTION REQ: Enter the label for a descendent and if less than 10 characters, conclude with the RETURN key. If no descendents are to be added or if the last descendent has been added, enter DONE or depress the RETURN key when the next entry (i.e. next box is drawn) is asked for.

SCREEN DISPLAY 2 will be repeated for each node entered until the entire tree structure has been entered. You can exit the option at any time SCREEN DISPLAY 2 is shown by entering EXIT in the descendent box. When the final node label has been entered or EXIT has been entered, the program will return control to the MAIN MENU.

OPTION: STA (tree STAtistics)

USE: This option displays the current number of nodes in the tree structure, the number of levels in the tree structure, and the number of alternatives evaluated by the tree structure.

****************** C A U T I O N S ******************

There are no cautions associated with this option

****************** C A U T I O N S ******************

PROMPT 1:
NUMBER OF NODES nn
NUMBER OF LEVELS nn
NUMBER OF SYSTEMS nn

(ANY KEY) CONTINUE

SITUATION: nn represents various numbers which would indicate the number of nodes, levels, and systems.
The output is paused so that the user can examine the various statistics with regard to the tree structure.

ACTION REQ: After examining the data, press any key to continue.
The program will return control to the MAIN MENU.

OPTION: SYS (entering, adding, or deleting SYStems)

USE: This is used to enter labels for the alternatives (systems) being considered. These are later used to prompt the user to prompt the user for values, and as part of various displays and review formats.

********** C A U T I O N **********

1. USING THE N(EW SUBOPTION WILL NOT DESTROY ANY INFORMATION THAT CURRENTLY RESIDES IN THE DATA NODES. IT SIMPLY RELABELS THE ALTERNATIVES AND INCREASES/DECREASES THE NUMBER OF ALTERNATIVES.

********** C A U T I O N **********

PROMPT 1: A(DD D(ELETE N(EW E(XIT

SITUATION: The user is asked to select a suboption. They are:

ADD add a new alternative to an existing list of alternatives (see MESSAGE 1 and PROMPT 2).

DELETE delete an alternative to an existing list of alternatives (see PROMPT 3 and MESSAGES 2 and 3).

NEW create a new set of alternatives (up to a maximum of five) (see PROMPT 4 AND MESSAGES 4 an 5).

EXIT exit the option and return control to the MAIN MENU.

ACTION REQ: Depress the A, D, N, or E key for the option desired. Pressing any other key will result in PROMPT 1 being repeated.

MESSAGE 1: MAX NUMBER OF SYSTEMS EXCEEDED

SITUATION: The A key was depressed in response to PROMPT 1. There are already five alternatives being evaluated against the current tree structure. After the message is presented, PROMPT 1 will be repeated.

PROMPT 2: ADDING SYSTEM
LABEL?

SITUATION: The A key has been depressed in response to PROMPT 1. The computer is now asking for the title for the alternative that is wished to be added by the user.

ACTION REQ: Enter the title or name of the alternative to be added. The title can be of any length; however, only the first ten (10) characters will be used. Conclude the entry by depressing the RETURN key. The computer will respond with MESSAGE 2.

MESSAGE 2: USE WVC FOR ENTERING VALUES AND
RECALCULATING TREE

SITUATION: This message is printed out to advise the user that the user needs to enter values for the alternative just entered at all data nodes. In addition, the user needs to recalculate the tree in order to obtain correct results in subsequent reviews, displays, and sensitivities. The program will return control to PROMPT 1.

PROMPT 3: CURRENT SYSTEMS . . .
(ALTERNATIVE #1)
(ALTERNATIVE #2)
etc.

ENTER SYSTEM TO BE DELETED

SITUATION: The D key has been depressed in response to PROMPT 1. (ALTERNATIVE #n) refers to the name of the alternative(s) which are currently evaluated by the present tree structure. The user is now asked to input the name of the alternative (system) which is to be deleted. In order to escape from this mode at this point, enter an incorrect alternative name.

ACTION REQ: Enter the name of the alternative, from the list given in the prompt. Conclude entry by depressing the RETURN key. If a correct alternative label is entered, the program will respond with PROMPT 1. Otherwise, the program will respond with MESSAGE 3.

MESSAGE 3: SYSTEM NOT FOUND

SITUATION: The alternative entered in PROMPT 3 was not the same as that listed in PROMPT 3. The program will return control to PROMPT 1 of this option.

PROMPT 4: ENTER...SYSTEM n LABEL?

SITUATION: The N key has been depressed in response to PROMPT 1. This option is used when developing a new tree structure where a several alternatives are to be entered at once and any old data is meaningless.

ACTION REQ: Enter the name of the given system (alternative) and conclude by depressing the RETURN key. The entry can be of any length; however, only the first ten (10) characters will be used. PROMPT 4 will then be repeated.
After entering the last system, and there are less than four alternatives being investigated, type DONE and depress the RETURN key or press the RETURN key alone. MESSAGE 4 will appear and control will return to PROMPT 1 of this option.

MESSAGE 4: NUM OF SYSTEMS n

SITUATION: Message tells the user how many systems (n) that were entered.

MESSAGE 5: SORRY...YOU MUST ENTER AT LEAST ONE SYSTEM

SITUATION: The RETURN key was depressed or DONE was entered and there were no systems (alternatives) entered. The computer will respond with PROMPT 4.

OPTION: TTL (TiTLe of the data structure)

USE: This option elicits a title for later use as a header to various output summaries.

********** C A U T I O N S **********

There are no cautions associated with this option.

********** C A U T I O N S **********

PROMPT 1: ENTER A TITLE FOR THIS DATA STRUCTURE

SITUATION: The computer is requesting that the user enters a title for the current data structure.

ACTION REQ: Enter a title, concluding with the RETURN key. The title can be any length; however, only the first 60 characters will be retained.

OPTION: WVC (Weights, Values, and Calculations)

USE: This option is used to access suboptions for entering weights, enter valuations, or (re)calculating interior values.

********** C A U T I O N S **********

SEE SPECIFIC SUBOPTIONS FOR CAUTIONS

********** C A U T I O N S **********

PROMPT 1: WVC: W(EIGHT V(ALUES C(ALCULATE E(XIT

SITUATION: This initial prompt elicits a suboption which is to be selected.

WEIGHT Input weights to those nodes which have descendents (see next page).

VALUE Enter alternative values at the data nodes (see next page plus one).

CALCULATE Calculate the valuations for the internal (branching) nodes. This is often called 'collapsing' the tree. A detailed discussion of the CALCULATE suboption follows.

EXIT Exit the WVC option and return to the main menu.

ACTION REQ: Depress the W, V, C, or E key to select the desired option. Depressing any other key will result in PROMPT 1 being repeated.

SUBOPTION W(EIGHT

USE: This suboption input weights to those nodes which have descendents.

********** C A U T I O N S **********

1. INPUT OF A NEGATIVE WEIGHT IS NOT DETECTED; HOWEVER, IT WOULD INVALIDATE ALL THE CALCULATIONS WHICH USE IT.
2. ENTERING A ZERO FOR ALL WEIGHTS FOR ANY GIVEN NODE WILL RESULT IN A PROGRAM FAILURE.
3. THERE IS NO ESCAPE FROM THIS SUBOPTION IF A(LL IS SELECTED UNTIL ALL NODES HAVE BEEN WEIGHTED.

********** C A U T I O N S **********

PROMPT 1: WEIGHTS: A(LL S(ELECT

SITUATION: Elicits whether all nodes which have descendents are to be selected or if a user selected node is to be chosen.

ACTION REQ: Depress the A or S key depending on the option desired. If A is depressed, PROMPT 3 and on will occur. If any other key other than A or S is depressed, PROMPT 1 will be repeated.

PROMPT 2: ENTER...NRN?

SITUATION: The S(ELECT option has been selected and the program is asking for a node reference number.

ACTION REQ: Enter the node reference number and conclude with the RETURN key.

PROMPT 2A: NRN ENTERED IS INVALID
(ANY KEY) CONTINUE

SITUATION: The node reference number elicited by PROMPT 2 is not a node which has descendents.

ACTION REQ: Press any key. The computer will respond with PROMPT 2.

PROMPT 3:

	OLD WTS	NEW WTS	NORMALIZED
DESCENDENT #1	XX		
.			
.			
.			
DESCENDENT #N	XX		

SITUATION: The labels of the descendent nodes are listed as well as the current normalized (i.e. the sum equals 1) weights multiplied by 100.

ACTION REQ: Enter the relative weights among the descendents listed. Conclude each entry by depressing the RETURN key. Entries need not equal 100.

PROMPT 4: ARE THESE WEIGHTS OKAY? (Y/N)

SITUATION: The new weights have been normalized (summed to 100). You are asked as to whether the weights entered are as desired. A yes response will result in PROMPT 5. A no response will result in PROMPT 3 being repeated.

ACTION REQ: Depress the Y or N key. Conclude by depressing the RETURN key.

PROMPT 5: ENTER RATIONALE FOR WEIGHTS
(MAX 60 LETTERS)

SITUATION: You are asked to enter any comments concerning the just inserted weights.

ACTION REQ: Enter any comments as desired. The entered rationale may be up to 100 characters in length; however, only the first 60 will be significant.

NOTE: IF A WAS SELECTED FOR PROMPT 1, PROMPTS 3 THRU 5 WILL BE REPEATED UNTIL THE ENTIRE TREE HAS BEEN WEIGHTED.

AT THE COMPLETION OF THIS SUBOPTION, CONTROL WILL BE RETURNED TO OPTION WVC.

SUBOPTION: V(ALUES

USE: This suboption inputs values to the data nodes for each alternative.

********** C A U T I O N S **********

1. ALL NEW VALUES OF ALL ALTERNATIVES FOR THE GIVEN NODE ARE SET TO ZERO.
2. THE ONLY WAY TO SAVE NEW VALUES FOR THE NODE IS TO USE THE E(XIT VALUE SUBOPTION.
3. THE USER MUST UNDERSTAND THE MEANING OF THE VALUES BEFORE ENTERING THEM. VALUES CAN ONLY BE SCALED FROM 0 TO 100.

********** C A U T I O N S **********

PROMPT 1: VALUES: A(LL S(ELECT

SITUATION: Elicits whether all nodes which have descendents are to be selected or if a user selected node is to be chosen.

ACTION REQ: Depress the A or the S key depending on the option desired. If A is depressed, the program will proceed to the SCREEN DISPLAY. An S will result in PROMPT 2. Any other key will result in PROMPT 1 being repeated.

PROMPT 2: ENTER...NRN?

SITUATION: The S(ELECT option has been selected and the program is asking for a data node reference number.

ACTION REQ: Enter the node reference number and conclude with the

RETURN key.

PROMPT 2A: NRN ENTERED IS INVALID
(ANY KEY) CONTINUE

SITUATION: The node reference number selected in response to PROMPT 2 is not a data node.

ACTION REQ: Press any key. The computer will repeat PROMPT 2.

SCREEN DISPLAY:

VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN: 1 2 1 | NODE LABEL

OLD VALUE NEW VALUE

— — —

ALTERNATIVE | F-4

OLD VALUE | 50.00

NEW VALUE |

SITUATION: This display will appear for the selected data node if S was depress in PROMPT 1 or for each data node in turn if A was depressed.

1. NODELABEL represents the title of the data node that the alternative values are entered for.

2. OLD VALUES represents in bar graph format the current values of given alternatives for the present node. Each alternative has a distinctive color (orange, violet, blue, green, or white) and is identified by item 4 (below).

3. NEW VALUES represents in bar graph format, the user supplied values of given alternatives for the present node. Each alternative has its own color as in the OLD VALUE.

4. ALTERNATIVE lists the name of a given alternative surrounded by its distinctive color.

5. OLD VALUE lists the numeric value of the alternative color bar in item 2(above), and labeled in item 4 (above).

6. NEW VALUE Area where the user inputs the new value of the alternative. At the completion of the entry (RETURN key), the appropriate color bar will be drawn to reflect the user input.

OPTIONS UNDER VALUE:

A(BORT	Aborts all changes made by the user and reseves the node with its original values. The program will proceed to the next node or the WVC menu depending on the response to PROMPT 1 of VALUES.
B(ACK	Goes back to the immediate predeeding alternative without changing the value of the present alternative.
E(XIT	Exit the present node, and save the alternative values as indicated in the NEW VALUE graph. (Item 3 above).
N(EXT	Goes to the next alternative without updating the value in the NEW VALUE bar graph (Item 3 above) for the present alternative.
(RETURN)	depressing the RETURN key copies the value of the OLD VALUE (Item 2 above) for the given alternative to the NEW VALUE.
(ESC)	depressing the ESC key will abort the VALUE suboption regardless as to whether all nodes or a selected node was requested. The program

will return to the WVC menu. The node will retain its old values.

(number)(RETURN) Entering a value and depressing the RETURN key will result in the value of the alternative being displayed in bar graph format. Number greater than 100 will be treated as 100 and number less than zero will be treated as zero.

ACTION REQ: Depress the A, B, E, N, RETURN, ESC key for the option desired, or enter the value and conclude with the RETURN key. The value can be up to 10 characters long (including the decimal point). The SCREEN DISPLAY will continue to appear until the E or ESC key is depressed.

At the conclusion of the last node entry (concluding with the E key), the program will automatically return to the WVC menu.

SUBOPTION: C(ALCULATE

USE: Calculate the value for the internal (branching) nodes. Collapses the tree. Resets warnings in main options SEN, REV, DIS, and NUM. Details of how the calculations are carried out can be found in the Programmer's Manual of this document.

********** C A U T I O N S **********

There are no cautions associated with this suboption.

********** C A U T I O N S **********

MESSAGE 1: CALCULATING TREE..

SITUATION: Message appears as tree is being collasped. Additional periods (".") are added to the message to indicate that calculations are still going on. At the completion of the calculations, control will be returned to OPTION WVC.

V. SAMPLE SESSION

The following few paragraphs will demonstrate the use of the DASS through a sample session. The objective of this session will be to install the hierarchical structure shown in Figure 1, conduct a cumulative sensitivity analysis on the node SURVIVABLE, and then to exit the program.

During this presentation, the computer responses will be prefixed by COMPUTER:, while the user's response will be prefixed by USER:. Any side notes regarding specific actions will be enclosed by parentheses. In addition, entries which required the depressing of the RETURN key will be indicated by the symbol (cr).

After turning on the system, the user should depress the X key (to execute the program) and then type DASS and depress the RETURN key.

```
COMPUTER:   ENTER...DISK NAME TO BE USED
            (E.G. APPLE0)

USER:       (cr)

COMPUTER:   PLEASE INSERT  AND DEPRESS RETURN

USER:       (cr)

COMPUTER:   ENTER...FILE NAME TO BE ACCESSED

USER:       EXA (cr)

COMPUTER:   (CR) OR NEW

USER:       NEW (cr)   (Create a new data file)

COMPUTER:   ENTER ATTRIBUTE CHARACTERISTIC (REGRET OR VALUE)?
```

USER: VALUE (cr)

COMPUTER: ENTER A TITLE FOR THIS DATA STRUCTURE

USER: BEST AIRCRAFT (cr)

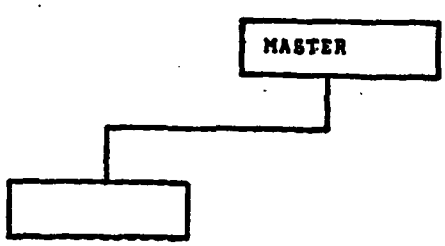
COMPUTER: SPAN NODES. . . A(LL S(ELECT

USER: A

COMPUTER: DO YOU WANT TO BUILD A NEW TREE? (Y/N)

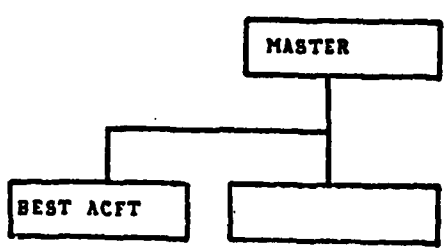
USER: Y

COMPUTER:



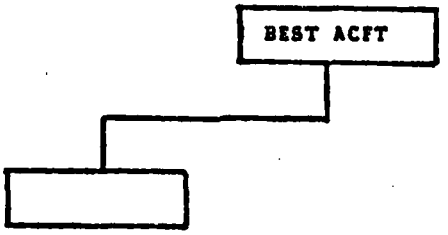
USER: BEST ACFT (cr)

COMPUTER:



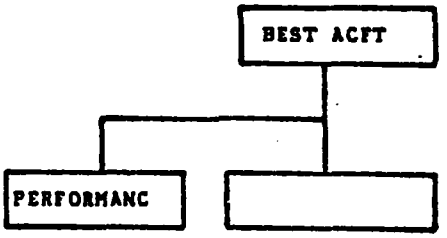
USI (cr)

COMPUTER:



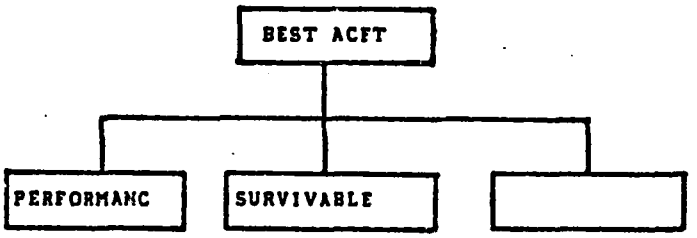
USER: PERFORMANCE (NOTE: the DASS only allows 10 character labels. If labels are greater than 9 characters, do not depress the RETURN key)

COMPUTER:



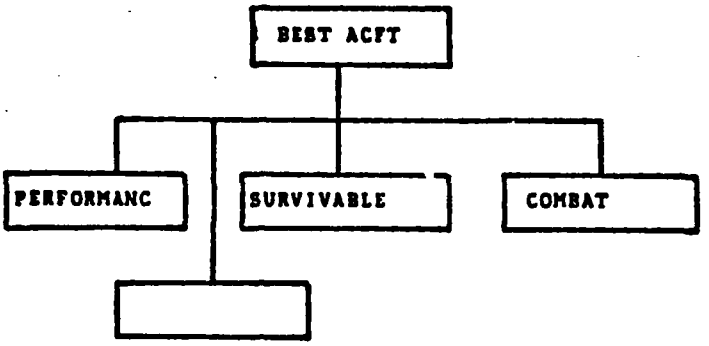
USER: SURVIVABLE

COMPUTER:



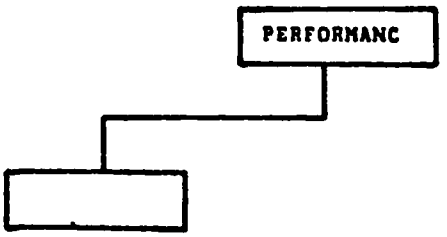
USER: COMBAT (or)

COMPUTER:



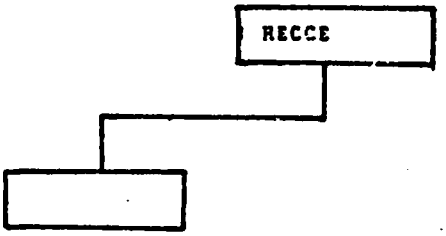
USER: (cr)

COMPUTER:



USER: HI ALT (cr) (The user then enters the remaining nodes putting on descendents as requested. The last entry is stated below)

COMPUTER:



USER: (cr)

COMPUTER: ATT DIS DON MOD NEW NUM PRU REV SEL
SEN SPA STA SYS TTL WVC
OPTION?

USER: SYS (cr)

```
COMPUTER:   A(DD  D(ELETE  N(EW  E(XIT

USER:       N   (Note no (cr))

COMPUTER:   ENTER...SYSTEM 1 LABEL?

USER:       F-4 (cr)

COMPUTER:   ENTER...SYSTEM 2 LABEL?

USER:       F-15 (cr)

COMPUTER:   ENTER...SYSTEM 3 LABEL?

USER:       F-111 (cr)

COMPUTER:   ENTER...SYSTEM 4 LABEL?

USER:       (cr)

COMPUTER:   A(DD  D(ELETE  N(EW  E(XIT

USER:       E   (Note no (cr))

COMPUTER:   ATT DIS DON MOD NEW NUM PRU REV SEL
            SEN SPA STA SYS TTL WVC
            OPTION?

USER:       WVC (cr)      (Enter weights, values and calculate tree)

COMPUTER:   WVC:  W(EIGHT  V(ALUES  C(ALCULATE  E(XIT

USER:       W    (Enter the relative weights on the spams)

COMPUTER:   WEIGHT:  A(LL  S(ELECT

USER:       A

COMPUTER:
                              OLD WTS      NEW WTS      NORMALIZED
            PERFORMANC          0
            SURVIVABLE          0
            COMBAT              0

USER:       1 (cr)
            2 (cr)
            4 (cr)
```

COMPUTER:

	OLD WTS	NEW WTS	NORMALIZED
PERFORMANC	0	1	14
SURVIVABLE	0	2	29
COMBAT	0	4	57

ARE THESE WEIGHTS OKAY? (Y/N)

USER: Y (cr)

COMPUTER: ENTER RATIONALE FOR WEIGHTS

USER: (cr)

COMPUTER:

	OLD WTS	NEW WTS	NORMALIZED
HI ALT	0		
LO ALT	0		

USER: .3 (cr) (This demonstrates any style of weighting scheme
.7 (cr) can be used. The program does the normalisation)

COMPUTER:

	OLD WTS	NEW WTS	NORMALIZED
HI ALT	0	.3	30
LO ALT	0	.7	70

ARE THESE WEIGHTS OKAY? (Y/N)

USER: Y (cr)

COMPUTER: ENTER RATIONALE FOR WEIGHTS
(MAX 60 LETTERS)

USER: (cr) (The user then proceeds on and inserts the relative weights on the last span: AIR-AIR, AIR-GND, and RECCE)

.
.
.

COMPUTER: WVC: W(EIGHT V(ALUES C(ALCULATE E(XIT

USER: V (Enter alternative values of the attributes)

COMPUTER: VALUES: A(LL S(ELECT

USER: A

COMPUTER: VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN:1 1 1 | HI ALT |

OLD VALUE	NEW VALUE
— — —	— — —
ALTERNATIVE	F-4
OLD VALUE	0.0
NEW VALUE	

USER: 45.0 (cr)

COMPUTER: VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN:1 1 1 | HI ALT |

OLD VALUE	NEW VALUE
— — —	— — —
ALTERNATIVE	F-15
OLD VALUE	0.0
NEW VALUE	

USER: 80 (cr)

COMPUTER: VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN:1 1 1 | HI ALT |

OLD VALUE — — —

NEW VALUE

ALTERNATIVE	F-111
OLD VALUE	0.0
NEW VALUE	

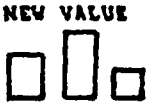
USER: 30 (er)

COMPUTER: VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN:1 1 1 | HI ALT |

OLD VALUE — — —

NEW VALUE



ALTERNATIVE	F-111
OLD VALUE	0.0
NEW VALUE	

USER: E

COMPUTER: VALUE: A(BORT B(ACK E(XIT N(EXT (ESC)

NRN: 1 1 2 | LO ALT |

	OLD VALUE	NEW VALUE
	---	---
ALTERNATIVE	F-4	
OLD VALUE	0.0	
NEW VALUE		

USER: (Enters the worth values for all remaining nodes)

COMPUTER: WVC: W(EIGHT V(ALUES C(ALCULATE E(XIT

USER: C

COMPUTER: CALCULATING TREE.....

WVC: W(EIGHT V(ALUES C(ALCULATE E(XIT

USER: E

COMPUTER: ATT DIS DON MOD NEW NUM PRU REV SEL
SEN SPA STA SYS TTL WVC
OPTION?

USER: NUM (or) (Conduct numeric review to get Node Reference Number)

COMPUTER: A(LL S(ELECT

USER: A

COMPUTER: C(ONSOLE P(RINTER

USER: C

COMPUTER:

BEST AIRCRAFT

NODE	REF NUMBER	LABEL	REL WT	CUMWT
1		BEST PLANE	1.00	1.00
1	1	AERO	.14	.14
1	1 1	HIGH ALT	.30	.04
1	1 2	LOW ALT	.70	.10
1	2	SURVIVABLE	.29	.29
1	3	COMBAT	.57	.57
1	3 1	AIR-AIR	.43	.24
1	3 2	AIR-GND	.43	.24
1	3 3	RECCE	.14	.08

PRESS ANY KEY TO CONTINUE

USER: (Depress the CTRL and A key simultaneously)

COMPUTER:

LABEL	F-4	F-15	F-111
BEST PLANE	47.79	55.04	37.81
AERO	34.50	31.00	47.50
HIGH ALT	45.00	80.00	20.00
LOW ALT	30.00	10.00	55.00
SURVIVABLE	50.00	80.00	40.00
COMBAT	50.00	48.57	34.29
AIR-AIR	50.00	90.00	20.00
AIR-GND	50.00	20.00	40.00
RECCE	50.00	10.00	60.00

USER: (Depress the CTRL and A key simultaneously)

COMPUTER:

BEST AIRCRAFT

NODE REF NUMBER	LABEL	REL WT	CUMWT
1	BEST PLANE	1.00	1.00
1 1	AERO	.14	.14
1 1 1	HIGH ALT	.30	.04
1 1 2	LOW ALT	.70	.10
1 2	SURVIVABLE	.29	.29
1 3	COMBAT	.57	.57
1 3 1	AIR-AIR	.43	.24
1 3 2	AIR-GND	.43	.24
1 3 3	RECCE	.14	.08

PRESS ANY KEY TO CONTINUE

USER: (cr)

COMPUTER: ATT DIS DON MOD NEW NUM PRU REV SEL
SEN SPA STA SYS TTL WVC
OPTION?

USER: SEN (cr) (Sensitivity analysis)

COMPUTER: SENSITIVITY: C)UMWT R)ELWT V)ALUE E)XIT

USER: C

COMPUTER: SENSITIVITY ANALYSIS FOLLOWS

NRN FOR WHICH CUMWT IS
TO BE PERTURBED
ENTER...NRN?

USER: 1 2 (cr)

COMPUTER: 1 2 SURVIVABLE
CURRENT NODE CUMWT IS 0.29

MINIMUM CUMWT (0-1.0) IS?

USER: 0 (cr)

COMPUTER: MAXIMUM CUMWT (0-1.0) IS?

USER: 1 (cr)

COMPUTER: CUMWT: T)ABULAR G)RAPHICAL E)XIT

USER: C

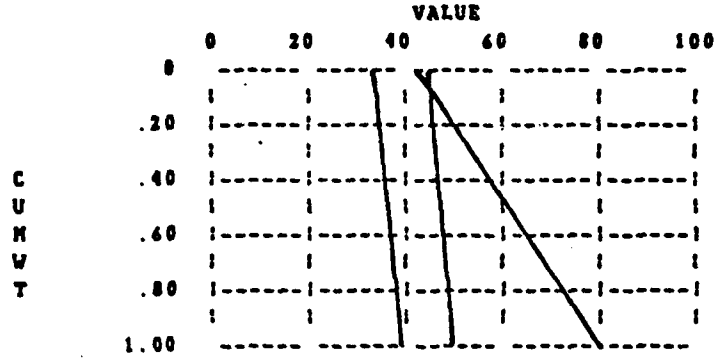
COMPUTER: GRAPHIC: N(ORMAL E(XPANDED

USER: N

COMPUTER:

F-4	F-111
F-15	

SENSITIVITY ANALYSIS
FOR SURVIVABLE NRM 1 Z



USER: (cr)

COMPUTER: CUMWT: T)ABULAR G)RAPHICAL E)XIT

USER: E

COMPUTER: ATT DIS DON MOD NEW NUM PRU REV SEL
SEN SPA STA SYS TTL WVC
OPTION?

USER: DON (cr) (Computer now exits the program)

APPENDIX C

PROGRAMMER'S MANUAL FOR DASS

Table of Contents

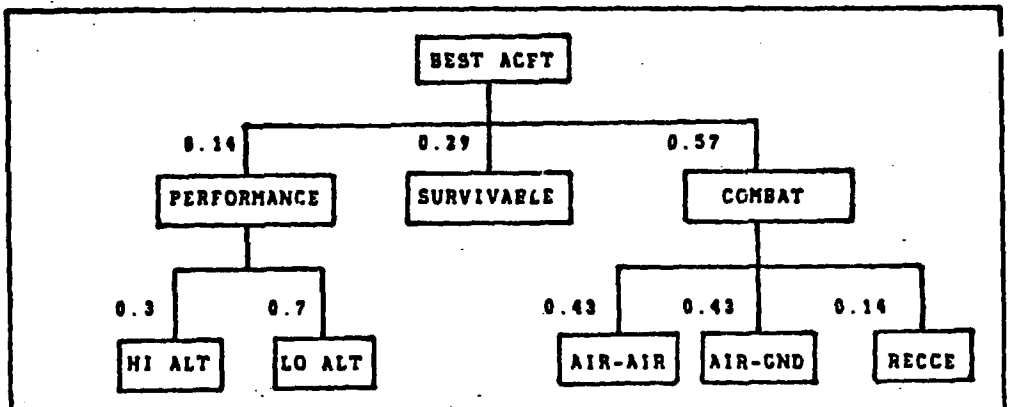
LIST OF FIGURES

I. INTRODUCTION

This programmer's manual is in support of the Decision Analysis Support System as described in the preceeding thesis and appendices. The purpose is to provide future analysts with information pertinent to the program itself for modifications and improvements.

This program was written in USCD PASCAL as supported by the Apple II Mirorecomputer. Included in the program is computer color graphics which is an implemented feature on the Apple II.

Many of the comments and examples presented in this manual refer to Figure 1. The programmer and interested readers are urged to refer to this figure when reading the following sections.



Value						
F-4	45.0	30.0	50.0	50.0	50.0	50.0
F-15	80.0	10.0	80.0	90.0	20.0	10.0
F-111	30.0	55.0	40.0	20.0	40.0	60.0

Figure 1 Sample Tree Structure

II. ARRAY STRUCTURES

The array structures in this program follows that of the DASS program put forth in the Applesoft BASIC program. There are three primary arrays that are carried in internal memory. The names are IRAY, ARAY, and VRAY. The following paragraphs will describe each array and provide appropriate row and column definitions.

All arrays are set up such that each row, starting with row 1, represents a level in the structure. For example, if row two had node data in it, that would mean that that particular node was a level two type node (i.e. first descendent from the root node). The only case that this is not true is when PROCEDURE CROSS (in UNIT A) is executed. In this procedure, the arrays additionally hold a complete span of the tree which contains the node which invoked the procedure.

Array IRAY is a an 11 row by 6 column array housing the various link pointers. The rows and columns are labelled starting with zero and running to ten and five respectively. Each row represents a particular node with each column representing the node number, the location of the node on its span relative to its brothers and sisters, the downlink node, the crosslink node, the backlink node, and the backlink node.

Column zero represents the node number that is assigned to the node at the time of entry. Each node has a uniquely assigned node number. This number represents the record number of the disk file where

all nodes reside in mass storage. In addition, this node number is used in identifying downlinks, crosslinks, backlinks, and uplinks in array IRAY.

Column one represents the nodes position in the span for the level the node is at. The very first node in the span is labelled one. This is used to identify the node using the node reference number system (As described above and in the User's Manual).

Column two represents the downlink node. The downlink node is that node which is the first descendent to the node identified in column zero. If there are no descendents to the node, the downlink will be zero.

Column three represents the crosslink node. The crosslink node is that node which is a sibling of the node. If the node does not have any more sib'ings, the crosslink will be zero.

Column four represents the backlink node. The backlink node is that node which eventually links the node back to the root node (top-most node). The backlink can be either the parent node or a sibling depending on the nodes position on the span. If the node is the first node on the span (reading left to right), its backlink node will be its parent. If the node is on any other position, its backlink node will be its sibling on its immediate left.

Column five represents the uplink node. This column is not saved on the disk file but is used in PROCEDURE SPAN in SEGMENT PROCEDURE DUMMY and the uplink node is used in PROCEDURES NEXT and FIND in UNIT

DASSA. It points to the node's parent, during node search and node spanning.

Array ARAY is the data array which houses the relative and cumulative weights for a node. Relative weights refers to the relative importance of the node to its siblings. All weights are between zero and one, but the sum of the relative weights of all siblings on a given span is equal to one. Relative weights are assigned using option WVC (see User's Manual) which is found in SEGMENT PROCEDURE WVLOAD under PROCEDURE RDWT.

Cumulative weights refer to the impact of the node to the overall tree structure. This is particularly important in accomplishing sensitivity analyses. The cumulative weights are calculated using PROCEDURE CALC i SEGMENT PROCEDURE WVLOAD. The actual calculations used in calculating the tree (cumulative weights) will be discussed in a later section.

Array VRAY is the data array which carries worth value information for the alternatives. The VRAY array consists of five columns each representing an alternative under investigation. Values inside this array are inputted directly into data nodes (the bottom-most nodes in the tree structure) using option WVC (PROCEDURE RDV in SEGMENT PROCEDURE WVLOAD). The values of nodes other than data nodes are calculated using PROCEDURE CALC in SEGMENT PROCEDURE WVLOAD.

III. Disk File Structures

Two separate files are created and maintained for each tree structure developed by the DASS program. The first file called NODE (see UNIT DASSA) contains the record for each node. In addition, tree structure statistics and various titles are stored in this file. The second file, called SYSTEMS (also in UNIT DASSA) contains the labels for all the alternatives being investigated. The following few paragraphs will describe the files in detail.

File NODE, as indicated above contains record data for each node. The record contains the label of the node (NODETITLE), the cell number (CELLNUMBER), the position the node is on its span (NRNDIGIT), its downlink node (DOWNLINK), its crosslink node (CROSSLINK), and its backlink node (BACKLINK). In addition, the record contains the node's relative weight (RELWEIGHT) and cumulative weight (CUMWEIGHT), as well as values of the alternatives for that node either calculated or inputted (SYSTEMVALUES ARRAY[1..5]). Finally, each record contains a comment concerning that particular node (RATIONALE).

For ease of programming, the node record number (used in various GETs and PUTs throughout the program) is the same as the cell number of the node. Thus, values of the various linking pointers (DOWNLINK, CROSSLINK, and BACKLINK) can be used to GET the specified node directly.

In the case of APPLE II all random access files (of which NODE

SYSTEMS are), the first record of the file is labelled zero, the xt one, and so on. Since the record number for the node is also the cell number of the node, record zero is not used for the nodes themselves. However, node zero is used to contain various tree statistics and titles. The following paragraphs describe which global variables are stored in record zero of file XXXNODE.DATA.

The NODETITLE variable contains the current attribute characteristic contained in global variable LATT, which is created when option ATT is exercised (PROCEDURE LABELATTRIBUTES in SEGMENT PROCEDURE DUMMY).

The CELLNUMBER variable contains the value of the global variable NNODES (number of nodes in the tree).

The NRNDIGIT variable in the record contains the current number of levels contained in the tree structure. Levels as discussed previously are the number of generations eminating from the root node, including the root node.

The DOWNLINK variable contains the current number of alternatives (or systems) that are being investigated by the tree. The information is transferred to and from the global variable NSYS.

The CROSSLINK variable contains the condition of the tree structure as identified by the global variable FLAG. If FLAG is non-zero, additions and/or deletions have been made to the tree structure, and the tree has not been re-calculated.

Finally, the RATIONALE variable contains the title of the tree

structure which is obtained by option TTL (PROCEDURE READTITLE in SEGMENT PROCEDURE DUMMY). The global variable TITLE contains the tree structure title during program operation.

Other variables in record 0 of file XXXNODE.DATA not mentioned are currently not used and are available for future use.

Disk access of the nodes are conducted throughout the program; however, node zero is only accessed at two points. The first point is when the file is first accessed (PROCEDURE SELECTFILE in SEGMENT PROCEDURE DUMMY) and the second is when the file is closed (PROGRAM DASS or PROCEDURE SELECTFILE in SEGMENT PROCEDURE DUMMY).

While access to the nodes occur throughout the program, the information transferred from the nodes to the holding arrays (IRAY, ARAY, and VRAY) are all done via PROCEDURES NODEARRAYTODISK and NODEDISKTOARRAY found in UNIT DASSA.

The second disk file used by the DASS program is called SYSTEMS. Its purpose is to contain the labels of the alternatives (systems) currently being evaluated by the tree structure. The variable used in the file is called SYSTEMNAME. Since the file is small (current restrictions, due to the number of colors available, limit the number of alternatives to five), whenever a requirement for an alternative name is required, the file is accessed (except in SEGMENT PROCEDURE WVLOAD where the systems are read into an array SYS and used during loading values in PROCEDURE RDV).

IV. Sensitivity Analysis

The sensitivity analysis in this version of DASS not only conducts variations for a particular node's cumulative weight, but also a node's relative weight and if the node is a data node, the variation of a value of a particular alternative. The following paragraphs will describe the models for the options above. The code which is used to implement these models are PROCEDURE CALCARRAY within SEGMENT PROCEDURE SENSITIVITY.

Sensitivity on Cumulative Weight

In this analysis, we are examining the effect on the overall tree ("root" node) based on a change in the cumulative weight on any node.

Note that the composite value of the "root" node for a given alternative is:

$$\text{VALUE(ROOT NODE)} = \sum_{\substack{\text{all} \\ \text{attributes}}} \text{CUMULATIVE WEIGHT(ATTRIBUTE)} * \text{VALUE(ATTRIBUTE)}$$

In the analysis of cumulative weight, we are interested in replacing a particular cumulaive weight of any node with a new weight. Since the tree is normalized, we can calculate a new composite value.

The procedure is to remove the contribution of the node from the

"root" node and replace it with a new contributed value. Note that the contribution of any node to the overall "root" node is:

CONTRIBUTION(NODE) = CUMULATIVE WEIGHT(NODE) * VALUE(NODE,ALTERNATIVE)

Since changing the cumulative weight of any node results in the tree no longer being normalised, the tree, itself, must undergo a transformation. The transformation must be such that the transformed tree's cumulative weight is equal to one minus the cumulative weight of the perturbed node.

A single equation was developed to accomplish the above procedure for a given alternative:

$$\begin{aligned} \text{NEW VALUE(ROOT NODE)} = {} & (\text{VALUE(NODE)} * \text{NEW CUMULATIVE WEIGHT}) + \\ & [(1 - \text{NEW CUMULATIVE WEIGHT}) / \\ & \quad (1 - \text{OLD CUMULATIVE WEIGHT(NODE)})] * \\ & ((\text{VALUE(ROOT NODE)} - \\ & \quad (\text{VALUE(NODE)} * \text{OLD CUMULATIVE WEIGHT(NODE)}))) \end{aligned} \qquad [1]$$

Thus the procedure to determine the change in the "root" value of a given alternative given a change in a cumulative weight of some attribute or objective is as follows:

(1) Pick a new cumulative weight of the node to be examined.

(2) For each alternative, determine the new "root" node value for the alternative using equation [1].

For example, let's look at the effect of changing the cumulative weight of node HI ALT from 0.042 to 0.06 on alternative F-4.

For alternative F-4:

VALUE(BEST ACFT,F-4) = 47.79

VALUE(HI ALT,F-4) = 45.00

For node HI ALT

CUMULATIVE WEIGHT(HI ALT) = 0.042

Using equation [1]:

NEW VALUE(BEST ACFT) = (VALUE(HI ALT,F-4)* NEW CUMULATIVE WEIGHT) +

((1 - NEW CUMULATIVE WEIGHT)/
(1 - CUMULATIVE WEIGHT(HI ALT)) *

(VALUE(BEST ACFT,F-4) -
(VALUE(HI ALT,F-4) * CUMULATIVE WEIGHT(HI ALT))

= (45.00 * 0.06) +

((1 - 0.06) / (1 - 0.042)) *

(47.79 - (45.00 * 0.042))

= 47.73

Sensitivity on Relative Weight

What we are examining is to see the change in alternative values at the "root" node based on a change of a relative weight on a span

anywhere in the tree. Note that the "root" node is the overall objective of the tree structure.

In performing the analysis, we need to look at how the values of a node are calculated. Note that for any node, the value of an alternative for that node is just the sum of the products of each immediate descendent alternative value and its relative weight or:

$$\text{VALUE(PARENT NODE)} = \sum_{\substack{\text{all} \\ \text{immediate} \\ \text{descendents}}} \text{RELATIVE WEIGHT(DESCENDENT)*VALUE(DESCENDENT)} \qquad [2]$$

Thus we can say that a change in the relative weight of any node will only affect the value of the parent node. In addition, note that the value of any node is only dependent on its own immediate descendents and not on the relative weight of the node itself. Therefore, in examining a change of the relative weight of a node, the new alternative values of its parent need to be calculated and substituted for the incumbent values of the parent. This substitution will then effect the value of the "root" node.

Substitution of the new values into the "root" node, fortunately is straight-forward. Note that the value contribution of any node to the "root" node is just the product of its value multiplied by its cumulative weight or:

NODE'S CONTRIBUTION TO THE ROOT NODE = CUMULATIVE WEIGHT(NODE) * VALUE (NODE) [3]

Since the cumulative weight of the parent is unchanged, the change in the value of the parent can be added to the "root" node or:

NEW VALUE(ROOT NODE) = OLD VALUE(ROOT NODE) - (CUMULATIVE WEIGHT(PARENT) * OLD VALUE(PARENT)) + (CUMULATIVE WEIGHT(PARENT) * NEW VALUE(PARENT)) [4]

A question arises as to how to distribute the remaining relative weight to the node's siblings. An arbitrary rule was made to keep the relative weights of the siblings at the same proportions as was in the incumbent situation. For example, let us have three objective nodes A, B, and C. Let each of these objectives have the relative weights of 0.7, 0.2, and 0.1 respectively. If we vary the relative weight of A from 0.7 to 0.8, then node B will have a relative weight of 0.133 and node C will have a relative weight of 0.067. If we varied node A from 0.7 to 0.6, node B will have a relative weight of 0.267 and node C 0.133. Note that the ratio of B to C is unchanged in all cases.

Thus the overall procedure in examining the effects of a change in a relative weight of a given node on the "root" node is:

(1) Pick a new relative weight of the node to be examined.

(2) Redistribute the remaining relative weights (that is one minus the relative picked in step (1)) among the node's siblings.

(3) Recalculate the value of the node's parent using the new relative weights selected in steps (1) and (2) using equation [2]. Note that the values of the node and its siblings are unchanged.

(4) Substitute the new value of the parent node in place of its old value at the "root" node using equation [4].

For example, let us evaluate the impact on alternative F-15 when the relative weight of AIR-GND changes from its incumbent relative weight of 0.43 to 0.50.

After establishing the new relative weight for node AIR-GND, compute the new relative weights of nodes AIR-AIR and RECCE using the relative weight of AIR-GND of 0.50 and maintaining the old proportion between AIR-AIR and RECCE. We can determine the appropriate values through the following equation:

$$\text{NEW RELATIVE WEIGHT(SIBLING)} = \frac{\text{OLD RELATIVE WEIGHT (SIBLING)}}{(1 - \text{OLD RELATIVE WEIGHT (NODE)})} * \frac{}{(1 - \text{NEW RELATIVE WEIGHT (NODE)})}$$

Thus for node AIR-AIR:

$$\text{NEW RELATIVE WEIGHT(AIR-AIR)} = \frac{\text{OLD RELATIVE WEIGHT (AIR-AIR)}}{(1 - \text{OLD RELATIVE WEIGHT (AIR-GND)})} * \frac{}{(1 - \text{NEW RELATIVE WEIGHT (AIR-GND)})}$$

$$= \frac{0.43}{(1-0.43)} * (1-0.50)$$

$$= 0.377$$

and for RECCE:

$$\text{NEW RELATIVE WEIGHT(RECCE)} = \frac{0.14}{(1-0.43)} * (1-0.50) = 0.123$$

The next step is to calculate the new value of AIR-GND's parent, COMBAT using the new relattive weights calculated above using equation [2] for alternative F-15:

$$\begin{aligned}\text{NEW VALUE(COMBAT,F-15)} = {} & \text{RELATIVE WEIGHT(AIR-AIR)} * \text{VALUE(AIR-AIR,F-15)} + \\ & \text{RELATIVE WEIGHT(AIR-GND)} * \text{VALUE(AIR-GND,F-15)} + \\ & \text{RELATIVE WEIGHT(RECCE)} * \text{VALUE(RECCE,F-15)}\end{aligned}$$

$$= 0.377 * 90 + 0.50 * 20 + 0.123 * 18$$

$$= 45.16$$

The final step is to substitute the new value just calculated for COMBAT into the "root" node, BEST ACFT. Using equation [4]:

NEW VALUE(BEST ACFT) = OLD VALUE(BEST ACFT,F-15) -
CUMULATIVE WEIGHT(COMBAT) * OLD VALUE(COMBAT,F-15) +
CUMULATIVE WEIGHT(COMBAT) * NEW VALUE(COMBAT,F-15)

= 55.84 - 0.57 * 48.57 + 0.57 * 45.16

= 53.09

Therefore, the change in the value of alternative F-15 in the objective BEST ACFT given a change in te relative weight of node AIR-GND from 0.43 to 0.50 is 53.09.

Sensitivity on Attribute Value

The objective in this senitivity analysis is to examine the effect varying an attribute alternative value on the "root" node.

In this analysis note that varying a value of an alternative does not affect the values of any other alternatives (independence among alternatives). In addition, note that changing values does not effect the tree structure in either the cumulative or relative weights. Therefore, a change in the "root" node can be experienced only for the alternative varied and then by the amount of the cumulative weight of the attribute or:

NEW VALUE(ROOT NODE,ALTERNATIVE) = OLD VALUE(ROOT NODE,ALTERNATIVE) -

CUMULATIVE WEIGHT(ATTRIBUTE) *
INCUMBENT VALUE (ATTRIBUTE,ALTERNATIVE) +

CUMULATIVE WEIGHT(ATTRIBUTE) *
NEW VALUE (ATTRIBUTE,ALTERNATIVE) . [5]

Note that varying the value of an alternative for an attribute only effects the value of the "root" node for that alternative and the values of the "root" node for all other alternatives are unchanged.

Thus the procedure in examining the effects of changing an alternative value for a given attribute is:

(1) Select a new value of the attribute for a given alternative.

(2) Calculate the new value of the "root" node for the given alternative using equation [4]. All other alternative values for the "root" node will remain unchanged.

For example, let us look at the effect on the value of alternative F-111 at the overall objective BEST ACFT when alternative F-111 changes value at node SURVIVABLE from 40.0 to 60.0.

Using equation [5]:

NEW VALUE(BEST ACFT,F-111) = OLD VALUE(BEST ACFT,F-111) -

CUMULATIVE WEIGHT(SURVIVABLE) *
OLD VALUE(SURVIVABLE,F-111) +

CUMULATIVE WEIGHT(SURIVIABLE) *
NEW VALUE(SURVIVABLE,F-111)

= 37.81 - 0.29 * 40 + 0.29 * 60

= 43.61

Thus a change in the value of the alternative F-111 at node SURVIVABLE from 40 to 60 changes the value of the "root" node BEST ACFT from 37.81 to 43.61.

V. Hierarchy Manipulation

PROCEDURES NODIN and FIND

These procedures (found in UNIT DASSA) are used to elicit a single node (by NRN) from the user, and find that node. If the selected node does not exist the routines will find the node which is closest in the data structure to where the input node would be. Most options (with the exception of MOD) will interpret the non-existence of a node as an indication that the user is finished with the option, and will return program control to the program unit which called that option. In the case of option MOD (SEGMENT PROCEDURE MODPRU), additional branches and nodes will be added to match the node entered by the user.

PROCEDURE NODIN. The routine NODIN merely reads in the user selected node reference number. If the user entered an NRN, then NODIN will call FIND to search for that node.

PROCEDURE FIND. FIND searches the tree for a match to the input node. This is accomplished through a modified breadth-first search. Starting with the top level, a cross' nk search is conducted for an NRN digit match at each level. Failure to find a match at a level indicates that the input node should be added as a new crosslink at that level. If a match is found, the next level down is searched for the next input NRN digit. If there is no downlink path, then the routine is terminated. Figure 2 provides an example of the PROCEDURE FIND

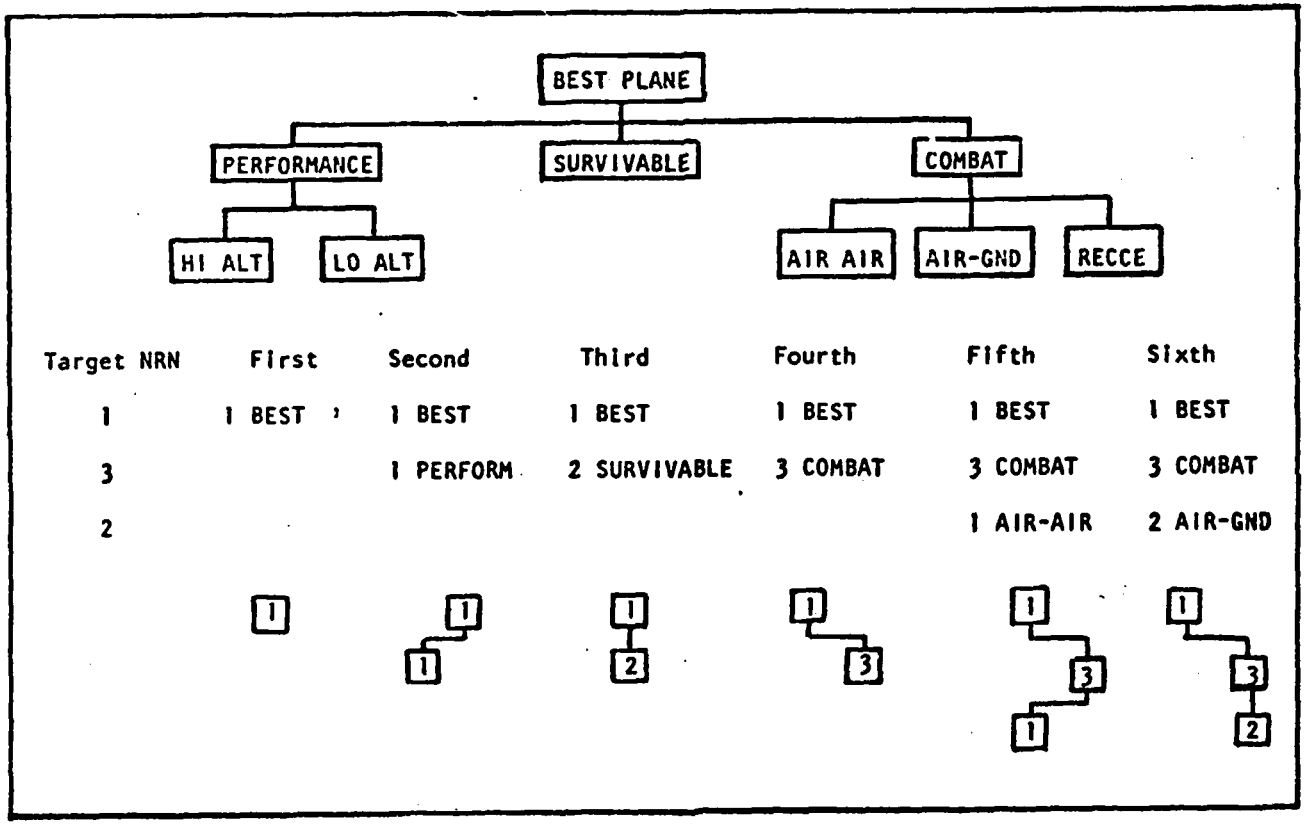


Figure 2 Finding a Node

routine given that the NRN provided from PROCEDURE NODIN was 1,3,2.

These routines are used extensively by other options within the program, where the user must select a single node.

PROCEDURES PRENEX, PRETOT, and NEXT.

These procedures (found in UNIT DASSA) are used by DASS to sequentially traverse the tree structure. PROCEDURES PRENEX and PRETOT are used to indicate where the traversal is to start: at the "root" node in the case of PROCEDURE PRETOT, or at any descendent node in the case of PROCEDURE PRENEX. The actual traversal is accomplished by PROCEDURE NEXT.

PROCEDURE PRENEX. This procedure elicits a NRN from the user using PROCEDURE NODIN as described above. If a valid NRN is entered, the procedure starts the traversal at the node entered, and cause PROCEDURE NEXT to traverse only its descendents (sets ITOTL equal to one).

PROCEDURE PRETOT. This procedure causes the array pointers (LVL) to be reset to the top of the arrays and sets the appropriate flags for an entire tree traversal (ICONT, NDIFF, and ITOTL equal to one).

PROCEDURE NEXT. This procedure tours the hierarchical structure starting at locations specified by PROCEDURES PRETOT or PRENEX. The algorithm used is Captain Morlan's improved version which was as described in his Technical Report. The basic flow is shown in Figure 3.

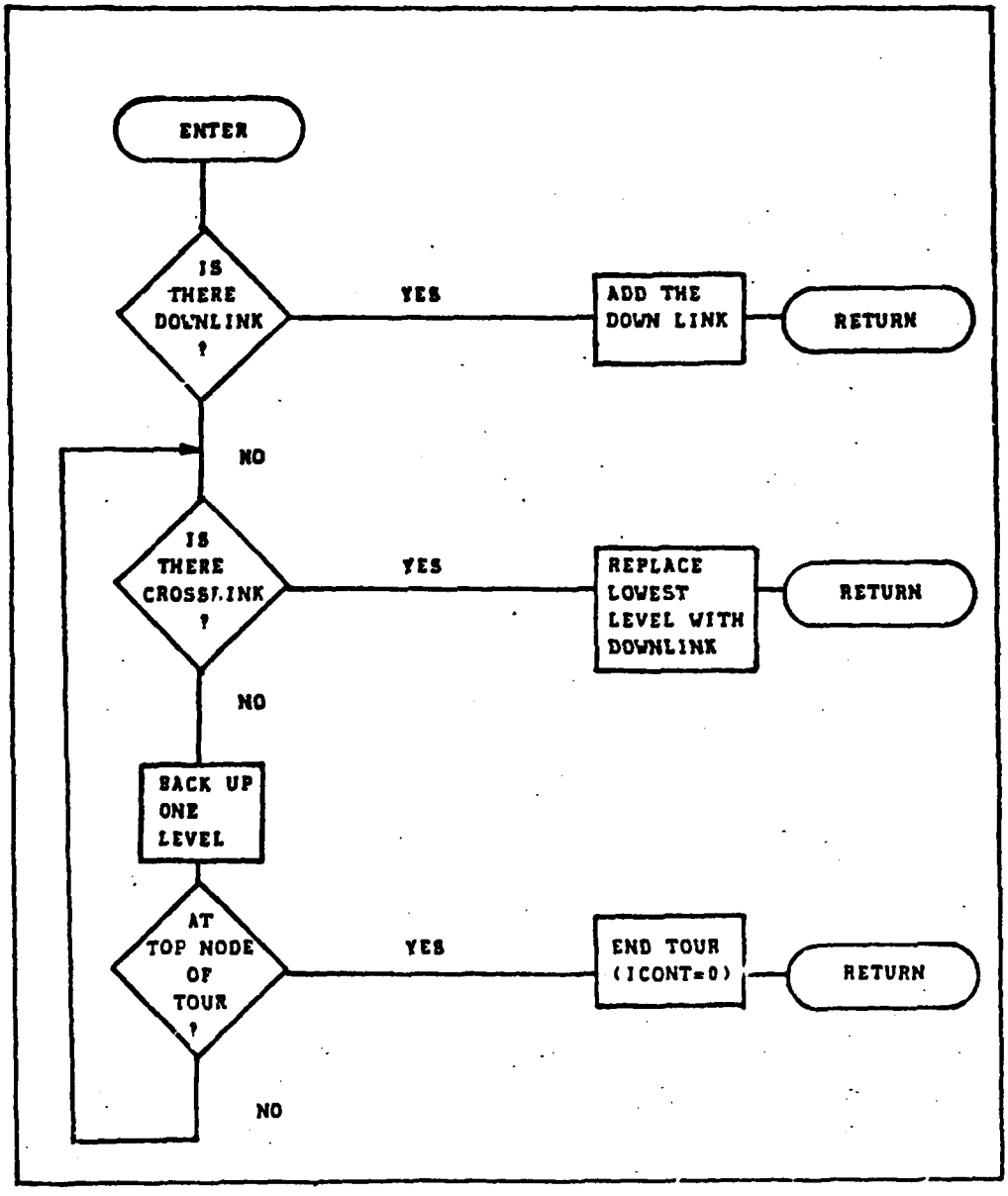


Figure 3 Flow of PROCEDURE NEXT

PROCEDURE SPAN

This procedure (found in UNIT DASSA) allows the user to quickly create a tree structure. The user enters the immediate descendents of selected nodes. If no descendents are to be added, the procedure will go to the next sibling and ask for descendents. If there are no siblings left, the procedure will go up one level ask for descendents to the next parent's sibling. Figure 4 shows the logic option for the procedure.

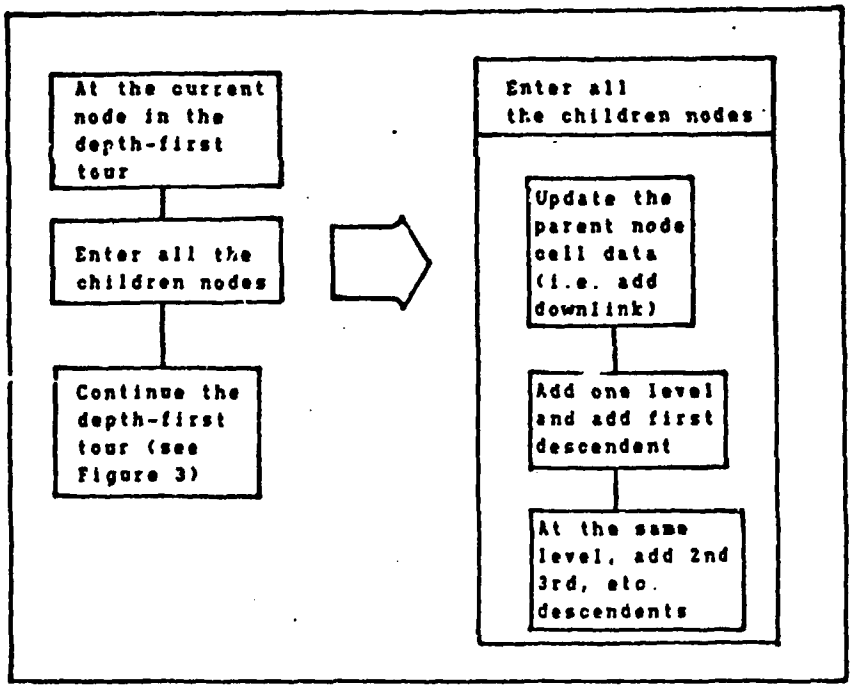


Figure 4 Flow Diagram of PROCEDURE SPAN

PROCEDURE CALC

The calculation of composite values (collapsing) for the hierarchy is accomplished through a modified depth-first tour. The tree is examined branch by branch (that is, a path from the "root" node to an attribute) and the cumulative weights are calculated. Values based on the attributes are then computed and assigned to the appropriate node. The logic for this process is shown in Figure 5. For example, the first branch that would be examined would be BEST ACFT - PERFORMANCE - HI ALT. The procedure would then calculatte the cumulative weight of each node following the following equation:

CUMULATIVE WEIGHT(NODE) = CUMULATIVE WEIGHT(PARENT) *
RELATIVE WEIGHT(NODE)

The resulting cumulative weights would be BEST ACFT 1.0, PERFORMANCE 0.14, and HI ALT 0.042.

The value of the alternatives are then multiplied by the cumulative weight of the upper nodes. This provides the actual value contributions of the attributes to all parents. For example, let us calculate the value of the contribution of attribute HI ALT to the "root" node BEST ACFT and intermediate node PERFORMANCE for alternative F-4.

For BEST MODE:

CONTRIBUTION OF HI ALT TO BEST ACFT = (CUMULATIVE WEIGHT(HI ALT)/ CUMULATIVE WEIGHT(BEST ACFT)) * VALUE(HI ALT,F-4)

numerically:

CONTRIBUTION OF HI ALT TO BEST ACFT = (0.042/1.00) * 45

= 1.89

For PERFORMANCE:

CONTRIBUTION OF HI ALT = (CUMULATIVE WEIGHT(HI ALT)/ CUMULATIVE WEIGHT(PERFORMANCE) * VALUE(HI ALT,F-4)

= (0.042/0.14) * 45

= 13.5

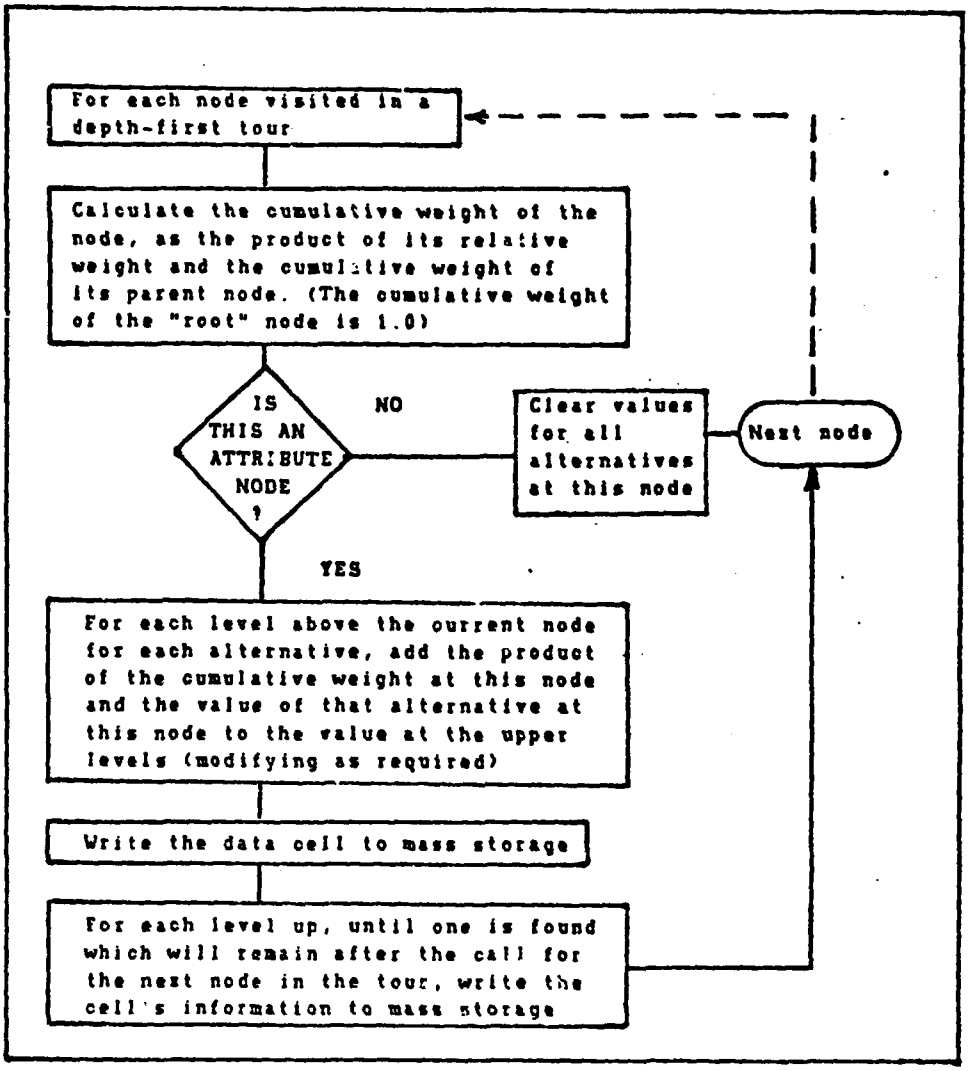


Figure 5 Collapsing the Hierarchy

VI. Variable List

This section describes the global variable list which contains variables available to all units and segments in the dass program. These variables can be found in UNIT DASSA in the program listing.

GLOBAL VARIABLE LIST

ANSWER	STRING VARIABLE USED TO CONTAIN USER RESPONSES TO VARIOUS NON-SINGLE LETTER PROMPTS.
ARAY	REAL ARRAY USED TO CONTAIN THE RELATIVE WEIGHTS AND CUMULATIVE WEIGHTS OF THE BRANCH OF THE TREE STRUCTURE CURRENTLY RESIDING IN THE ARRAY
CH	CHARACTER VARIABLE WHICH CONTAINS VARIOUS RESPONSES FOR COMPUTER PROMPTED QUESTIONS AND MENUS
CMD	STRING VARIABLE WHICH CONTAINS THE INPUTTED COMMAND OPTION IN RESPONSE TO THE MAIN MENU
COMMENTSTRING	STRING VARIABLE WHICH CONTAINS THE NUMBER OF BLANKS EQUAL TO THE CURRENT MAXIMUM COMMENT SIZE
DISKNAME	STRING VARIABLE WHICH CONTAINS THE DISK NAME (VOLUME NAME) OF THE DISK WHICH CONTAINS THE NODE AND ALTERNATIVE DATA SETS FOR THE TREE STRUCTURE
FILENAME	STRING VARIABLE WHICH CONTAINS THE APPLE II DISK FILE NAME HOUSING THE NODE INFORMATION FOR A TREE STRUCTURE

FILEOFSYSTEMS	STRING VARIABLE WHICH CONTAINS THE APPLE II DISK FILE NAME HOUSING THE NAMES OF THE ALTERNATIVES FOR A TREE STRUCTURE
FLAG	INTEGER VARIABLE WHICH IDENTIFIES WHETHER ADDITIONS AND DELETIONS FOR NODES AND ALTERNATIVES HAVE BEEN MADE AND THAT THE TREE STRUCTURE HAS OR HAS NOT BEEN (RE)CALCULATED. IF THE FLAG IS SET, WARNING MESSAGES WILL OCCUR IN OPTIONS DIS, REV, NUM (SEGMENT PROCEDURE NUM) AND OPTION SEN (SEGMENT PROCEDURE SENSITIVITY). 0 = TREE HAS BEEN RECALCULATED 1 = TREE HAS NOT BEEN RECALCULATED
I	INTEGER VARIABLE WHICH IS USED AS A GENERALIZED COUNTER USED THROUGHOUT ALL THE PROGRAM.
ICONT	FLAGS WHETHER TO CONTINUE OP NOT IN TRANVERSING TREE CREATED BY PROCEDURES PRETOT, PRENEX, OR NEXT GROUP OF ROUTINES TO TRANVERSE TREE. 0 = DO NOT CONTINUE WITH PROCESSING 1 = CONTINUE PROCESSING
IFADD	FLAGS RELATIONSHIP OF IFIND NODE TO INPUT NODE, 2 = IFIND IS PARENT TO INPUT NODE 3 = IFIND IS BROTHER TO INPUT NODE
IFIND	POINTS TO CELL CONTAINING THE TERMINAL OF THE BRANCH OF NODES WHICH MATCH THE INPUT NRN VECTOR.
IRAY	INTEGER ARRAY WHICH HOUSES A BRANCH OF THE TREE STRUCTURE AND CONTAINS THE NODE NUMBER, POSITION OF THE NODE ON THE SPAN, DOWNLINK, CROSSLINK, BACKLINK AND UPLINK
ISTR	STRING VARIABLE AUXILIARY TO STRING VARIABLE ANSWER. CONTAINS USER INPUTTED RESPONSES TO VARIOUS NON-SINGLE LETTER PROMPTS.
ITOTL	FLAGS TYPE OF TREE TRANVERSAL 0 = NOT A TOTAL TREE TRAVERSAL

1 = TOTAL DOWN NODE TRAVERSAL (FROM INPUT NODE ON DOWN)

L	INTEGER VARIABLE WHICH IS USED AS A GENERALIZED COUNTER USED THROUGHOUT THE PROGRAM.
LABELSTRING	STRING VARIABLE WHICH CONTAINS NODE LABELS AT THE TIME OF ENTRY (PROCEDURE SPAN IN SEGMENT PROCEDURE DUMMY)
LATT	CONTAINS THE LABEL FOR THE ATTRIBUTES (REGRET OR VALUE)
LVL	LENGTH OF LEVEL NRN VECTOR (FROM FIND). ALSO USED TO DETERMINE THE CURRENT DEPTH OF THE BRANCH CURRENTLY IN THE ARRAYS.
NCROSS	INTEGER VARIABLE WHICH CONTAINS THE NUMBER OF NODES ON A GIVEN SPAN. USED IN PROCEDURE CROSS (UNIT DASSA) AND THOSE ROUTINES WHICH USE PROCEDURE CROSS (PROCEDURE DISPLAY (SEGMENT PROCEDURE NUM) AND SEGMENT PROCEDURE SENSITIVITY).
NDEEP	THE DEPTH OF THE TREE STRUCTURE (MAXIMUM NUMBER OF LEVELS, MAXMUM NLVLS). SET BY CALC (PROCEDURE SEGMENT WVLOAD).
NDIFF	NUMBER OF LEVELS NOT MATCHED IN NRNVECTOR (LENGTH INPUT VECTOR) - (LENGTH OF IFIND VECTOR)
NFLAG	FLAG WHICH INDICATES WHETHER OR NOT AN OPEN DISK FILE IS PRESENTLY OPEN. USED IN PROCEDURE SELECTFILE IN SEGMENT PROCEDURE DUMMY.
NLVLS	CONTAINS THE LENGTH OF THE INPUT NRN VECTOR (LEVEL)
NNODES	NUMBER OF NODES IN TREE

NODELABEL	STRING ARRAY USED TO HOUSE THE TITLES OF VARIOUS NODES CURRENTLY RESIDING IN ARRAYS IRAY, ARAY, AND VRAY.
NRNVECTOR	CONTAINS THE INPUT NRN VECTOR (NUMBERS)
NSYS	NUMBER OF ALTERNATIVES (SYSTEMS) TO BE CONSIDERED
OLDCELLNUM	INTEGER VARIABLE USED IN PROCEDURE SPAN (SEGMENT PROCEDURE DUMMY) WHICH CONTAINS PRIOR CELL INFORMATION IN THE CONSTRUCTION OF THE TREE
QUESTION	STRING VARIABLE WHICH CONTAINS TK* PROMPT FOR WHICH A USER RESPONSE WILL BE REQUIRED
RECORDID	INTEGER VARIABLE NOT USED
TITLE	STRING VARIABLE WHICH CONTAINS THE TITLE OF THE TREE STRUCTURE
VRAY	REAL ARRAY WHICH CONTAINS THE VALUES OF THE ALTERNATIVES FOR GIVEN NODES FOR THE BRANCH CURRENTLY RESIDING THE ARRAY

SEGMENT VARIABLES

This section discusses all local variables that are used in the program. Note that segments are enclosed by asterisks and local procedures within the segment are labelled by leading and trailing asterisks.

```
**********************************************************
*                                                        *
*      UNIT DASSA                                        *
*                                                        *
**********************************************************
```

***** PROCEDURES NODEDISKTOARRAY, NODEARRAYTODISK *****

I	INTEGER COUNTER USED TO LOAD ALTERNATIVE VALUES TO OR FROM VRAY TO DISK FILES
X	INTEGER ARGUMENT WHICH SPECIFIES WHICH ROW IN ARRAYS IRAY, ARAY, AND VRAY THE DATA IS GOING TO OR COMING FROM

***** PROCEDURE NEXT *****

Z	INTEGER USED TO HOLD CURRENT VALUE OF LEVEL IN ARRAYS IRAY, ARAY, AND VRAY

***** PROCEDURE FIND *****

CONTFLAG	INTEGER FLAG WHICH IS RESET WHEN THE INPUT NRN DOES NOT MATCH THE CURRENT BRANCH IN ARRAY IRAY BUT A SIBLING EXISTS

```
J               INTEGER VARIABLE USED AS A TREE LEVEL INDICATOR


QUITFLAG        INTEGER FLAG WHICH EXITS PROCEDURE FIND
                        1 = END PROCESSING
                        0 = CONTINUE PROCESSING


***** PROCEDURE NODIN *****

I               INTEGER COUNTER


VAL             STRING CONSTANT CONTAINING ALL PERMISSIBLE ELEMENTS THAT
                CAN BE IN AN NRN


X               STRING VARIABLE USED TO EXAMINE THE USER ENTERED NRN
                ELEMENT BY ELEMENT


***** PROCEDURE INTTOSTRING *****

I               INTEGER ARGUMENT WHICH IS INTEGER TO BE CONVERTED INTO
                STRING TEXT


ILONG           LONG INTEGER TO CONVERT VARIABLE I (this procedure) INTO
                A STRING


***** FUNCTION STRTOREAL *****

FLAG            INTEGER FLAG
                   0 = EXAMINE TEMP AS A TENS DIGIT
                   1 = EXAMINE TEMP AS A TENTHS DIGIT


I               INTEGER COUNTER
```

ISTR	STRING INPUT ARGUMENT
K	INTEGER VARIABLE WHICH STATES THE NUMBER OF PLACES TEMP IS TO THE RIGHT OF THE DECIMAL POINT
M	INTEGER COUNTER
TEMP	STRING VARIABLE CARRYING A SINGLE OF STRING ISTR (this procedure)
TENS	REAL VARIABLE CONTAINING THE NUMERICAL VALUES OF THE TENS DIGITS IN VARIABLE ISTR (this procedure)
TENTHS	REAL VARIABLE CONTINUING THE DECIMAL VALUES OF THE TENTHS DIGITS IN VARIABLE ISTR (this procedure)
VAL	STRING CONSTANT CONTAINING ALL PERMISSIBLE ELEMENTS THAT CAN BE IN A NRN
X	REAL VARIABLE TRANSLATING THE POSITION OF TEMP (this procedure) IN VAL (this procedure) INTO A NUMBER

***** PROCEDURE CROSS *****

L	INTEGER VARIABLE WHICH IS A STORAGE VARIABLE FOR VARIABLE LVL (see GLOBAL VARIABLES)

```
**************************************************
*                                                *
*    SEGMENT PROCEDURE DUMMY                     *
*                                                *
**************************************************
```

CONTROL — USED BY PROCEDURE GRAPHICS TO INDICATE WHICH WIRE BLOCK (1..5) IS TO BE DRAWN

***** PROCEDURE GRAPHICS *****

ANS — STRING VARIABLE WHICH HOLDS THE INPUT CHARACTER FOR FURTHER PROCESSING (STRING OPERATIONS CANNOT BE DONE ON VARIABLE CHAR)

J — INTEGER VARIABLE USED TO HOLD THE NUMERIC ASCII VALUE OF A CHARACTER

X,Y — INTEGER VARIABLES USED AS SCREEN COORDINATES FOR GRAPHICS

***** PROCEDURE LABELS *****

X,Y — INTEGER ARGUMENTS WHICH INDICATE THE LOWER LEFT-HAND CORNER WHERE CHARACTERS ARE TO BE DISPLAYED

X1 — INTEGER HOLDING CELL FOR VARIABLE X (this procedure)

***** PROCEDURE DRAWBLK *****

X,Y — INTEGER ARGUMENTS WHICH INDICATE THE LOWER LEFT-HAND CORNER WHERE THE NODE BLOCKS ARE TO BE DRAWN

```
**********************************************************
*                                                        *
*      SEGMENT PROCEDURE WVLOAD                          *
*                                                        *
**********************************************************
```

ANS	STRING VARIABLE WHICH HOLDS THE INPUT CHARACTER CH (see GLOBAL VARIABLES) FOR FUTURE STRING OPERATIONS
CH1	CHARACTER VARIABLE WHICH CONTAINS RESPONSE TO VARIOUS OPTION WVC SINGLE CHARACTER RESPONSE PROMPTS
COLOR	ARRAY USED TO IDENTIFY COLORS USED IN DISPLAYS
EXITFLAG	INTEGER FLAG USED TO INDICATE NORMAL EXIT FROM PROCEDURE RDV
J,K,L	INTEGER VARIABLES USED AS COUNTERS AND ARRAY INDICES
LABEL1	STRING VARIABLE WHICH CONTAINS 'WEIGHTS' OR 'VALUE' DEPENDING ON THE SUB-OPTION SELECTED. USED IN OUTPUT DISPLAYS
OPT	CHARACTER VARIABLE WHICH CONTAINS USER INPUT SUB-OPTION UNDER OPTION WVC
PPOS	INTEGER VARIABLE WHICH CONTAINS THE CURRENT [illegible] BEING DISPLAYED (1 FIRST ALTERNATIVE, ETC)
X,Y	INTEGER VARIABLES WHICH ARE USED AS THE SCREEN COORDINATES FOR GRAPHICS. THEY OFTEN INDICATE THE LOWER LEFT-HAND CORNER FOR CHARACTERS, BARS AND BOXES
X1	INTEGER VARIABLE CONTAINING THE ABSOLUTE LEFT MARGIN OF GRAPHICS SCREEN. USED IN INTERACTIVE GRAPHICS
VRAY1	REAL ARRAY WHICH CONTAINS THE INPUTTED NEW VALUES

OF ALTERNATIVES FOR A SPECIFIED ATTRIBUTE. LOADED INTO VRAY (see GLOBAL VARIABLES) WHEN SUB-OPTION VALUE IS EXITED NORMALLY

***** PROCEDURE DRAWBAR *****

A,B — INTEGER ARGUMENT REPRESENTING THE X AND Y SCREEN COORDINATES OF A COLOR BAR

C — INTEGER ARGUMENT REPRESENTING THE ALTERNATIVE (1=FIRST ALTERNATIVE, ETC)

V — REAL ARGUMENT REPRESENTING THE WORTH VALUE OF THE ALTERNATIVE SPECIFIED BY VARIABLE C (this procedure)

***** PROCEDURE SQUARES *****

A,B — INTEGER ARGUMENTS REPRESENTING THE X AND Y SCREEN COORDINATES OF THE LOWER LEFT-HAND CORNER OF NODE BOXES

***** PROCEDURE RDWT *****

INT — INTEGER VARIABLE CONTAINING TRUNCATED RELATIVE WEIGHTS MULTIPLIED BY 100

J — INTEGER COUNTER AND ARRAY INDEX

LR — REAL VARIABLE USED AS THE SUM OF ALL INPUT WEIGHTS FOR NORMALIZATION

***** PROCEDURE CALC *****

QUITFLAG — INTEGER FLAG WHICH IS SET TO ONE WHEN A BRANCH OF A TREE HAS BEEN PROCESSED (CUMULATIVE WEIGHTS CALCULATED AND VALUES ASSIGNED)

Z — REAL VARIABLE WHICH CONTAINS INTERMEDIATE NODE VALUES

****
* *
* SEGMENT PROCEDURE NUM *
* *
****

CH1 — CHARACTER VARIABLE WHICH CONTAINS RESPONSES TO OPTION NUM SINGLE CHARACTER RESPO PROMPTS

F — FILE OF CHARACTERS USED AS AN OUTPUT BUFFER TO THE CONSOLE OR THE LINE PRINTER

J — INTEGER COUNTER AND ARRAY INDEX

OUT — STRING VARIABLE - NOT USED

***** PROCEDURE DISPLAY *****

I,J,K,L — INTEGER COUNTERS AND INDICES

***** PROCEDURE DISPLAY1 *****

I,J,K,L — INTEGER COUNTERS AND INDICES

***** PROCEDURE DISPLAY2 *****

CONTROL — INTEGER VARIABLE WHICH CONTAINS THE NODE DIGIT VALUE OF A DESCENDENT DURING GRAPHICAL DISPLAY

FLAG2 — SPECIFIES PARTICULAR GRAPHICAL DISPLAY PACKAGE
- 1 = DESCENDENT DISPLAY (BARS UNDER NODE BOX)
- 2 = PARENT DISPLAY (BARS TO THE RIGHT OF NODE BOX)
- 3 = LEGEND

J — INTEGER COUNTER AND ARRAY INDEX

X,Y — INTEGER SCREEN COORDINATES FOR A VARIETY OF GRAPHICAL ITEMS. FOR BOXES AND CHARACTERS, THEY REPRESENT THE LOWER LEFT-HAND CORNER

***** PROCEDURE SYSBLK *****

CONTROL1 — INTEGER VARIABLE WHICH IDENTIFIES ALTERNATIVES DURING COLOR BAR PRINTOUTS

Y1 — INTEGER VARIABLE WHICH CONTAINS THE SCALED HEIGHT OF COLOR BARS (45 VERTICAL SCREEN LINES IS 100 IN ALTERNATIVE VALUE)

****
*
* SEGMENT PROCEDURE SENSITIVITY
*
****

CH1 — CHARACTER VARIABLE WHICH CONTAINS RESPONSES TO SENSITIVITY SINGLE CHARACTER RESPONSE PROMPTS

DELTA — REAL VARIABLE CONTAINING THE INTERVAL SIZE BETWEEN THE "ROOT" NODE VALUE AXIS (X-AXIS) ON THE SENSITIVITY GRAPH

I,J,K,L	INTEGER COUNTERS AND ARRAY INDICES
INDX	INTEGER CONTAINING NODE DIGIT OF NODE WHEN RELATIVE WEIGHT SENSITIVITY ANALYSIS IS CONDUCTED
MAX	REAL VARIABLE CONTAINING THE MAXIMUM VALUE OF ALL THE ALTERNATIVES FOR THE ENTIRE SENSITIVITY ANALYSIS IF EXPANDED GRAPHICS IS EXERCISED. ELSE 100
MIN	REAL VARIABLE CONTAINING THE MINIMUM VALUE OF ALL THE ALTERNATIVES FOR THE ENTIRE SENSITIVITY ANALYSIS IF EXPANDED GRAPHICS IS EXERCISED. ELSE 0
SENS	STRING VARIABLE WHICH CONTAINS THE TYPE OF SENSITIVITY ANALYSIS BEING CONDUCTED (CUMWT, RELWT, OR VALUE)
SYSNAME	STRING VARIABLE HOLDING ALTERNATIVE NAME WHEN VALUE SENSITIVITY IS INVOKED
SYSNUM	INTEGER VARIABLE HOLDING SYSNAME (this procedure) POSITION IN THE SYSTEM RECORD FILE (see GLOBAL VARIABLES)
W	REAL VARIABLE NOT USED
WDELTA	REAL VARIABLE WHICH CONTAINS THE STEP SIZE OF THE PERTURBED VARIABLE (CUMLATIVE WEIGHT, RELATIVE WEIGHT OR ATTRIBUTE VALUE)
WHOLD	REAL ARRAY HOLDING ALTERNATIVE VALUES BY ALTERNATIVES IN COLUMNS AND 11 VARIABLE PERTURBATIONS (CUMULATIVE WEIGHT, RELATIVE WEIGHT, OR ATTRIBUTE VALUE) IN THE ROWS. ROW 0 HOLDS THE MINIMUM PERTURBATION, ROW 10 HOLDS THE MAXIMUM
WMAX	REAL VARIABLE CONTAINING THE MAXIMUM POSSIBLE INPUT VALUE (1.0 FOR WEIGHTS AND 100 FOR VALUES). IF WMAX1 (this procedure) IS LESS THAN THESE MAXIMUMS AND

GREATER THAN WMIN, WMAX IS WMAX1

WMAX1 — REAL VARIABLE WHICH CONTAINS THE USER INPUTTED MAXIMUM VALUE TO BE USED IN THE SENSITIVITY ANALYSIS

WMIN — REAL VARIABLE WHICH CONTAINS THE USER INPUTTED MINIMUM VALUE TO BE USED IN THE SENSITIVITY ANALYSIS

X — REAL VARIABLE WHICH CONTAINS THE VALUE OF THE "ROOT" NODE FOR DISPLAY ON THE X-AXIS OF THE SENSITIVITY GRAPH

***** PROCEDURE CALCARRAY *****

CONTROL — INTEGER VARIABLE WHICH INVOKES THE NECESSARY SENSITIVITY ANALYSIS DEPENDING ON ITS VALUE
- 1 = CUMULATIVE WEIGHT
- 2 = RELATIVE WEIGHT
- 3 = VALUE

NEWSUM — REAL VARIABLE NOT USED

SUM — REAL VARIABLE CONTAINING ONE MINUS THE RELATIVE WEIGHT OF THE PERTURBED NODE WHICH IS USED TO INSURE PROPORTIONALITY AMONG THE SIBLINGS DURING RELATIVE WEIGHT SENSITIVITY ANALYSIS

VTEMP — REAL VARIABLE NOT USED

***** PROCEDURE HEADERS *****

I — INTEGER ARCUMENT WHICH SELECTS VARIOUS OUTPUT HEADERS. WHEN EQUAL TO TWO, I IS ALSO USED AS AN ARRAY INDEX

***** PROCEDURE TABDISPLAY *****

A	STRING VARIABLE USED TO DETERMINE REGRET (PUTTING THE ASTERISK ON THE LOWEST VALUE) OR NOT REGRET (PUTTING THE ASTERISK ON THE HIGHEST VALUE)
F	FILE OF CHAR USED AS AN OUTPUT BUFFER TO CONSOLE OR LINE PRINTER
STAR	INTEGER VARIABLE CONTAINING THE NUMBER OF THE ALTERNATIVE IN ARRAY WHOLD (see SEGMENT PROCEDURE SENSITIVITY) WHICH HAS THE LOWEST VALUE (IF VARIABLE A (this procedure) IS 'R') OR THE HIGHEST VALUE (IF VARIABLE A (this procedure) IS NOT 'R'). AN ASTERISK WILL BE PLACED BY THIS VALUE
W	REAL VARIABLE CONTAINING CURRENT LOWEST VALUE OR CURRENT HIGHEST VALUE DEPENDING ON VARIABLE A (this procedure) DURING THE SEARCH FOR THE LOWEST OR HIGHEST VALUE OF A GIVEN ROW OF WHOLD (see SEGMENT PROCEDURE SENSITIVITY)

***** PROCEDURE GRAPH *****

I1,J1	INTEGER VARIABLES WHICH CONTAIN THE X AND Y SCREEN COORDINATES WHERE THE LINES OF THE ALTERNATIVES ARE TO BE DRAWN ON THE SENSITIVITY GRAPH
K	INTEGER COUNTER
COLOR	ARRAY OF AVAILABLE COLORS USED IN THE SENSITIVITY GRAPH

```
**********************************************************
*                                                        *
*      SEGMENT PROCEDURE READSYSTEMS                     *
*                                                        *
**********************************************************
```

***** PROCEDURE NEW *****

SYSLABEL — STRING VARIABLE USED TO READ IN ALTERNATIVE LABELS

***** PROCEDURE DELSYS *****

SYS — STRING ARRAY CONTAINING ALL CURRENT ALTERNATIVE NAMES PRESENTLY EVALUATED IN HIERARCHICAL STRUCTURE

J — INTEGER COUNTER

```
**********************************************************
*                                                        *
*     SEGMENT PROCEDURE MODPRU                           *
*                                                        *
**********************************************************
```

***** PROCEDURE PRUNE *****

FLAG1 — INTEGER FLAG (IN PROCEDURE GRAPHICS) WHICH GOES FROM 0 TO 1 WHEN THE LAST NODE IS REACHED AFTER PRUNING. USED TO COUNT NEW NUMBER OF NODES.

J,K — INTEGER ARRAY INDEX AND COUNTERS

NEWLVL — INTEGER VARIABLE CONTAINING THE NUMBER OF LEVELS IN THE HIERARCHICAL STRUCTURE AFTER PRUNING

NEWNNODES — INTEGER VARIABLE WHICH CONTAINS THE NUMBER OF NODES IN THE TREE AFTER PRUNING

PRAY — INTEGER ARRAY WHICH CONTAINS ALL LINKS FOR ALL THE NODES IN THE HIERARCHICAL TREE. NODE NUMBER RUNS ALONG THE COLUMNS.

- ROW 1 THE NUMBER 1..NUMBER OF NODES (0 MEANS NO NODE)
- ROW 2 THE NUMBER 1..NUMBER OF NODES (ALTERED DURING PRUNING - 0 IF NO NODE)
- ROW 3 NODE DIGIT
- ROW 4 DOWNLINK
- ROW 5 CROSSLINK
- ROW 6 BACKLINK

PRNUM — INTEGER VARIABLE CONTAINING THE NODE RECORD NUMBER OF NODE TO BE PRUNED

***** PROCEDURE MODIFY *****

IQUIT — SET FROM 0 TO 1 WHEN THE USER WISHES TO EXIT THE OPTION MOD PRIOR TO ENTERING A NEW LABEL ENTRY

VII. Program Structure

The program consists of several segments which allows greater program capability. The following list show all the procedures that are currently in the DASS program. When the procedure is indented, it means that the procedure is local to the program above it.

```
PROGRAM DASS;
 UNIT DASSA;
   PROCEDURE NODEDISKTOARRAY;
   PROCEDURE NODEARRAYTODISK;
   PROCEDURE NEXT;
   PROCEDURE FIND;
   PROCEDURE NODIN;
   PROCEDURE PRENEX;
   PROCEDURE ANSWERTOQUESTION;
   PROCEDURE MASTERNODESETUP;
   PROCEDURE PRETOT;
   PROCEDURE INTTOSTRING;
   PROCEDURE NUMTOSTRING;
   FUNCTION  STRTOREAL;
   PROCEDURE CROSS;

SEGMENT PROCEDURE DUMMY;
   PROCEDURE GRAPHICS;
     PROCEDURE LABELS;
     PROCEDURE DRAWBLK;
   PROCEDURE LABELATTRIBUTES;
   PROCEDURE READTITLE;
   PROCEDURE SPAN;
   PROCEDURE INITIALIZE;
   PROCEDURE SELECTFILE;

SEGMENT PROCEDURE WVLOAD;
   PROCEDURE RDV;
     PROCEDURE DRAWBAR;
     PROCEDURE SQUARES;
     PROCEDURE SETUUP;
   PROCEDURE REWT;
   PROCEDURE CALC;
   PROCEDURE WVLOAD1;
```

```
SEGMENT PROCEDURE NUM;
  PROCEDURE OUTDEVICE;
  PROCEDURE NUMERICREVIEW;
    PROCEDURE NEWPG;
  PROCEDURE DISPLAY;
    PROCEDURE DISPLAY1;
    PROCEDURE DISPLAY2;
      PROCEDURE WTS;
      PROCEDURE DRAWBLK;
      PROCEDURE SYSBLK;
    PROCEDURE DISPLAY0;

SEGMENT PROCEDURE SENSTIVITY;
  PROCEDURE ANYKEY;
  PROCEDURE WARNING;
  PROCEDURE DETERMINENODE;
    PROCEDURE SENVALUE;
  PROCEDURE CALCARRAY;
  PROCEDURE HEADERS;
  PROCEDURE TABDISPLAY;
  PROCEDURE GRAPH;
    PROCEDURE GRAPHHEADER;

SEGMENT PROCEDURE READSYSTEMLABELS;
  PROCEDURE NEW;
  PROCEDURE DELSYS;
  PROCEDURE ADDSYS;

SEGMENT PROCEDURE MODIFY;
  PROCEDURE PRUNE;
    PROCEDURE WDOT;
    PROCEDURE RELINK;
    PROCEDURE COMPRESS;
    PROCEDURE REWRITE;
  PROCEDURE MODIFY;

PROCEDURE STAT;
```

In the program listing that follows, each program will contain a summary enclosed with asterisks (*). In the summary, the name of the procedure will be repeated. In addition, a brief discussion of the purpose or nature of the procedure is presented. Following that, a list

of procedures which call the procedure will be presented. The format lists the segment the call is from and then specifies the procedure name(s) in parentheses. Next, is a list of procedure name(s) that the procedure calls during its execution. Finally, a listing of variables is provided, being divided into two groups. The first group (labelled USED) indicates those variables which are used but not changed while the procedure is being executed. The second group (labelled MODIFIED) are those values which may change during execution of the procedure. Note that if the procedure calls another procedure, variables which change values under the second call will not appear in the original variable list.

```
                                                    (* PROGRAM DASS *)

PROGRAM DASS; (*$C+,S+,LPRINTER:*)

(* ************************************************************
*                                                              *
*  PROGRAM DASS                                                *
*  USE:  MAIN PROGRAM FOR THE DECESION ANALYSIS SUPPORT SYSTEM. *
*        CONSISTS OF ONE UNIT (DASSA), SIX SEGMENTS (DUMMY,    *
*        WVLOAD,NUM,SENSITIVITY,READSYSTEMLABELS,MODPRU) AND   *
*        ONE PROCEDURE (STAT).  GLOBAL VARIABLES ARE FOUND IN  *
*        UNIT DASSA.  SEGMENT VARIABLES ARE FOUND IN THE       *
*        VARIOUS SEGMENTS AND SUB-PROCEDURES IN THE RESPECTIVE *
*        SEGMENTS.                                             *
*        THE MAIN PROGRAM GENERALLY SETS UP THE VARIABLES      *
*        CRITICAL IN RUNNING OPTIONS AND SOLICITS THE COMMAND  *
*        OPTIONS.  IN EXECUTING THE DON OPTION, THE MAIN       *
*        CLOSES ALL OPEN FILES AND ELEGANTLY RETURNS CONTROL   *
*        TO THE DISK OPERATING SYSTEM.                         *
*        **** NOTE ****                                        *
*        UNIT DASSA MUST BE COMPILED SEPARATELY AND LOADED INTO*
*        THE SYSTEM.LIBRARY (UNDER APPLE II PASCAL OPERATIONS) *
*        FOR THE REST OF THE PROGRAM TO BE COMPILED CORRECTLY. *
*        IN ADDITION, OPTIONS ALLOWING GOTO AND MEMORY SWAPPING*
*        MUST BE PRESENT.                                      *
*                                                              *
*        PROGRAM USES THREE UNITS:                             *
*              UNIT DASSA  CONTAINS FUNCTIONS NECESSARY TO RUN *
*                          THE DASS PROGRAM AND CAN BE         *
*                          MODIFIED.                           *
*              UNIT TURTLEGRAPHICS CONTAINS FUNCTIONS          *
*                          NECESSARY TO CONDUCT THE GRAPHICS   *
*                          CAPABILITY OF THE APPLE II MICRO-   *
*                          COMPUTER. THIS UNIT CANNOT BE       *
*                          MODIFIED BY THE USER.               *
*              UNIT APPLESTUFF CONTAINS THE FUNCTION WHICH     *
*                          ALLOWS THE COMPUTER TO WAIT UNTIL A *
*                          SINGLE KEY IS DEPRESSED (KEYPRESS)  *
*                          THIS UNIT CANNOT BE MODIFIED BY THE *
*                          USER.                               *
*                                                              *
*  CALLED BY: (none)                                           *
*  ROUTINES CALLED: SEGMENT PROCEDURE DUMMY                    *
*                   SEGMENT PROCEDURE WVLOAD                   *
*                   SEGMENT PROCEDURE NUM                      *
*                   SEGMENT PROCEDURE SENSITIVITY              *
*                   SEGMENT PROCEDURE READSYSTEMLABELS         *
*                   SEGMENT PROCEDURE MODPRU                   *
*                   PROCEDURE STAT                             *
*  VARIABLES:                                                  *
```

```
                                                    (* PROGRAM DASS *)

*       USED: TENSTRING,TITLE,LATT,NNODES,NDEEP,NSYS,FLAG       *
*                    (see UNIT DASSA)                           *
*       MODIFIED: CMD,NODERATIONALE,NODETITLE,CELLNUMBER,       *
*                 NRNDIGIT,DOWNLINK,CROSSLINK),FLAG,NSYS,NFLAG,*
*                 NDEEP,COMMENTSTRING,LABELSTRING,IRAY          *
*                    (see UNIT DASSA)                           *
*                                                               *
*****************************************************************)

USES DASSA, APPLESTUFF, TURTLEGRAPHICS;
```

```
                                                        (* UNIT DASSA *)

(*$C+,S+ *)
UNIT DASSA;
(* ************************************************************
*                                                              *
*   UNIT DASSA                                                 *
*   USE:  CONTAINS THE GLOBAL VARIABLE LISTING AND             *
*         PROCEDURES WHICH ARE UNIVERSAL TO ALL                *
*         SEGMENT PROCEDURES.  THE UNIT IS NOT CALLED          *
*         EXPLICITLY, BUT IS USED WHENEVER THE PROCEDURES      *
*         IN IT ARE CALLED.                                    *
*   CALLED BY: (none)                                          *
*   ROUTINES CALLED: (none)                                    *
*   VARIABLES:                                                 *
*       USED:  (none)                                          *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************** *)

INTERFACE

  CONST
    MAXSYSTEMS=5;
    MAXLABELSIZE=10;
    MAXCOMMENTSIZE=60;
    MAXARRAYSIZE=10;
    MAXNUMBEROFNODES=101;
    TENSTRING='          ';

  VAR
    COMMENTSTRING,LABELSTRING,FILENAME,ANSWER,FILEOFSYSTEMS,DISKNAME : STRING;
    CH:CHAR;
    NSYS,I,RECORDID,NLVLS : INTEGER;
    CMD,ISTR,LATT,TITLE,QUESTION : STRING;
    ICONT,IFADD,L,LVL,
    FLAG,NFLAG,NCROSS,IFIND,ITOTL,NDEEP,NDIFF,NNRN,NNODES,OLDCELLNUM :
      : INTEGER;

    NODELABEL : ARRAY[0..MAXARRAYSIZE] OF STRING;

    IRAY : ARRAY[0..MAXARRAYSIZE,0..5] OF INTEGER;
    ARAY : ARRAY[0..MAXARRAYSIZE,1..2] OF REAL;
    VRAY : ARRAY[0..MAXARRAYSIZE,1..MAXSYSTEMS] OF REAL;

    NRNVECTOR : ARRAY [0..MAXARRAYSIZE] OF INTEGER;

    NODE:FILE OF RECORD
      NODETITLE : STRING[MAXLABELSIZE];
      CELLNUMBER, NRNDIGIT, DOWNLINK, CROSSLINK, BACKLINK : INTEGER;
```

```
                                                    (* UNIT DASSA *)

      RELWEIGHT, CUMWEIGHT : REAL;
      SYSTEMVALUES : ARRAY [1..MAXSYSTEMS] OF REAL;
      RATIONALE : STRING[MAXCOMMENTSIZE];
      END;

    SYSTEMS:FILE OF RECORD
      SYSTEMNAME : STRING[MAXLABELSIZE]
      END;

(* PROCEDURES IN DASSA *)

PROCEDURE NODEDISKTOARRAY(X:INTEGER);
PROCEDURE NODEARRAYTODISK(X:INTEGER);
PROCEDURE NEXT;
PROCEDURE FIND;
PROCEDURE NODIN;
PROCEDURE PRENEX;
PROCEDURE ANSWERTOQUESTION;
PROCEDURE MASTERNODESETUP;
PROCEDURE PRETOT;
PROCEDURE INTTOSTRING(I:INTEGER);
PROCEDURE NUMTOSTRING(X:REAL);
FUNCTION  STRTOREAL(ISTR:STRING):REAL;
PROCEDURE CROSS;


IMPLEMENTATION
```

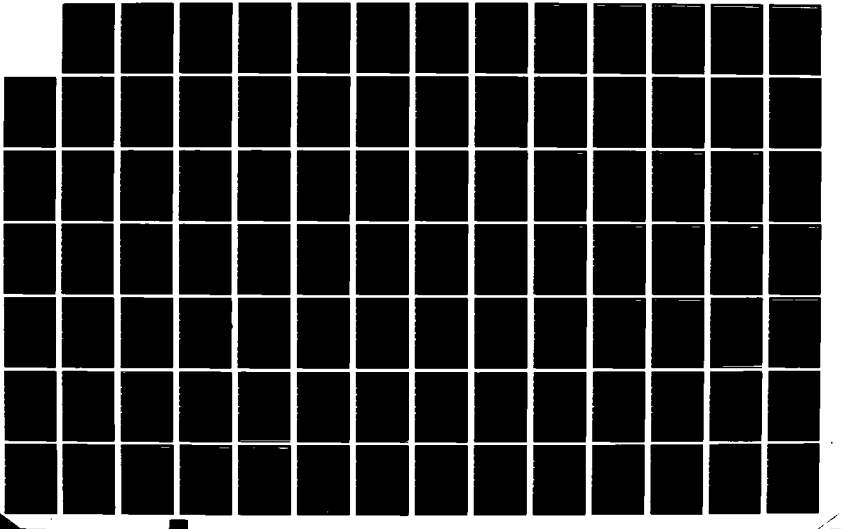
```
                                                        (* UNIT DASSA *)

PROCEDURE NODEDISKTOARRAY;

(* ************************************************************
*                                                              *
*    PROCEDURE NODEDISKTOARRAY                                 *
*    USE:   TO TRANSFER DATA READ IN FROM RECORD               *
*           NODE TO THE IRAY  ARAY, AND VRAY.                  *
*           REQUIRES AN ARG ENT WHICH IS EQUAL TO              *
*           THE PARTICULAR ROW (SPECIFIC NODE AT A             *
*           SPECIFIC LEVEL IN THE TREE) IN THE ARRAYS.         *
*    CALLED BY: UNIT DASSA (NEXT, CROSS, FIND)                 *
*               SEGMENT PROCEDURE DUMMY (SELECTFILE)           *
*    ROUTINES CALLED: (none)                                   *
*    VARIABLES:                                                *
*         USED: NODE(CELLNUMBER,NRNDIGIT,DOWNLINK              *
*                 CROSSLINK,BACKLINK,RELWEIGHT,CUMWEIGHT       *
*                 SYSTEMVALUES(1..NSYS),NODETITLE)             *
*               NSYS (see UNIT DASSA)                          *
*         MODIFIED:  NODELABEL, IRAY, ARAY, VRAY (see          *
*                     UNIT DASSA)                              *
*                    I (see PROCEDURE NODEDISKTOARRAY)         *
*                                                              *
************************************************************* *)

  VAR
    I:INTEGER;

BEGIN
 WITH NODE^ DO
  BEGIN
   NODELABEL[X]:=NODETITLE;
   IRAY[X,0]:=CELLNUMBER;
   IRAY[X,1]:=NRNDIGIT;
   IRAY[X,2]:=DOWNLINK;
   IRAY[X,3]:=CROSSLINK;
   IRAY[X,4]:=BACKLINK;
   ARAY[X,1]:=RELWEIGHT;
   ARAY[X,2]:=CUMWEIGHT;

   FOR I:=1 TO NSYS DO VRAY[X,I]:=SYSTEMVALUES[I];
  END
END;    (* END TRANSFERRING RECORD NODE DATA FROM DISK TO ARRAYS *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE NODEARRAYTODISK;

(* ************************************************************
*                                                              *
*    PROCEDURE NODEARRAYTODISK                                 *
*    USE:   TO TRANSFER DATA READ IN FROM ARRAYS IRAY,         *
*           ARAY, VRAY, AND NODETITLE TO RECORD NODE.          *
*           REQUIRES AN ARGUMENT WHICH IS EQUAL TO             *
*           THE PARTICULAR ROW (SPECIFIC NODE AT A             *
*           SPECIFIC LEVEL IN THE TREE) IN THE ARRAYS.         *
*    CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN)                 *
*               SEGMENT PROCEDURE WVLOAD (RDWT,RDV,CALC)       *
*    ROUTINES CALLED: (none)                                   *
*    VARIABLES:                                                *
*         USED: NODELABEL, IRAY, ARAY, VRAY,  NSYS             *
*                 (see UNIT DASSA)                             *
*         MODIFIED: NODE[CELLNUMBER,NRNDIGIT,DOWNLINK,         *
*                   CROSSLINK,BACKLINK,RELWEIGHT,CUMWEIGHT     *
*                   SYSTEMVALUES[1..NSYS],NODETITLE]           *
*                     (see UNIT DASSA)                         *
*                   I (see PROCEDURE NODEARRAYTODISK)          *
*                                                              *
************************************************************** *)

  VAR
    I : INTEGER;

BEGIN
 WITH NODE^ DO
  BEGIN
   NODETITLE:=NODELABEL[X];
   CELLNUMBER:=IRAY[X,0];
   NRNDIGIT:=IRAY[X,1];
   DOWNLINK:=IRAY[X,2];
   CROSSLINK:=IRAY[X,3];
   BACKLINK:=IRAY[X,4];
   RELWEIGHT:=ARAY[X,1];
   CUMWEIGHT:=ARAY[X,2];

   FOR I:=1 TO NSYS DO SYSTEMVALUES[I]:=VRAY[X,I];
  END
END;  (* END TRANSFERRING RECORD NODE DATA FROM ARRAYS TO DISK *)
```

```
                                                        (* UNIT DASSA *)
 PROCEDURE NEXT;

 (* ************************************************************
 *                                                              *
 *  PROCEDURE NEXT                                              *
 *  USE: THIS ROUTINE IS USED TO PERFORM A DEPTH-FIRST          *
 *       SEARCH OF THE TREE, NODE BY NODE.  THE VARIABLES       *
 *       NECESSARY ARE INITIALIZED BY PROCEDUUES FRENEX         *
 *       AND PRETOT, THEN THE ROUTINE WILL PERFORM THE          *
 *       THE DEPTH-FIRST SEARCH, NODE BY NODE.  A NEW NODE      *
 *       IS ACCESSED EACH TIME THAT PROCEDURE NEXT IS           *
 *       CALLED.                                                *
 *  CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN)                   *
 *             SEGMENT PROCEDURE WVLOAD (WVLOAD1)               *
 *  ROUTINES CALLED: UNIT DASSA (NODEDISKTOARRAY)               *
 *  VARIABLES:                                                  *
 *       USED: NDIFF (see UNIT DASSA)                           *
 *       MODIFIED: LVL,IRAY,ICONT (see UNIT DASSA)              *
 *                 Z (see PROCEDURE NEXT)                       *
 *                                                              *
 ************************************************************** *)

 LABEL 1;

 VAR
  Z : INTEGER;

BEGIN
  IF(IRAY(LVL,2] (= 0)

  THEN
   BEGIN
1:  IF((LVL(=1)OR(LVL=NDIFF))
      THEN ICONT:=0

      ELSE
       BEGIN
        Z:=LVL;
        IF(IRAY(LVL,3](=0)
           THEN
            BEGIN
             LVL:=LVL-1;
             GOTO 1
            END

           ELSE
            BEGIN
```

AD-A101 140 ENHANCED DECISION ANALYSIS SUPPORT SYSTEM(U) AIR FORCE 3/4
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING D B LEE MAR 81 AFIT/GST/OS/81M-8

UNCLASSIFIED F/G 9/2 NL



1.0 | 2.8 | 2.5 | 3.2 | 2.2 | 3.6 | 4.0 | 2.0 | 1.1 | 1.8 | 1.25 | 1.4 | 1.6

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
                                                    (* UNIT DASSA *)

                SEEK(NODE,IRAY[LVL,3]);
                GET(NODE),
                NODEDISKTOARRAY(Z);
                IRAY[Z,5]:=IRAY[LVL,0];
                LVL:=Z
               END
          END
      END

      ELSE
       BEGIN
        Z:=LVL+1;
        IF(IRAY[Z,0]=IRAY[LVL,2])
          THEN LVL:=Z

          ELSE
           BEGIN
            SEEK(NODE,IRAY[LVL,3]);
            GET(NODE);
            NODEDISKTOARRAY(Z);
            IRAY[Z,5]:=IRAY[LVL,0];
            LVL:=Z
           END
       END

END;    (* END NEXT *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE FIND;

(* ************************************************************
*                                                              *
*   PROCEDURE FIND                                             *
*   USE:  THIS PROCEDURE SEARCHES THE TREE FOR A MATCH         *
*         TO THE INPUT NODE INPUTTED BY PROCEDURE KODIN        *
*         UNIT DASSA.                                          *
*   CALLED BY: UNIT DASSA (NODIN)                              *
*   ROUTINES CALLED: UNIT DASSA (NODEDISKTOARRAY)              *
*   VARIABLES:                                                 *
*        USED: IFADD,IFIND,NDIFF,IRAY,NRNVECTOR                *
*                (see UNIT DASSA)                              *
*        MODIFIED: IFADD,IFIND,NDIFF,ICONT,LVL                 *
*                    (see UNIT DASSA)                          *
*                  J,Z,CONTFLAG,QUITFLAG                       *
*                    (see PROCEDURE FIND)                      *
*                                                              *
************************************************************ *)
  VAR
   CONTFLAG,QUITFLAG,J,Z : INTEGER;

BEGIN
 QUITFLAG:=0;
 IFADD:=2;
 IFIND:=1;
 NDIFF:=0;
 J:=1;

 REPEAT
  BEGIN
   CONTFLAG:=1;
   IF(IRAY[J,1]=NRNVECTOR[J])
    THEN
     BEGIN
      IF(J=NLVLS)
       THEN
        BEGIN
         ICONT:=1;
         LVL:=J;
         QUITFLAG:=1
        END

       ELSE
        BEGIN
         IF(IRAY[J,2]<1)
          THEN
           BEGIN
```

```
                                                        (* UNIT DASSA *)

            IFADD:=2;
            LVL:=J;
            QUITFLAG:=1
           END

          ELSE
           BEGIN  Z:=J+1;
            IF(((IRAY[Z,1]<>NRNVECTOR[Z])AND(IRAY[Z,0]<>IRAY[J,2]))OR
               (IRAY[Z,5]<>IRAY[J,0]))

               THEN
                BEGIN
                 SEEK(NODE,IRAY[J,2]);
                 GET(NODE);
                 NODEDISKTOARRAY(Z);
                 IRAY[Z,5]:=IRAY[J,0]
                END;

            J:=J+1
           END
        END
      END

   ELSE
    BEGIN
     IF(IRAY[J,3]>0)
      THEN
       BEGIN
        SEEK(NODE,IRAY[J,3]);
        GET(NODE);
        NODEDISKTOARRAY(J);
        CONTFLAG:=0
       END

      ELSE
       BEGIN
        IFADD:=3;
        LVL:=J;
        QUITFLAG:=1
       END
    END
  END
  UNTIL(((J)NLVLS)OR(QUITFLAG<>0))AND(CONTFLAG=1))

END;        (* END PROCEDURE FIND *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE NODIN;

(* ********************************************************
*                                                         *
*  PROCEDURE NODIN                                        *
*  USE:  READS IN USER INPUTTED NODE REFERENCE NUMBER.    *
*        HAS THE CAPABILITY TO FILTER NON-NUMERIC         *
*        INPUTS.  IF AT LEAST ONE NUMBER HAS BEEN         *
*        ENTERED, PROCEDURE FIND WILL BE INITIATED.       *
*  CALLED BY: UNIT DASSA (PRENEX)                         *
*             SEGMENT PROCEDURE NUM (DISPLAY)             *
*             SEGMENT PROCEDURE MODPRU (MODIFY)           *
*  ROUTINES CALLED: UNIT DASSA (FIND)                     *
*  VARIABLES:                                             *
*       USED: VAL (see PROCEDURE NODIN)                   *
*       MODIFIED: NLVLS,ICONT,NRNVECTOR,,ANSWER           *
*                    (see UNIT DASSA)                     *
*                 I,X (see PROCEDURE NODIN)               *
*                                                         *
******************************************************** *)

 CONST
  VAL='0123456789';

 VAR
  I:INTEGER;
  X:STRING;

BEGIN
 WRITELN(OUTPUT);
 WRITELN(OUTPUT,'ENTER...NRN?');
 READLN(INPUT,ANSWER);
 ANSWER:=CONCAT(' ',ANSWER);
 NLVLS:=0;
 ICONT:=0;
 NRNVECTOR[0]:=0;

 FOR I:=1 TO LENGTH(ANSWER) DO
  BEGIN
   X:=COPY(ANSWER,I,1);
   IF(POS(X,VAL)<>0)
    THEN
     BEGIN
      NLVLS:=NLVLS+1;
      NRNVECTOR[NLVLS]:=POS(X,VAL)-1;
     END
  END;
```

```
                                                    (* UNIT DASSA *)
    IF(NLVLS)0)
     THEN FIND

END;    (*  END PROCEDURE NODIN *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE PRENEX;

(* ********************************************************
*                                                         *
*  PROCEDURE PRENEX                                       *
*  USE:  ELICITS A NODE REFERENCE NUMBER FROM THE USER    *
*        AND IF VALID, STARTS THE TREE TRAVERSAL AT       *
*        NODE.                                            *
*  CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN)              *
*             SEGMENT PROCEDURE WVLOAD (WVLOAD1)          *
*             SEGMENT PROCEDURE NUM                       *
*             SEGMENT PROCEDURE NODPRU (PRUNE)            *
*  RCUTINES CALLED:  UNIT DASSA (NODIN)                   *
*  VARIABLES:                                             *
*       USED: LVL,ICONT (see UNIT DASSA)                  *
*       MODIFIED: NDIFF,ITOTL (see UNIT DASSA)            *
*                                                         *
******************************************************** *)

BEGIN
 NODIN;
 IF(ICONT()0)
  THEN
   BEGIN
    NDIFF:=LVL;
    ITOTL:=0
   END
END;      (* END PROCEDURE PRENEX *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE ANSWERTOQUESTION;

(* ************************************************************
*                                                              *
*  PROCEDURE ANSWERTOQUESTION                                  *
*  USE:  WRITES OUT A PROMPT CARRIED BY THE VARIABLE           *
*        QUESTION AND ACCEPTS A RESPONSE THROUGHT THE          *
*        VARIABLE ANSWER. LOOPS UNTIL THE LETTER Y OR N        *
*        IS DEPRESSED FOLLOWED BY A CARRIAGE RETURN.           *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD (RDWT)                  *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: QUESTION (see UNIT DASSA)                        *
*       MODIFIED: ANSWER (see UNIT DASSA)                      *
*                                                              *
************************************************************** *)

BEGIN
 REPEAT
  BEGIN
   WRITE(OUTPUT,QUESTION);
   READLN(INPUT,ANSWER)
  END
 UNTIL((ANSWER='Y')OR(ANSWER='N'))
END;       (* END PROCEDURE ANSWERTOQUESTION *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE MASTERNODESETUP;

(* ************************************************************
*                                                              *
*  PROCEDURE MASTERNODESETUP                                   *
*  USE:  INITIALIZES THE VARIABLES AND ARRAYS WHEN CREATING    *
*        A NEW TREE STRUCTURE                                  *
*  CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN,INITIALIZE)        *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: (none)                                           *
*       MODIFIED: NNODES,NODELABEL,IRAY,ARAY,LVL               *
*                 (see UNIT DASSA))                            *
*                                                              *
************************************************************** *)

BEGIN
 NNODES:=0;
 NODELABEL[0]:='MASTER';
 FOR I:=0 TO 4 DO IRAY[0,I]:=0;
 ARAY[0,1]:=0;
 ARAY[0,2]:=0;
 LVL:=0
END;        (* END PROCEDURE MASTERNODESETUP *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE PRETOT;

(* ************************************************************
*                                                              *
*  PROCEDURE PRETOT                                            *
*  USE:  SETS ARRAY POINTERS TO THE TOP OF THE TREE STRUCTURE  *
*        TO PREPARE FOR AN ENTIRE TRAVERSAL.                   *
*  CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN)                   *
*             SEGMENT PROCEDURE WVLOAD (WVLOAD1.CALC)          *
*             SEGMENT PROCEDURE NUM                            *
*             SEGMENT PROCEDURE MODPRU (PRUNE)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: MNODES (see UNIT DASSA)                          *
*       MODIFIED: ICONT,NDIFF,LVL,ITOTL  (see UNIT DASSA)      *
*                                                              *
************************************************************ *)

BEGIN
 ICONT:=1;
 NDIFF:=1;
 LVL:=1;
 ITOTL:=1;
 IF(MNODES(1) THEN ICONT:=0
END;        (* END PROCEDURE PRETOT *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE INTTOSTRING;

(* ************************************************************************
*                                                                         *
*  PROCEDURE INTTOSTRING                                                  *
*  USE:  TRANSLATE AN INTEGER VALUE INTO A CHARACTER STRING               *
*        SO THAT IT CAN BE PRINTED ON EITHER THE CONSOLE OR THE           *
*        PRINTER IN EITHER THE TEXT OR GRAPHIC MODE.                      *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD (SETUP)                            *
*             SEGMENT PROCEDURE NUM (NUMERICREVIEW,DISPLAY1)              *
*             SEGMENT PROCEDURE SENSITIVITY (HEADERS)                     *
*                                                                         *
*  ROUTINES CALLED: (none)                                                *
*  VARIABLES:                                                             *
*       USED: I (see PROCEDURE INTTOSTRING)                               *
*       MODIFIED: ISTR (see UNIT DASSA)                                   *
*                 ILONG (see PROCEDURE INTTOSTRING)                       *
*                                                                         *
************************************************************************ *)

 VAR
  ILONG:INTEGER;

BEGIN
 ILONG:=I;
 STR(ILONG,ISTR);
 ISTR:=COPY(ISTR,LENGTH(ISTR),1);
END;        (* END PROCEDURE INTTOSTRING *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE NUMTOSTRING;

(* ************************************************************
*                                                              *
*  PROCEDURE NUMTOSTRING                                       *
*  USE:  TRANSLATE AN REAL NUMBER VALUE INTO A CHARACTER STRING *
*        SO THAT IT CAN BE PRINTED ON EITHER THE CONSOLE OR THE *
*        PRINTER IN EITHER THE TEXT OR GRAPHIC MODE            *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD (RDV)                   *
*             SEGMENT PROCEDURE NUM (DISPLAY1,WTS)             *
*             SEGMENT PROCEDURE SENSITIVITY (GRAPH,GRAPHHEADERS)*
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: X (see PROCEDURE NUMTOSTRING)                    *
*       MODIFIED: ISTR (see UNIT DASSA)                        *
*                 ILONG (see PROCEDURE NUMTOSTRING             *
*                                                              *
*                                                              *
************************************************************** *)


   VAR
    ILONG : INTEGER;

   BEGIN
    ILONG:=TRUNC(100*X);
    STR(ILONG,ISTR);
    IF (ILONG<100) THEN ISTR:=CONCAT('0',ISTR);
    INSERT ('.',ISTR,PRED(LENGTH(ISTR)));
    ISTR:=CONCAT('      ',ISTR);
    ISTR:=COPY(ISTR,LENGTH(ISTR)-5,6)
   END;
```

```
                                                        (* UNIT DASSA *)

FUNCTION STRTOREAL;

(* ************************************************************
*                                                              *
*  FUNCTION STRTOREAL                                          *
*  USE: TRANSLATE A STRING VARIABLE AND RETURNS A REAL VALUE.  *
*       CAPABLE OF FILTERING NON-NUMERIC CHARACTERS FROM THE   *
*       INPUT STRING.                                          *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD (RDV,RDVT)              *
*             SEGMENT PROCEDURE SENSITIVITY(DETERMINENODE)     *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: ISTR (see UNIT DASSA)                            *
*             VAL (see FUNCTION  STRTOREAL)                    *
*       MODIFIED: FLAG,K,TENS,TENTHS,I,TEMP,J,X,M              *
*                 (see  FUNCTION STRTOREAL)                    *
*                                                              *
************************************************************** *)

 CONST
  VAL='0123456789.';
 VAR
  TENS,TENTHS,X:REAL;
  I,J,K,M,FLAG:INTEGER;
  TEMP:STRING;

BEGIN
 FLAG:=0;
 K:=1;
 TENS:=0;
 TENTHS:=0;
 FOR I:=1 TO LENGTH(ISTR) DO
  BEGIN
   TEMP:=COPY(ANSWER,I,1);
   J:=POS(TEMP,VAL);
   IF(J<>0)
    THEN
     BEGIN
      IF(J=11)
       THEN FLAG:=1

       ELSE
        IF(FLAG=1)
         THEN
          BEGIN
           X:=J-1;
           FLAG:=1;
           FOR M:=1 TO X DO X:=X/10;
```

```
                                                        (* UNIT DASSA *)

            TENTHS:=TENTHS+X;
            K:=K+1
           END

          ELSE
           BEGIN
            X:=J-1;
            TENS:=TENS*10+X
           END
      END
     END;
 STRTOREAL:=TENS+TENTHS
END;            (* END FUNCTION STRTOREAL *)
```

```
                                                        (* UNIT DASSA *)

PROCEDURE CROSS;

(* ***************************************************************
*                                                                 *
*  PROCEDURE CROSS                                                *
*  USE: PLACES ALL THE NODES OF A DESCENDENT SPAN IN THE ARRAYS   *
*       UNDERNEATH A GIVEN NODE.                                  *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY,DISPLAY0)            *
*             SEGMENT PROCEDURE SENSITIVITY (DETERMINENODE)       *
*  ROUTINES CALLED: UNIT DASSA(NODEDISKTOARRAY)                   *
*  VARIABLES:                                                     *
*       USED:  (none)                                             *
*       MODIFIED: LVL,NCROSS,IFADD,IRAY                           *
*                  (see UNIT DASSA)                               *
*                 L (see PROCEDURE CROSS)                         *
*                                                                 *
*************************************************************** *)
 VAR
  L:INTEGER;
BEGIN
 L:=LVL;
 NCROSS:=0;
 IFADD:=2;

 WHILE(IRAY[LVL,IFADD]<>0) DO
  BEGIN
   LVL:=LVL+1;
   IRAY[LVL,0]:=IRAY[LVL-1,IFADD];
   SEEK(NODE,IRAY[LVL,0]);
   GET(NODE);
   NODEDISKTOARRAY(LVL);
   IFADD:=3
  END;

 NCROSS:=LVL-L;
 LVL:=L
END;          (* END PROCEDURE CROSS *)
```

```
                                                    (* UNIT DASSA *)

(* ****************************************
*                                          *
*            UNIT DASSA MAIN TEXT          *
*                                          *
**************************************** *)

BEGIN  (* BEGIN DASSA *)

END.
```

```
                                                        (* UNIT DASSA *)


SEGMENT PROCEDURE DUMMY;

(* ************************************************************
*                                                              *
*   SEGMENT PROCEDURE DUMMY                                    *
*   USE:  THIS SEGMENT CONTAINS THOSE ROUTINES WHICH ARE       *
*         USED TO GENERATE NEW TREE STRUCTURES.  IN ADDITION,  *
*         THIS SEGMENT IS USED WHEN OPENING NEW DISK FILES     *
*         FOR SUBQUENT ACTIVITY.                               *
*   CALLED BY: DASS                                            *
*   ROUTINES CALLED: SEGMENT PROCEDURE DUMMY (GRAPHICS,        *
*                    LABELATTRIBUTES,READTITLE,SPAN,           *
*                    INITIALIZE, SELECTFILE)                   *
*   VARIABLES:                                                 *
*        USED: CMD                                             *
*        MODIFIED:  (none)                                     *
*                                                              *
************************************************************ *)

   VAR
      CONTROL:INTEGER;
```

```
                                                    (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE GRAPHICS;

(* ************************************************************
*                                                              *
*   PROCEDURE GRAPHICS                                         *
*   USE.  THIS PROCEDURE IS CALLED BY SPAN TO CREATE A         *
*         GRAPHICAL DISPLAY OF THE TREE STRUCTURE AS IT IS     *
*         CREATED. THIS GRAPHICAL PACKAGE DRAWS THE WIRING     *
*         TREE DIAGRAM AS WELL AS HANDLES THE LINKAGES FOR     *
*         INPUTTING NODE LABELS INTO THE TREE STRUCTURE.       *
*         THIS PROCEDURE HAS TWO SUBORDINATE PROCEDURES        *
*         (LABELS and DRAWBLK) WHICH AID IN THE GRAPHICS       *
*         DISPLAY.                                             *
*   CALLED BY: SEGMENT PROCEDURE DUMMY (SPAN)                  *
*   ROUTINES CALLED: SEGMENT PROCEDURE DUMMY (LABELS,DRAWBLK)  *
*   VARIABLES:                                                 *
*        USED: NODELABEL (see UNIT DASSA)                      *
*              CONTROL (see SEGMENT PROCEDURE DUMMY)           *
*        MODIFIED: ANSWER (see UNIT DASSA)                     *
*                  X,Y,ANS                                     *
*                                                              *
************************************************************ *)

   VAR
     X,Y,J:INTEGER;
     ANS:STRING;
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE LABELS(X,Y:INTEGER);

(* ************************************************************
*                                                              *
*   PROCEDURE LABELS                                           *
*   USE:  SUBORDINATE PROCEDURE TO PROCEDURE GRAPHICS, THIS    *
*         PROCEDURE ENABLES CHARACTER INPUT FROM THE USER      *
*         TO BE DISPLAYED IN "REAL TIME" ON THE MONITOR IN     *
*         THE GRAPHICS MODE.  THE ARGUMENTS USED IN THIS       *
*         PROCEDURE ARE THE SCREEN COORDINATES OF THE          *
*         LOWER LEFT CORNER WHERE THE FIRST CHARACTER IS TO    *
*         BE DISPLAYED.  BACKSPACE FUNCTION IS AVAILABLE.      *
*         ENTRY TERMINATES WHEN ANY OF THREE CONDITIONS ARE    *
*         MET:                                                 *
*             1)  THE WORD DONE IS TYPED                       *
*             2)  WHEN THE RETURN KEY IS DEPRESSED             *
*             3)  WHEN MORE THAN 10 CHARACTERS (NOT            *
*                 INCLUDING BACKSPACED CHARACTERS) ARE         *
*                 ENTERED                                      *
*   CALLED BY: SEGMENT PROCEDURE DUMMY (GRAPHICS)              *
*   ROUTINES CALLED: (none)                                    *
*   VARIABLES:                                                 *
*        USED: (none)                                          *
*        MODIFIED: X,Y,X1                                      *
*                  ANSWER,CH (see UNIT DASSA)                  *
*                  J,ANS  (see PROCEDURE GRAPHICS)             *
*                                                              *
************************************************************* *)

  VAR
     X1:INTEGER;


  BEGIN
    X1:=X;
    ANSWER:='';ANS:=' ';
    MOVETO(X,Y);

 REPEAT
    BEGIN


    REPEAT
     UNTIL KEYPRESS;
    READ(CH);
    ANS[1]:=CH;
    J:=ORD(CH);
    IF(NOT EOLN)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

        THEN
          BEGIN
          IF(J<>8)
            THEN
              BEGIN
              IF(CH=' ') THEN CH:=' ';
              WCHAR(CH);
              ANSWER:=CONCAT(ANSWER,ANS);
              X:=X+7
              END
            ELSE
              IF(X1(X)
                THEN
                 BEGIN
                  MOVETO(X-7,Y);
                  WCHAR(' ');
                  MOVETO(X-7,Y);
                  ANSWER:=COPY(ANSWER,1,(LENGTH(ANSWER)-1));
                  X:=X-7
                 END;
          END;


    END

    UNTIL ((ANSWER='DONE')OR(EOLN)OR(LENGTH(ANSWER))=10));
END;
```

```
                                                    (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE DRAWBLK(X,Y:INTEGER);

(*****************************************************************
*                                                                *
*  PROCEDURE DRAWBLK (X,Y:INTEGER)                               *
*  USE: DRAWS RECTANGLES CONTAINING NODE LABELS.  THIS           *
*       PROCEDURE IS SUBORDINATE TO PROCEDURE GRAPHICS.  THE     *
*       TWO ARGUMENTS REPRESENT THE LOWER LEFT CORNER OF THE     *
*       RECTANGLE.  THE RECTANGLE IS DRAWN BY COLORING THE       *
*       ENTIRE RECTANGLE AREA WITH THE DESIRED RECTANGLE         *
*       COLOR OUTLINE AND IS THEN FILLED IN BY THE BACKGROUND    *
*       COLOR.                                                   *
*  CALLED BY: SEGMENT PROCEDURE DUMMY (GRAPHICS)                 *
*  ROUTINES CALLED: (none)                                       *
*  VARIABLES:                                                    *
*       USED: X,Y                                                *
*       MODIFIED: (none)                                         *
*                                                                *
***************************************************************** *)

BEGIN

  VIEWPORT(X,X+75,Y,Y+13);
  FILLSCREEN(GREEN);
  VIEWPORT(X+1,X+73,Y+1,Y+12);
  FILLSCREEN(BLACK);
  VIEWPORT(0,279,0,191)

END;
```

```
                                                    (* SEGMENT PROCEDURE DUMMY *)

(* ************************************************************
*                                                              *
*       START  PROCEDURE  GRAPHICS                             *
*                                                              *
************************************************************** *)

BEGIN

  ANSWER:=' ';

    CASE CONTROL OF

  1:
  BEGIN
  X:=105; Y:=178; DRAWBLK(X,Y);
  ANSWER:=NODELABEL[LVL];
  MOVETO(107,180); WSTRING(ANSWER);
  X:=5; Y:=113; DRAWBLK(X,Y);
  MOVETO(139,178); PENCOLOR(GREEN);
  MOVETO(139,141); MOVETO(39,141); MOVETO(39,126);
  PENCOLOR(NONE);
  LABELS(X+2,Y+2)
  END;

  2:
  BEGIN
  X:=105; Y:=113; DRAWBLK(X,Y);
  MOVETO(139,141);
  PENCOLOR(GREEN); MOVETO(139,126);
  PENCOLOR(NONE);
  LABELS(X+2,Y+2)
  END;

  3:
  BEGIN
  X:=205; Y:=113; DRAWBLK(X,Y);
  MOVETO(139,141); PENCOLOR(GREEN);
  MOVETO(239,141); MOVETO(239,126);
  PENCOLOR(NONE);
  LABELS(X+2,Y+2)
  END;


  4:
  BEGIN
  X:=55; Y:=45; DRAWBLK(X,Y);
  MOVETO(93,141); PENCOLOR(GREEN);
  MOVETO(93,58);  PENCOLOR(NONE);
```

```
                                                (* SEGMENT PROCEDURE DUMMY *)

    LABELS(X+2,Y+2)
    END;


    5:
    BEGIN
    X:=155; Y:=45; DRAWBLK(X,Y);
    MOVETO(191,141); PENCOLOR(GREEN);
    MOVETO(191,58);  PENCOLOR(NONE);
    LABELS(X+2,Y+2)
    END;


    END

  END; (* END GRAPHICS *)
```

```
                                                    (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE LABELATTRIBUTES;

(* ************************************************************
*                                                              *
*  PROCEDURE LABELATTRIBUTES                                   *
*  USE:  USED TO SUPPORT OPTION ATT WHICH ALLOWS THE USER      *
*        TO ENTER THE ATTRIBUTE CHARACTERISTIC.  THE USER      *
*        IS ALLOWED TO ENTER ANY STRING; HOWEVER, ANY          *
*        CHARACTERISTIC ENTERED WITH A R  WILL BE ASSUMED      *
*        BY THE SENSITIVITY ANALYSIS TO BE REGRET AND WILL     *
*        ANNOTATE THOSE ALTERNATIVES WITH THE LOWEST VALUES.   *
*  CALLED BY:  SEGMENT PROCEDURE DUMMY                         *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED:  COMMENTSTRING,MAXLABELSIZE  (see UNIT DASSA)    *
*       MODIFIED:  LATT (see UNIT DASSA)                       *
*                                                              *
************************************************************ *)

  BEGIN
    PAGE(OUTPUT);
    WRITELN(OUTPUT,'ENTER ATTRIBUTE CHARACTERISTIC  ');
    WRITE(OUTPUT,'(REGRET OR VALUE)?   ');
    READLN(INPUT,LATT);
    LATT:=COPY(CONCAT(LATT,LABELSTRING),1,MAXLABELSIZE);
  END; (* END LABEL ATTRIBUTES CHARACTERISTICS *)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE READTITLE;

(* ************************************************************
*                                                              *
*  PROCEDURE READTITLE                                         *
*  USE:  PROCEDURE USED TO ALLOW THE USER TO ENTER THE TITLE   *
*        OF THE TREE STRUCTURE WHICH WILL BE RE-DISPLAYED      *
*        IN VARIOUS REVIEW TYPE OPTIONS. USED TO SUPPORT       *
*        OPTION TTL.                                           *
*  CALLED BY:  SEGMENT PROCEDURE DUMMY                         *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: COMMENTSTRING,MAXCOMMENTSIZE (see UNIT DASSA)    *
*       MODIFIED: TITLE (see UNIT DASSA)                       *
*                                                              *
************************************************************ *)

  BEGIN
    PAGE(OUTPUT);
    WRITELN(OUTPUT,'ENTER A TITLE FOR THIS DATA STRUCTURE  ');
    READLN(INPUT,TITLE);
    TITLE:=COPY(CONCAT(TITLE,COMMENTSTRING),1,MAXCOMMENTSIZE)
  END;  (* END READ TITLE FOR DATA STRUCTURE *)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE SPAN;  (* SPAN *)

(* ************************************************************
*                                                              *
*  PROCEDURE SPAN                                              *
*  USE: USED TO SUPPORT OPTION SPA AND IN THE CREATION OF A    *
*       NEW TREE STRUCTURE.  THIS ROUTINE TAKES ADVANTAGE OF   *
*       THE GRAPHICS CAPABILITIES OF THE APPLE II MICRO-       *
*       COMPUTER.  USING SUBROUTINE GRAPHICS, THE USER ENTERS  *
*       THE ATTRIBUTES (NODES) DIRECTLY ONE THE SCREEN WITHIN  *
*       THE WIRING DIAGRAM.  THE PROCEDURE WILL UPDATE         *
*       THE NODE STRUCTURE AS WELL AS GIVING CURRENT NUMBER    *
*       NODES IN THE SYSTEM.                                   *
*  CALLED BY: SEGMENT PROCEDURE DUMMY                          *
*  ROUTINES CALLED: UNIT DASSA (PRETOT,PRENEX,NEXT,            *
*                               NODEARRAYTODISK,               *
*                               MASTERNODESETUP)               *
*                   SEGMENT PROCEDURE DUMMY (GRAPHICS)         *
*  VARIABLES:                                                  *
*       USED:     LABELSTRING (see UNIT DASSA)                 *
*       MODIFIED: CH,ANSWER,ICONT,NNRN,IFADD,L,LVL,NDESP,      *
*                 NNODES,I,ISTR,IRAY,OLDCELLNUM,NODELABEL,     *
*                   (see UNIT DASSA)                           *
*                 CONTROL (see SEGMENT PROCEDURE DUMMY)        *
*                                                              *
************************************************************** *)

  BEGIN
    PRETOT;

    REPEAT
      BEGIN
       PAGE(OUTPUT);
       WRITE(OUTPUT,'SPAN NODES. . .A(LL  S(ELECT ');
       REPEAT UNTIL KEYPRESS;
       READ(INPUT,CH)
      END
    UNTIL((CH='A')OR(CH='S'));

    IF(CH='S') THEN PRENEX;

    IF(ICONT()1)

      THEN
       BEGIN

          REPEAT
            BEGIN
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

          PAGE(OUTPUT);
          WRITE(OUTPUT,'DO YOU WANT TO BUILD A NEW TREE? (Y/N) ');
          REPEAT UNTIL KEYPRESS;
          READ(INPUT,CH)
         END
       UNTIL((CH='Y')OR(CH='N'));

       IF (CH='N')

       THEN ANSWER:='EXIT'

       ELSE
        BEGIN
        MASTERNODESETUP;
        ICONT:=1
        END
    END;

NNRN:=0;
INITTURTLE;

REPEAT

  BEGIN
  IF (ICONT()0)
  THEN
   BEGIN

    IFADD:=2;  (* 2=PARENT, 3=BROTHER *)
    L:=1;
    CONTROL:=1;

    VIEWPORT(0,279,11,191);
    FILLSCREEN(BLACK);
    VIEWPORT(0,279,0,191);

    REPEAT

      BEGIN

      GRAPHICS;
      CONTROL:=CONTROL+1;

      IF((CONTROL)6)OR(ANSWER='')OR(ANSWER='DONE'))
        THEN
        BEGIN
        LVL:=LVL+L-1;
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

IF(LVL)NDEEP) THEN NDEEP:=LVL;

NEXT;
NNRN:=0
END

ELSE
 BEGIN
  IF(ANSWER()'EXIT')

   THEN
    BEGIN
    ANSWER:=COPY(CONCAT(ANSWER,LABELSTRING),1,MAXLABELSIZE);
    NNRN:=NNRN+1;
    NNODES:=NNODES+1;

    MOVETO(2,0);
    WSTRING('CURRENT NUMBER OF NODES: ');
    I:=NNODES;
    STR(I,ISTR);
    WSTRING(ISTR);

    IRAY[LVL,IFADD]:=NNODES;

    SEEK(NODE,IRAY[LVL,0]);
    NODEARRAYTODISK(LVL);
    PUT(NODE);

    OLDCELLNUM:=IRAY[LVL,0];
    LVL:=LVL+L;
    L:=0;
    IFADD:=3;
    IRAY[LVL,0]:=NNODES;
    IRAY[LVL,1]:=NNRN;
    IRAY[LVL,2]:=0;
    IRAY[LVL,3]:=0;
    IRAY[LVL,4]:=OLDCELLNUM;
    IRAY[LVL,5]:=OLDCELLNUM;
    ARAY[LVL,1]:=0;
    ARAY[LVL,2]:=0;
    NODELABEL[LVL]:=ANSWER;

    SEEK(NODE,IRAY[LVL,0]);
    NODEARRAYTODISK(LVL);
    PUT(NODE);

    END;
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

            END
          END

          UNTIL((CONTROL)6)OR(ANSWER='DONE')OR(ANSWER='')OR(ANSWER='EXIT'))
        END;
      END

    UNTIL((ICONT=0)OR(ANSWER='EXIT'));

PAGE(OUTPUT);
TEXTMODE;
END;    (* END OF SPAN *)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE INITIALIZE;

(* ***************************************************************
*                                                                 *
*  PROCEDURE INITIALIZE                                           *
*  USE:  CREATES NEW FILES ON THE DISK FOR NAMES ENTERED IN       *
*        RESPONSE TO OPTIONS SEL OR NEW.  TWO NEW FILES ARE       *
*        CREATED: ONE FOR THE NODES, THE OTHER FOR THE NAMES      *
*        OF THE ALTERNATIVES.  IN ADDITION, CALLS OTHER           *
*        PROCEDURES IN THIS SEGMENT (LABELATTRIBUTES,             *
*        READTITLE, AND SPAN) TO EXPEDITE CREATION OF THE TREE    *
*  CALLED BY: SEGMENT PROCEDURE DUMMY (SELECTFILE)                *
*  ROUTINES CALLED: UNIT DASSA (MASTERNODESETUP)                  *
*                   SEGMENT PROCEDURE DUMMY (LABELATTRIBUTES,     *
*                       READTITLE,SPAN)                           *
*  VARIABLES:                                                     *
*       USED: FILEOFSYSTEMS,MAXSYSTEMS,FILENAME (see UNIT DASSA)  *
*       MODIFIED: NODE NODETITLE,CELLNUMBER,NRNDIGIT,DOWNLINK,    *
*                      CROSSLINK,BACKLINK,RELWEIGHT,CUMWEIGHT,    *
*                      RATIONALE,SYSTEMVALUES (see UNIT DASSA)    *
*                 SYSTEMS SYSTEMNAME (see UNIT DASSA)             *
*                 I (see UNIT DASSA)                              *
*                                                                 *
*************************************************************** *)

  BEGIN

    WITH SYSTEMS^ DO SYSTEMNAME:='NONE';

    REWRITE(SYSTEMS,FILEOFSYSTEMS);
    FOR I:=1 TO MAXSYSTEMS DO PUT(SYSTEMS);
    CLOSE(SYSTEMS, LOCK);
    RESET(SYSTEMS,FILEOFSYSTEMS);

    WITH NODE^ DO
     BEGIN
     NODETITLE:='BLANK      ';
     CELLNUMBER:=0;
     NRNDIGIT:=0;
     DOWNLINK:=0;
     CROSSLINK:=0;
     BACKLINK:=0;
     RELWEIGHT:=0.0;
     CUMWEIGHT:=0.0;

     FOR I:=1 TO MAXSYSTEMS DO SYSTEMVALUES[I]:=0.0;

     RATIONALE:='NO COMMENT'
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

    END;

  REWRITE(NODE,FILENAME);
  FOR I:=1 TO MAXNUMBEROFNODES DO PUT(NODE);
  CLOSE(NODE, LOCK);
  RESET(NODE,FILENAME);

  LABELATTRIBUTES;
  READTITLE;
  MASTERNODESETUP;
  SPAN

END;  (* END INITIALIZE *)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

PROCEDURE SELECTFILE;

(* ************************************************************
*                                                              *
*  PROCEDURE SELECTFILE                                        *
*  USE:  USED TO ELICIT THE NAMES OF THE DISK AND FILE THAT    *
*        A TREE STRUCTURE EXISTS ON OR WILL BE CREATED ON.     *
*        PROCEDURE USED IN SUPPORT OF OPTIONS SEL AND NEW.     *
*        TWO FILES ARE CREATED AND HAVE THE LABELS OF          *
*                XXXSYS.DATA                                   *
*                XXXNODE.DATA                                  *
*        WHERE XXX REPRESENTS THE FIRST THREE CHARACTERS       *
*        ENTERED BY THE USER WHEN ASKED FOR A FILE NAME.  IN   *
*        THE CASE THAT THE USER ENTERS LESS THAN THREE         *
*        CHARACTERS, THE ENTRY WILL BE LEFT JUSTIFIED IN THE   *
*        X FIELD, WITH THE REMAINING X CHARACTERS.             *
*  CALLED BY: SEGMENT PROCEDURE DUMMY                          *
*  ROUTINES CALLED: UNIT DASSA (NODEDISKTOARRAY)               *
*                   SEGMENT PROCEDURE DUMMY (INITIALIZE)       *
*  VARIABLES:                                                  *
*       USED:  (none)                                          *
*       MODIFIED: NODE NODETITLE,RATIONALE,CELLNUMBER,         *
*                      NRNDIGIT,DOWNLINK,CROSSLINK,BACKLINK    *
*                 DISKNAME,ANSWER,FILENAME,L,I,FLAG,NFLAG,     *
*                 LVL,IRAY (see UNIT DASSA)                    *
*                                                              *
************************************************************** *)
  BEGIN

  PAGE(OUTPUT);
  IF(NFLAG=1)
    THEN BEGIN
         SEEK(NODE,0);
         GET(NODE);
         WITH NODE^ DO
          BEGIN
           RATIONALE:=TITLE;
           NODETITLE:=LATT;
           CELLNUMBER:=NNODES;
           NRNDIGIT:=NDEEP;
           DOWNLINK:=NSYS;
           CROSSLINK:=FLAG
          END;
         SEEK(NODE,0);
         PUT(NODE);
         CLOSE(NODE, LOCK),
         CLOSE(SYSTEMS, LOCK)
         END;
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

NFLAG:=1;

WRITELN(OUTPUT,'ENTER...DISK NAME TO BE USED');
WRITE(OUTPUT,'(E.G. APPLE0:)  ');
READLN(INPUT,DISKNAME);
WRITELN(OUTPUT,'PLEASE INSERT ',DISKNAME,' AND DEPRESS RETURN');
READLN(INPUT,ANSWER);
WRITELN(OUTPUT,'ENTER...FILE NAME TO BE ACCESSED');
READLN(INPUT,FILENAME);

FILENAME:=CONCAT(FILENAME,'XXX');
FILEOFSYSTEMS:=CONCAT(DISKNAME,COPY(FILENAME,1,3),'SYS.DATA');
FILENAME:=CONCAT(DISKNAME,COPY(FILENAME,1,3),'NODE.DATA');

FOR I:=0 TO MAXARRAYSIZE DO FOR L:=1 TO MAXSYSTEMS DO VRAY[I,L]:=0.0;
FOR I:=0 TO MAXARRAYSIZE DO FOR L:=0 TO 5          DO IRAY[I,L]:=0;
FOR I:=0 TO MAXARRAYSIZE DO FOR L:=1 TO 2          DO ARAY[I,L]:=0;

 WRITELN(OUTPUT,'(CR) OR NEW ? ');
 READLN(INPUT,ANSWER);

IF(ANSWER='NEW')
  THEN INITIALIZE

  ELSE
    BEGIN
    RESET(NODE,FILENAME);
    RESET(SYSTEMS,FILEOFSYSTEMS);
    SEEK(NODE,0);
    GET(NODE);
    WITH NODE^ DO
      BEGIN
      TITLE:=RATIONALE;
      LATT:=NODETITLE;
      MNODES:=CELLNUMBER;
      NDEEP:=NRNDIGIT;
      NSYS:=DOWNLINK;
      FLAG:=CROSSLINK;
      END;
    LVL:=1;
    IRAY[1,0]:=1;
    SEEK(NODE,IRAY[LVL,0]);
    GET(NODE);
    NODEDISKTOARRAY(LVL)
    END

END;  (* END SELECTFILE *)
```

```
                                        (* SEGMENT PROCEDURE DUMMY *)

(* ***********************************************
*                                                 *
*    START SEGMENT PROCEDURE DUMMY                *
*                                                 *
*********************************************** *)

  BEGIN
    IF(CMD='NEW') THEN SELECTFILE
   ELSE IF(CMD='ATT') THEN LABELATTRIBUTES
    ELSE IF(CMD='SEL') THEN SELECTFILE
    ELSE IF(CMD='SPA') THEN SPAN
    ELSE IF(CMD='TTL') THEN READTITLE
  END;
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

SEGMENT PROCEDURE WVLOAD;

(* ************************************************************
*                                                              *
*   SEGMENT PROCEDURE WVLOAD                                   *
*   USE:  THIS SEGMENT CONTAINS ROUTINES WHICH ARE USED        *
*         TO INPUT WEIGHTS AND VALUES INTO THE TREE            *
*         STRUCTURE.  IN ADDITION, THIS SEGMENT CONTAINS       *
*         THE PROCEDURE (CALC) WHICH PERFORMS THE              *
*         VALUATIONS OF THE INTERNAL NODES OR COLLAPSING       *
*         THE TREE.                                            *
*  CALLED BY:  PROGRAM DASS                                    *
*  ROUTINES CALLED:  SEGMENT PROCEDURE WVLOAD (CALC,WVLOAD1)   *
*  VARIABLES:                                                  *
*       USED: (none)                                           *
*       MODIFIED: OPT (see SEGMENT PROCEDURE WVLOAD)           *
*                                                              *
************************************************************** *)

(* VARIABLES USED UNIQUELY BY THIS SEGMENT *)
VAR
  OPT,CH1:CHAR;
  LABEL1,ANS:STRING;
  SYS: ARRAY [1..5] OF STRING;
  EXITFLAG,PPOS,J,X1,X,Y,K,L:INTEGER;
  VRAY1:ARRAY [1..5] OF REAL;
  COLOR: ARRAY [1..5] OF SCREENCOLOR;
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE RDV;

(* ************************************************************
*                                                              *
*  PROCEDURE RDV                                               *
*  USE:  THIS PROCEDURE ENABLES THE USER TO INSERT             *
*        ALTERNATIVE VALUES (0-100) FOR EACH DATA NODE IN      *
*        THE TREE STRUCTURE.  OPTIONS INCLUDE INPUTTING ALL    *
*        THE DATA NODES OR SELECTED NODES AT THE USER'S        *
*        DISCRETION.  THIS PROCEDURE USES THREE SUBORDINATE    *
*        PROCEDURES (DRAWBAR and SETUP) WHICH ENABLE           *
*        THIS ROUTINE TO BE INTERACTIVE AS WELL AS             *
*        DISPLAYING OUTPUTS USING THE GRAPHICAL MODE.          *
*  CALLED BY:  SEGMENT PROCEDURE WVLOAD (WVLOAD1)              *
*  ROUTINES CALLED: SEGMENT PROCEDURE WVLOAD (DRAWBAR,         *
*                        SETUP)                                *
*                   UNIT DASSA (NODEARRAYTODISK,NUMTOSTRING,   *
*                               STRTOREAL)                     *
*  VARIABLES:                                                  *
*       USED:      TENSTRING,NSYS,LVL,ISTR (see UNIT DASSA)    *
*                  ANS,COLOR,Y,SYS  (see SEGMENT PROCEDURE     *
*                                    WVLOAD)                   *
*       MODIFIED: CK,ICONT,VRAY,ANSWER (see UNIT DASSA)        *
*                 EXITFLAG,FPOS,J,X,VRAY (see SEGMENT          *
*                                    PROCEDURE WVLOAD)         *
*                                                              *
************************************************************** *)
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE DRAWBAR(A,B,C:INTEGER;V:REAL);

(* ************************************************************
*                                                              *
*   PROCEDURE DRAWBAR                                          *
*   USE: USES COMPUTER GRAPHICS TO DRAW COLOR BARS             *
*        WHOSE HEIGHTS REPRESENTS THE VALUES OF THE            *
*        ALTERNATIVES.  THE COLORS ARE DEPENDENT ON THE        *
*        ALTERNATIVES; ALTERNATIVE 1 = ORANGE                  *
*                      ALTERNATIVE 2 = VIOLET                  *
*                      ALTERNATIVE 3 = BLUE                    *
*                      ALTERNATIVE 4 = GREEN                   *
*                      ALTERNATIVE 5 = WHITE!                  *
*        THE A,B,C VARIABLES CORRESPONDS TO THE LOWER LEFT     *
*        HAND SIDE OF THE TALL RECTANGLE, WHERE C REPRESENTS   *
*        THE ALTERNAIVE NUMBER.  VARIABLE B CORRESPONDS TO     *
*        VALUE OF THE ALTERNATIVE AT THAT NODE.                *
*   CALLED BY:  SEGMENT PROCEDURE WVLOAD (RDV,SETUP)           *
*   ROUTINES CALLED: (none)                                    *
*   VARIABLES:                                                 *
*        USED: A,B,C,V (see PROCEDURE DRAWBAR)                 *
*        MODIFIED: (none)                                      *
*                                                              *
************************************************************** *)


 BEGIN
  VIEWPORT(A+C*15,A+(C*15)+10,B,B+50);
  FILLSCREEN(BLACK);
  VIEWPORT(A+C*15,A+(C*15)+10,B,B+TRUNC(V/2));
  FILLSCREEN(COLOR(C+1));
  VIEWPORT(0,279,0,191)
 END;
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE SQUARES(A,B:INTEGER);

(* ************************************************************
*                                                              *
*  PROCEDURE SQUARES                                           *
*  USE:  USED TO DRAW HOLLOWED OUT BOXES WHERE DATA            *
*        THE CURRENT VALUE OF A PARTICULAR ALTERNATIVE AND     *
*        THE BOX WHERE NEW VALUES OF THAT ALTERNATIVE WILL     *
*        BE ENTERED.  THE TECHNIQUE USED IN CONSTRUCTING THE   *
*        BOX IS TO SET THE VIEWPORT TO THE SIZE OF THE BOX,    *
*        FILL IT WITH THE DESIRED COLOR (GREEN) AND THEN       *
*        HOLLOW OUT THE BOX WITH THE BACKGROUND COLOR          *
*        (BLACK). THE ARGUMENTS A AND B CORRESPONDS TO THE     *
*        LOWER LEFT HAND CORNER WHERE THE BOX IS TO BE         *
*        DRAWN.                                                *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD (SETUP)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED:  A,B (see PROCEDURE SQUARES)                     *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************ *)

 BEGIN

  VIEWPORT(A,A+75,B,B+13);
  FILLSCREEN(GREEN);
  VIEWPORT(A+1,A+73,B+1,B+12);
  FILLSCREEN(BLACK);

  VIEWPORT(0,279,0,191);

 END;
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE SETUP;

(* ************************************************************
*                                                              *
*  PROCEDURE SETUP                                             *
*  USE:  SETS UP AND DISPLAYS A PATICULAR DATA NODE FOR USER   *
*        VALUE INPUT.  PROVIDES CURRENT VALUES OF THE          *
*        ALTERNATIVES, IN BAR FORMAT (USING PROCEDURE          *
*        DRAWBAR).  ALSO PROVIDES DISPLAY FOR THE FIRST        *
*        ALTERNATIVE AND ITS VALUE IN DECIMAL FORM.            *
*  CALLED BY:  SEGMENT PROCEDURE WVLOAD (RDV)                  *
*  ROUTINES CALLED:  UNIT DASSA (INTTOSTRING)                  *
*                    SEGMENT PROCEDURE WVLOAD (SQUARES,        *
*                       DRAWBAR)                               *
*  VARIABLES:                                                  *
*       USED:     NODELABEL,TENSTRING,LVL,IRAY,NSYS            *
*                          (see UNIT DASSA)                    *
*       MODIFIED:  I,ANSWER,ISTR (see UNIT DASSA)              *
*                  VRAY1,PPOS,COLOR (see SEGMENT PROCEDURE     *
*                          WVLOAD)                             *
*                                                              *
************************************************************** *)

BEGIN

  COLOR[1]:=ORANGE; COLOR[2]:=VIOLET;
  COLOR[3]:=BLUE;   COLOR[4]:=GREEN;
  COLOR[5]:=WHITE1;

  FOR I:=1 TO NSYS DO
   BEGIN
    SEEK(SYSTEMS,I-1);
    GET(SYSTEMS);
    WITH SYSTEM^ DO SYS[I]:=SYSTEMNAME
   END;

  FOR I:=1 TO 5 DO VRAY1[I]:=0;

  MOVETO(0,181);
  WSTRING('VALUE: A)BORT B)ACK E)XIT N)EXT (ESC)');

  SQUARES(103,157);


  MOVETO(105,159);
  WSTRING(NODELABEL[LVL]);
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

  MOVETO(0,159); WSTRING('NRN: ');
  ANSWER:='';
  FOR I:=1 TO LVL DO
   BEGIN
    INTTOSTRING(IRAY[I,1]);
    ANSWER:=CONCAT(ANSWER,ISTR,' ')
   END;
  WSTRING(COPY(CONCAT(ANSWER,TENSTRING),1,9));


  MOVETO(35,145); WSTRING('OLD VALUES');
  MOVETO(184,145); WSTRING('NEW VALUES');

  FOR I:=0 TO NBYS-1 DO
   DRAWBAR(34,85,I,VRAY[LVL,I+1]);

  FOR I:=0 TO NBYS-1 DO
   DRAWBAR(183,85,I,VRAY1[I+1]);

   MOVETO(70,45); WSTRING('ALTERNATIVE');

  MOVETO(30,30); WSTRING(CONCAT('OLD ',COPY(CONCAT(LATT,'      '),1,6)));
  SQUARES(113,27);

  MOVETO(30,15); WSTRING(CONCAT('NEW ',COPY(CONCAT(LATT,'      '),1,6)));
  SQUARES(113,12);

   PPOS:=1

END; (* END SETUP *)
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

(* ************************************************
*                                                  *
*            BEGIN PROCEDURE RDV                   *
*                                                  *
************************************************** *)
BEGIN

 INITTURTLE;
 EXITFLAG:=0;
 SETUP;

 REPEAT
  BEGIN
   VIEWPORT(113,188,42,57); FILLSCREEN(COLOR(PPOS));
   VIEWPORT(0,279,0,191); MOVETO(117,45); WSTRING(SYS[PPOS]);

   MOVETO(115,29); NUMTOSTRING(VRAY[LVL,PPOS]);
   WSTRING(ISTR);

   MOVETO(115,14); WSTRING(TENSTRING);


   ANS:=' ';
   REPEAT UNTIL KEYPRESS;
   READ(INPUT,CH);
   ANS[1]:=CH;

   IF((POS(ANS,'ABNE')<>0)OR(EOLN)OR(ORD(CH)=7))

    THEN
     CASE ORD(CH) OF

     27: ICONT:=0;

     13,32: BEGIN
            VRAY[PPOS]:=VRAY[LVL,PPOS];
            DRAWBAR(183,85,PPOS-1,VRAY[PPOS]);
            IF(PPOS<NSYS) THEN PPOS:=PPOS+1
            END;

     65,69: BEGIN
             IF (CH='A') THEN FOR J:=1 TO NSYS DO VRAY[J]:=VRAY[LVL,J];
             FOR J:=1 TO NSYS DO VRAY[LVL,J]:=VRAY[J];
             SEEK(NODE,IRAY[LVL,0]);
             GET(NODE);
             SEEK(NODE,IRAY[LVL,0]);
             NODEARRAYTODISK(LVL);
             PUT(NODE);
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

        EXITFLAG:=1
       END;

66:    IF(PPOS(=1) THEN PPOS:=1
                   ELSE PPOS:=PPOS-1;

78:    IF(PPOS)=NSYS) THEN PPOS:=NSYS
                      ELSE PPOS:=PPOS+1;

END (* END CASE *)


ELSE
 BEGIN
  ANSWER:=' ';
  ANSWER[1]:=CH;
  X:=115; Y:=14; X1:=X;
  MOVETO(X,Y);
  WCHAR(CH);
  X:=X+7;


  REPEAT
   BEGIN
    ANS:=' ';
    REPEAT
     UNTIL KEYPRESS;
    READ(INPUT,CH);
    ANS[1]:=CH;
    J:=ORD(CH);
    IF(NOT EOLN)
     THEN
      BEGIN
       IF(J()8)
        THEN
         BEGIN
          IF(CH=' ') THEN CH:=' ';
          WCHAR(CH);
          ANSWER:=CONCAT(ANSWER,ANS);
          X:=X+7
         END
        ELSE
         IF(X1(X)
          THEN
           BEGIN
            MOVETO(X-7,Y);
            WCHAR(' ');
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

                    MOVETO(X-7,Y);
                    ANSWER:=COPY(ANSWER,1,(LENGTH(ANSWER)-1));
                    X:=X-7
                   END;
              END;


              END

             UNTIL ((EOLN)OR(LENGTH(ANSWER))=10));
             VRAY1[PPOS]:=STRTOREAL(ANSWER);
             IF (VRAY1[PPOS])100) THEN VRAY1[PPOS]:=100;
             IF (VRAY1[PPOS](0)   THEN VRAY1[PPOS]:=0;
             DRAWBAR(183,85,PPOS-1,VRAY1[PPOS]);
             IF(PPOS(NSYS) THEN PPOS:=PPOS+1
             END

          END
       UNTIL((ORD(CH)=27)OR(EXITFLAG=1));
IF(ORD(CH)=27) THEN ITOTL:=0;

PAGE(OUTPUT);
TEXTMODE;
END;
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE RDWT;

(* ************************************************************
*                                                              *
*  PROCEDURE RDWT                                              *
*  USE:  THIS PROCEDURE ENABLES THE USER TO INSERT             *
*        WEIGHTS FOR ALL SPANS ON THE NODE IN                  *
*        THE TREE STRUCTURE.  OPTIONS INCLUDE INPUTTING ALL    *
*        SPANS OR A SINGLE SPAN  UNDERNEATH A SPECIFIED        *
*        NODE.                                                 *
*  CALLED BY:  SEGMENT PROCEDURE WVLOAD (WVLOAD1)              *
*  ROUTINES CALLED: SEGMENT PROCEDURE WVLOAD (DRAWBAR,         *
*                         SETUP)                               *
*                   UNIT DASSA (NODEARRAYTODISK,CROSS,         *
*                               STRTOREAL,ANSWERTOQUESTION)    *
*  VARIABLES:                                                  *
*       USED:     MAXCOMMENTSIZE,NODELABEL(see UNIT DASSA)     *
*       MODIFIED: QUESTION,LVL,ARAY,I,ANSWER,NODE:RATIONALE,   *
*                 ITOTL  (see UNIT DASSA)                      *
*                 LR,INT,J (see PROCEDURE RDWT)                *
*                                                              *
************************************************************** *)

  VAR
   J,INT : INTEGER;
   LR : REAL;

  BEGIN

  CROSS;

  REPEAT
   BEGIN
     PAGE(OUTPUT);
     WRITELN(OUTPUT,'            OLD WTS NEW WTS NORMALIZED');

     FOR I:=1 TO NCROSS DO
      BEGIN
      J:=LVL+I;
      GOTOXY(0,I+1);
      WRITE(OUTPUT,NODELABEL[J]);
      GOTOXY(14,I+1);
      INT:=TRUNC(100*ARAY[J,13]);
      WRITELN(OUTPUT,INT:4)
      END;

     LR:=0.0;
```

```
                                                (* SEGMENT PROCEDURE WVLOAD *)

    FOR I:=1 TO NCROSS DO
     BEGIN
     GOTOXY(22,I+1);
     J:=LVL+I;
     READLN(INPUT,ANSWER);
     ARAY[J,1]:=STRTOREAL(ANSWER);
     IF(ARAY[J,1]<0)
      THEN
      BEGIN
      ITOTL:=0;
      I:=NCROSS+100
      END

      ELSE
      LR:=LR+ARAY[J,1];

     END;

    IF((I<(NCROSS+5))OR(LR<>0))
      THEN
      BEGIN
      FOR I:=1 TO NCROSS DO
       BEGIN
        GOTOXY(32,I+1);
        J:=LVL+I;
        ARAY[J,1]:=ARAY[J,1]/LR;
        INT:=TRUNC(100*ARAY[J,1]);
        WRITE(OUTPUT,INT:4)
       END;

    GOTOXY(0,I+2);
    QUESTION:='ARE THESE WEIGHTS OKAY? (Y/N) ';
    ANSWERTOQUESTION

    END

  END

UNTIL((ANSWER='Y')OR(I>NCROSS+5)OR(LR=0));

IF((I<(NCROSS+5))OR(LR<>0))
 THEN
 BEGIN
 WRITELN(OUTPUT,'ENTER RATIONALE FOR WEIGHTS');
 WRITELN(OUTPUT,'(MAX ',MAXCOMMENTSIZE:3,' LETTERS)');
 READLN(INPUT,ANSWER);
 WITH NODE^ DO RATIONALE:=COPY(CONCAT(ANSWER,COMMENTSTRING),1,
       MAXCOMMENTSIZE);
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

    J:=LVL;
    FOR I:=1 TO NCROSS DO
     BEGIN
     LVL:=LVL+1;
     ARAY[LVL,2]:=ARAY[LVL,1];
     SEEK(NODE,IRAY[LVL,0]);
     NODEARRAYTODISK(LVL);
     PUT(NODE)
     END;

    LVL:=J
    END

END;  (* END PROCEDURE RDWT *)
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE CALC; (* CALCULATES THE TREE *)

(* ************************************************************
*                                                              *
*  PROCEDURE CALC                                              *
*  USE:  THIS PROCEDURE CALCULATES OR 'COLLAPSES' THE TREE     *
*        BASED ON THE WEIGHTS AND VALUES SPECIFIED FOR EACH    *
*        PARTICULAR NODE OR SPAN.  THE PROCEDURE CALCULATES    *
*        THE CUMULATIVE WEIGHTS FROM THE TOP OF THE TREE       *
*        DOWN.  IF THE PROCEDURE REACHES THE DATA NODE VALUES  *
*        ARE ADDED TO THE TOTAL VALUE OF THE TREE.             *
*  CALLED BY:   SEGMENT PROCEDURE WVLOAD                       *
*  ROUTINES CALLED:  UNIT DASSA (PRETOT,NODEARRAYTODISK,NEXT)  *
*  VARIABLES:                                                  *
*       USED:       IRAY,NSYS,ICONT (see UNIT DASSA)           *
*      MODIFIED:    ARAY,NDEEP,FLAG,VRAY,I,L,LVL               *
*                                 (see UNIT DASSA)             *
*                   J (see SEGMENT PROCEDURE WVLOAD)           *
*                   Z,QUITFLAG (see PROCEDURE RDWT)            *
*                                                              *
************************************************************** *)
  VAR
    Z:REAL;
    QUITFLAG:INTEGER;

BEGIN
     FLAG:=0;
     ARAY[0,1]:=1;
     ARAY[0,2]:=1;
     ARAY[1,1]:=1;
     ARAY[1,2]:=1;
     PRETOT;
     J:=0;
     PAGE(OUTPUT); WRITE(OUTPUT,'CALCULATING TREE.');
     NDEEP:=0;

     REPEAT
       BEGIN
        IF(ICONT<>0)
         THEN
         BEGIN
          ARAY[LVL,2]:=ARAY[LVL-1,2]*ARAY[LVL,1];
          J:=J+1;
          IF(LVL>NDEEP) THEN NDEEP:=LVL;
          IF(IRAY[LVL,2]>0)
            THEN
            BEGIN
             FOR I:=1 TO NSYS DO VRAY[LVL,I]:=0;
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

                NEXT
               END

               ELSE
               BEGIN
                FOR K:=1 TO NSYS DO
                 BEGIN
                 Z:=VRAY[LVL,K]*ARAY[LVL,2];
                 FOR L:=1 TO (LVL-1) DO VRAY[L,K]:=VRAY[L,K]+Z
                 END;
                SEEK(NODE,IRAY[LVL,0]);
                NODEARRAYTODISK(LVL);
                PUT(NODE);
                L:=LVL;
                QUITFLAG:=1;
                REPEAT
                  IF(IRAY[LVL,3]>0)
                    THEN
                     BEGIN
                     LVL:=L;
                     NEXT;
                     QUITFLAG:=0
                     END
                    ELSE
                     BEGIN
                     LVL:=LVL-1;
                     IF(ARAY[LVL,2]<>0) THEN FOR I:=1 TO NSYS DO
                          VRAY[LVL,I]:=VRAY[LVL,I]/ARAY[LVL,2];
                     SEEK(NODE,IRAY[LVL,0]);
                     NODEARRAYTODISK(LVL);
                     PUT(NODE);
                     IF(LVL<=1)
                       THEN
                       BEGIN
                        LVL:=L;
                        NEXT;
                        QUITFLAG:=0
                       END
                     END
                UNTIL(QUITFLAG=0)
              END
            END;
            WRITE(OUTPUT,'.')
          END
        UNTIL(ICONT=0)
END;    (* END PROCEDURE CALC *)
```

```
                                                    (* SEGMENT PROCEDURE WVLOAD *)

PROCEDURE WVLOAD1;

(* ************************************************************
*                                                              *
*  PROCEDURE WVLOAD1                                           *
*  USE:  THIS PROCEDURE IS EXECUTED IF LOADING THE WEIGHTS     *
*        OR VALUES FOR SPANS OR NODES ARE REQUIRED.            *
*        IF ALL NODES ARE TO BE ASSIGNED WEIGHTS OR VALUES     *
*        THE PROCEDURE STARTS FROM THE TOP OF THE TREE AND     *
*        USING PROCEDURE NEXT (UNIT DASSA) FINDS ALL THE       *
*        DATA NODES (FOR VALUES) OR NON-DATA NODES (FOR        *
*        WEIGHTS).                                             *
*        IF A SELECTED NODE IS DESIRED, PROCEDURE PRENEX       *
*        (UNIT DASSA) IS EXERCISED.                            *
*  CALLED BY: SEGMENT PROCEDURE WVLOAD                         *
*  ROUTINES CALLED: UNIT DASSA (PRETOT,PRENEX,NEXT)            *
*                   SEGMENT PROCEDURE WVLOAD (RDV,RDWT)        *
*  VARIABLES:                                                  *
*       USED:      ICONT,IRAY,ITOTL (see UNIT DASSA)           *
*                  OPT (see SEGMENT PROCEDURE WVLOAD)          *
*       MODIFIED: CH (see UNIT DASSA)                         *
*                  CH1,LABEL1 (see SEGMENT PROCEDURE WVLOAD)   *
*                                                              *
************************************************************ *)

 BEGIN
  REPEAT
   BEGIN
    IF(OPT='W') THEN LABEL1:='WEIGHTS'
                ELSE LABEL1:='VALUES';
    PAGE(OUTPUT);
    WRITE(OUTPUT,LABEL1,': A(LL S(ELECT ');
    REPEAT
     UNTIL KEYPRESS;
    READ(INPUT,CH1)
   END
  UNTIL((CH1='S')OR(CH1='A'));

  WRITELN(OUTPUT);
  REPEAT
   BEGIN
    PRETOT;
    IF(CH1='S') THEN PRENEX;
    IF(ICONT=0)
     THEN
      BEGIN
       WRITELN(OUTPUT,'NRN ENTERED IS INVALID');
       WRITELN(OUTPUT,'(ANY KEY) CONTINUE');
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

      REPEAT UNTIL KEYPRESS;
      READ(INPUT,CH)
     END
  END
 UNTIL(ICONT()0);

 REPEAT
  BEGIN
  IF((IRAY[LVL,2]=0)AND(OPT='V')) THEN RDV;
  IF((IRAY[LVL,2]>0)AND(OPT='W')) THEN RDWT;
  IF(ITOTL=1) THEN NEXT
  END
 UNTIL('ICONT=0)OR(ITOTL()1))

END; (* END WVLOAD1 *)
```

```
                                        (* SEGMENT PROCEDURE WVLOAD *)

(* ***************************************** ****
*                                              *
*        BEGIN SEGMENT PROCEDURE WVLOAD        *
*                                              *
********************************************** *)
BEGIN

REPEAT
 BEGIN

 REPEAT
  BEGIN
   PAGE(OUTPUT);
   WRITE(OUTPUT,'WVC: W(E!GHT V(ALUES C(ALCULATE E(XIT ');
   REPEAT
    UNTIL KEYPRESS;
   READ(INPUT,OPT);
  END
 UNTIL((OPT='W')OR(OPT='V')OR(OPT='C')OR(OPT='E'));

 IF(OPT='C') THEN CALC
  ELSE IF(OPT()'E') THEN WVLOAD1

 END
UNTIL(OPT='E')

END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

SEGMENT PROCEDURE NUM;

(* ************************************************************
*                                                              *
*  SEGMENT PROCEDURE NUM                                       *
*  USE:  THIS SEGMENT HOUSES THE PROCEDURES WHICH ARE          *
*        CALLED ON FOR OPTIONS DIS, NUM, AND REV.              *
*        PROCEDURES FOR OPTION DIS PROVIDES FOR BOTH           *
*        TABULAR AND GRAPHICAL OUTPUT.  IN ADDITION,           *
*        THE TABULAR OUTPUT CAN BE DIRECTED TO EITHER THE      *
*        CONSOLE OR THE LINE PRINTER.  PROCEDURES FOR          *
*        OPTIONS NUM AND REV ARE COMBINED TOGETHER BECAUSE     *
*        OF THEIR SIMILAR FORMATS (OPTION REV IS JUST A        *
*        SHORTENED VERSION OF OPTION NUM) AND CAN BE           *
*        DIRECTED TO EITHER THE CONSOLE OR THE PRINTER.        *
*  CALLED BY: PROGRAM DASS                                     *
*  ROUTINES CALLED: SEGMENT PROCEDURE NUM (OUTDEVICE,          *
*                          NUMERICREVIEW,DISPLAY)              *
*                   UNIT DASSA (NODIN,PRETOT,PREMEX)           *
*  VARIABLES:                                                  *
*       USED:  CMD,IRAY,ICONT,FLAG (see UNIT DASSA)            *
*       MODIFIED: CH,ANSWER (see UNIT DASSA)                   *
*                 CH1 (see SEGMENT PROCEDURE NUM)              *
*                                                              *
************************************************************ *)

   VAR
     CH1:CHAR;
     OUT:STRING;
     J:INTEGER;
     F:FILE OF CHAR;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE OUTDEVICE;

(* ************************************************************
*                                                              *
*  PROCEDURE OUTDEVICE                                         *
*  USE:  THIS PROCEDURE IS CALLED ON BY OTHER ?ROCEDURES IN    *
*        THIS SEGMENT IN ORDER TO DETERMINE WHICH DEVICE THE   *
*        TABULAR OUTPUT IS TO GO ON (CONSOLE OR PRINTER)       *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY0)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: F (see SEGMENT PROCEDURE NUM)                    *
*       MODIFIED: CH (see UNIT DASSA)                          *
*                                                              *
************************************************************ *)

BEGIN
    REPEAT
     BEGIN
      PAGE(OUTPUT);
      WRITE(OUTPUT,'C(ONSOLE P(RINTER ');
      REPEAT
       UNTIL KEYPRESS;
      READ(INPUT,CH)
     END
    UNTIL((CH='C')OR(CH='P'));

    IF(CH='P')
      THEN REWRITE(F,'PRINTER:')
      ELSE REWRITE(F,'CONSOLE:');
END;   (* END OUTDEVICE *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE NUMERICREVIEW;

(* ************************************************************************
*                                                                         *
*  PROCEDURE NUMERICREVIEW                                                *
*  USE:  MAIN PROCEDURE THAT SUPPORT OPTIONS REV AND NUM.                 *
*        PROVIDES OUTPUT DATA, NRN AND NODE LABELS IN THE CASE OF         *
*        OPTION REV AND NRN, NODE LABELS, CUMULATIVE WEIGHTS,             *
*        RELATIVE WEIGHTS AND ALL ALTERNATIVE VALUES IN THE CASE          *
*        OF OPTION NUM.  REVIEWS TO THE PRINTER ASSUME AN 80              *
*        COLUMN FORMAT WHEREAS REVIEWS TO THE CONSOLE ASSUMS AN           *
*        80 COLUMN FORMAT WHERE ONLY THE RIGHT OR LEFT HALF IS            *
*        DISPLAYED AT ANY GIVEN TIME.                                     *
*        THIS PROCEDURE USES ONE SUBORDINATE PROCEDURE WHICH IS           *
*        USED TO PAGE THE OUTPUT TO A NEW PAGE (OR CLEAR THE              *
*        SCREEN IN THE CASE OF THE CONSOLE) AND PRINT NEW HEADING         *
*        INFORMATION.                                                     *
*  CALLED BY:  SEGMENT PROCEDURE NUM                                      *
*  ROUTINES CALLED: UNIT DASSA (INTTOSTRING,NEXT)                         *
*                   SEGMENT PROCEDURE NUM (NEWPG)                         *
*  VARIABLES:                                                             *
*       USED: NSYS,CMD,LVL,IRAY,ISTR,NODELABEL,COMMENTSTRING              *
*                  ARAY,ICONT,VRAY,TITLE (see UNIT DASSA)                 *
*       MODIFIED: ANSWER,I    (see UNIT DASSA)                            *
*                 F,CH1,J (see SEGMENT PROCEDURE NUM)                     *
*                                                                         *
************************************************************************ *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE NEWPG;

(* ***************************************************************
*                                                                 *
*  PROCEDURE NEWPG                                                *
*  USE:  CONSTRUCTS THE HEADER FOR A NEW PAGE OF OUTPUT WHETHER   *
*        THE OUTPUT OCCURS ON THE CONSOLE OR THE LINE PRINTER.    *
*        HEADER CONSISTS OF LABELS AND A LIST OF THE              *
*        ALTERNATIVES (SYSTEMS).                                  *
*  CALLED BY: SEGMENT PROCEDURE NUM (NUMERICREVIEW)               *
*  ROUTINES CALLED (none)                                         *
*  VARIABLES:                                                     *
*       USED: TITLE,CH,NSYS,SYSTEM:SYSTEMNAME (see UNIT DASSA)    *
*       MODIFIED: ANSWER,I (see UNIT DASSA)                       *
*                 F (see SEGMENT PROCEDURE NUM)                   *
*                                                                 *
*************************************************************** *)

  BEGIN
    PAGE(F);
    IF(CH='P')
      THEN WRITELN(F,'                              R E V I E W');
    WRITELN(F,TITLE);
    WRITE(F,'NODE REF NUMBER   LABEL    REL WT CUM WT ');
    IF(CH='P')
       THEN WRITE(F,'           ')
       ELSE WRITE(F,' LABEL     ');
    SEEK(SYSTEMS,0);
    ANSWER:='';
    FOR I:=1 TO NSYS DO
     BEGIN
      GET(SYSTEMS);
      WITH SYSTEMS^ DO ANSWER:=CONCAT(ANSWER,COPY(SYSTEMNAME,1,5),' ')
     END;
    WRITELN(F,ANSWER);
    WRITELN(F,' ')
    END;    (* END NEWPG *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

(* ****************************************************
*                                                      *
*     BEGIN  PROCEDURE  NUMERICREVIEW                  *
*                                                      *
**************************************************** *)

BEGIN
    CH1:='1';

    REPEAT

    J:=2;
    IF(CMD='REV')
     THEN
      BEGIN
       PAGE(F);
       WRITELN(F,TITLE)
      END
     ELSE
      BEGIN
       NEWPG;
       J:=J+1
      END;


    WHILE (((ICONT=1)AND(((J(23)AND(CH='C'))OR((J(57)AND(CH='F')))) DO
      BEGIN
       ANSWER:='';
       FOR I:=1 TO LVL DO
        BEGIN
         INTTOSTRING(IRAY[I,1]);
         ANSWER:=CONCAT(ANSWER,ISTR,' ')
        END;
       ANSWER:=COPY(CONCAT(ANSWER,NODELABEL[LVL],COMMENTSTRING),1,26);
       WRITE(F,ANSWER);

       IF (CMD='NUM')
        THEN
         BEGIN
          WRITE(F,ARAY[LVL,1]:7:2,ARAY[LVL,2]:7:2);
          IF(CH='P')
           THEN WRITE(F,'           ')
           ELSE WRITE(F,NODELABEL[LVL]);

           FOR I:=1 TO (NSYS-1) DO
            WRITE(F,VRAY[LVL,I]:6:2);

           WRITELN(F,VRAY[LVL,NSYS]:6:2);
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

          END
         ELSE
          WRITELN(F);

       J:=J+1;
       NEXT

     END;

     IF(((J)=23)AND(CH='C'))OR((J)=57)AND(CH='P')))
       THEN
        BEGIN
         WRITE(OUTPUT,'(ANY KEY) CONTINUE  (ESC) EXIT');
         REPEAT
          UNTIL KEYPRESS;
         READ(INPUT,CH1);
         PAGE(OUTPUT)
        END;

       UNTIL((ORD(CH1)=27)OR(ICONT=0));

       IF(ORD(CH1)=27) THEN CH:=CH1;


END;  (* END NUMERICREVIEW *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE DISPLAY;

(* ***************************************************************
*                                                                 *
*  PROCEDURE DISPLAY                                              *
*  USE:  PROCEDURE SUPPORT OPTION DIS.  PROVIDES TABULAR OR       *
*        GRAPHICAL DISPLAY OF A NODE AND ITS IMMEDIATE DESCEND-   *
*        ENTS.  INCLUDED IN THE DISPLAY ARE RELATIVE WEIGHTS,     *
*        CUMULATIVE WEIGHTS. AND VALUES OF THE ALTERNATIVES FOR   *
*        THE PARENT AND CHILDREN.                                 *
*        USES TWO IMMEDIATE SUBORDINATE PROCEDURES (DISPLAY0 AND  *
*        DISPLAY2) TO PERFORM THE TABULAR AND GRAPHICAL DISPLAY.  *
*  CALLED BY:  SEGMENT PROCEDURE NUM                              *
*  ROUTINES CALLED: UNIT DASSA (CROSS)                            *
*                   SEGMENT PROCEDURE NUM (DISPLAY0,DISPLAY2)     *
*  VARIABLES:                                                     *
*       USED:  ICONT (see UNIT DASSA)                             *
*       MODIFIED: CH  (see UNIT DASSA)                            *
*                                                                 *
*************************************************************** *)

VAR
  I,J,K,L: INTEGER;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE DISPLAY1; (* DISPLAY *)

(* ***************************************************************
*                                                                 *
*  PROCEDURE DISPLAY1                                             *
*  USE:  DISPLAYS, IN TABULAR FORMAT, THE INPUTTED NODE AND ALL   *
*        IMMEDIATE CHILDREN OF THE TREE STRUCTURE.  DATA PROVIDED *
*        ARE THE LABELS OF THE NODES, EACH NODE'S CUMULATIVE      *
*        WEIGHT, RELATIVE WEIGHT, AND VALUES OF EACH ALTERNATIVE. *
*        A TOTAL IS PRINTED AT THE BOTTOM OF EACH COLUMN          *
*        REPRESENTING THE PARENT NODE INPUTTED BY THE USER.       *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY0)                    *
*  ROUTINES CALLED: UNIT DASSA (INTTOSTRING,NUMTOSTRING)          *
*  VARIABLES:                                                     *
*       USED: ISTR,NSYS,LVL,IRAY,NODELABEL,SYSTEM:SYSTEMNAME,     *
*             NCROSS,ARAY,VRAY,LABELSTRING (see UNIT DASSA)       *
*       MODIFIED: ANSWER (see UNIT DASSA)                         *
*             F (see SEGMENT PROCEDURE NUM)                       *
*             I,J,K,L (see PROCEDURE DISPLAY1)                    *
*                                                                 *
*************************************************************** *)

  VAR
   I,J,K,L : INTEGER;

BEGIN
      PAGE(F);
      ANSWER:='';
      FOR I:=1 TO LVL DO
       BEGIN
       INTTOSTRING(IRAY[I,1]);
       ANSWER:=CONCAT(ANSWER,ISTR,' ')
       END;
      WRITELN(F,ANSWER);
      WRITELN(F,NODELABEL[LVL],'-');
      ANSWER:='  FACTOR  ';
      FOR I:=1 TO NSYS DO
        BEGIN
        SEEK(SYSTEMS,I-1);
        GET(SYSTEMS);
        WITH SYSTEMS^ DO ANSWER:=CONCAT(ANSWER,' ',COPY(SYSTEMNAME,1,5))
        END;
      WRITELN(F,ANSWER);
      FOR I:=1 TO NCROSS DO
        BEGIN
        J:=LVL+I;
        ANSWER:=COPY(CONCAT(NODELABEL[J],LABELSTRING),1,8);
        IF(IRAY[J,2]=0)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

        THEN ANSWER:=CONCAT(ANSWER,'* ')
        ELSE ANSWER:=CONCAT(ANSWER,'  ');
      FOR K:=1 TO NSYS DO
        BEGIN
        NUMTOSTRING(VRAY[J,K]);
        ANSWER:=CONCAT(ANSWER,ISTR)
        END;
      WRITELN(F,ANSWER)
      END;

    ANSWER:='   TOTAL  ';
    FOR I:=1 TO NSYS DO
      BEGIN
      NUMTOSTRING(VRAY[LVL,I]);
      ANSWER:=CONCAT(ANSWER,ISTR)
      END;
    WRITELN(F,ANSWER);
    WRITELN(F,' ');
    WRITELN(F,'  FACTOR    (WT)    CUMWT');
    FOR I:=1 TO NCROSS DO
      BEGIN
      J:=LVL+I;
      ANSWER:=COPY(CONCAT(NODELABEL[J],LABELSTRING),1,8);
      IF(IRAY[J,2]=0)
        THEN ANSWER:=CONCAT(ANSWER,'* ')
        ELSE ANSWER:=CONCAT(ANSWER,'  ');
      NUMTOSTRING(ARAY[J,1]);
      ANSWER:=CONCAT(ANSWER,'(',COPY(ISTR,4,3),')');
      NUMTOSTRING(ARAY[J,2]);
      WRITELN(F,ANSWER,ISTR)
      END;
    NUMTOSTRING(ARAY[LVL,2]);
    WRITELN(F,'  TOTAL         ',ISTR)

END;  (* END DISPLAY1 *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE DISPLAY2;

(* ************************************************************
*                                                              *
*  PROCEDURE DISPLAY2                                          *
*  USE:  DRAWS THE GRAPHICAL REPRESENTATION OF OPTION DIS BY   *
*        DRAWING A WIRING DIAGRAM WITH THE PARENT NODE         *
*        INPUTTED BY THE USER AT THE TOP AND ITS CHILDREN      *
*        BELOW.  BELOW EACH CHILD (TO THE RIGHT OF THE         *
*        PARENT) IS A BAR-GRAPH REPRESENTATION OF THE VALUE    *
*        OF EACH ALTERNATIVE (INPUTTED OR COMPUTED).           *
*        JUST ABOVE EACH CHILD (BELOW FOR THE PARENT) THE      *
*        CUMULATIVE WEIGHT AND RELATIVE WEIGHT (THE RELATIVE   *
*        WEIGHT IS IN PARENTHESES) ARE OUTPUTTED.              *
*        THE UPPER LEFT HAND CORNER OF THE GRAPHICS OUTPUT     *
*        IS A LEGEND INDICATING WHICH COLOR (ORANGE,VIOLET,    *
*        BLUE, GREEN, AND WHITE1) REPRESENTS WHICH ALTERNATIVE *
*        PROCEDURE HAS THREE SUBORDINATE PROCEDURES (WTS,      *
*        DRAWBLK, AND SYSBLK) WHICH AID IN DRAWING THE WIRING  *
*        DIAGRAMS.                                             *
*                                                              *
*        HAS SPECIAL FLAG, FLAG2 WHICH DETERMINES THE STYLE    *
*        OF VARIOUS GRAPHICAL OUTPUTS                          *
*                     FLAG2 = 1  DISPLAY CHILDREN DATA STYLE   *
*                     FLAG2 = 2  DISPLAY PARENT DATA STYLE     *
*                     FLAG2 = 3  DISPLAY LEGEND DATA STYLE     *
*                                                              *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY)                  *
*  ROUTINES CALLED:  SEGMENT PROCEDURES NUM (WTS,DRAWBLK,      *
*                           SYSBLK)                            *
*  VARIABLES:                                                  *
*       USED:  NODELABEL,TENSTRING,NCROSS,IRAY,                *
*                 LVL (see UNIT DASSA)                         *
*       MODIFIED: ANSWER,CH (see UNIT DASSA)                   *
*                 J,X,Y,FLAG2,CONTROL (see PROCEDURE DISPLAY2) *
*                                                              *
************************************************************ *)

VAR
  X,Y:INTEGER;
  FLAG2,J,CONTROL:INTEGER;
```

```
                                        (* SEGMENT PROCEDURE NUM *)

PROCEDURE WTS;

(* ************************************************************
*                                                             *
*  PROCEDURE W                                                *
*  USE:  WRITES OUT THE CUMULATIVE WEIGHT AND RELATIVE WEIGHT *
*        (THE RELATIVE WEIGHT IN PARENTHESES) FOR ANY GIVEN   *
*        NODE. REQUIRES AN X AND Y VALUE BE SET WHICH         *
*        REPRESENTS THE LOWER LEFT SCREEN COORDINATE WHERE THE*
*        WRITING IS TO TAKE PLACE.                            *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY2)                *
*  ROUTINES CALLED: UNIT DASSA (NUMTOSTRING)                  *
*  VARIABLES:                                                 *
*       USED: ARAY (see UNIT DASSA)                           *
*             X,Y (see PROCEDURE DISPLAY2)                    *
*      MODIFIED: ISTR (see UNIT DASSA)                        *
*                                                             *
************************************************************* *)

BEGIN
     NUMTOSTRING(ARAY[J,1]);
     ISTR:=COPY(ISTR,3,4);
     MOVETO(X-5,Y+14);
     ISTR:=CONCAT('(',ISTR,')');
     WSTRING(ISTR);
     NUMTOSTRING(ARAY[J,2]);
     ISTR:=COPY(ISTR,3,4);
     MOVETO(X+46,Y+14);
     WSTRING(ISTR)

END;  (* END WTS *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE DRAWBLK(X,Y:INTEGER);

(* ************************************************************
*                                                              *
*  PROCEDURE DRAWBLK                                           *
*  USE: SUBORDINATE PROCEDURE TO DISPLAY2.  DRAWS THE BLOCKS IN *
*       THE WIRING DIAGRAM WHERE THE LABELS OF THE NODES ARE   *
*       PLACED.  SIMILAR IN FUNCTION TO PROCEDURE DRAWBLK IN   *
*       SEGMENT PROCEDURE DUMMY.                               *
*       USES TWO ARGUMENTS WHICH ARE THE SCREEN COORDINATES OF *
*       THE LOWER LEFT HAND CORNER OF THE BLOCKS               *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY2)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: X,Y (see PROCEDURE DRAWBLK)                      *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************ *)

BEGIN

  VIEWPORT(X,X+75,Y,Y+13);
  FILLSCREEN(GREEN);
  VIEWPORT(X+1,X+73,Y+1,Y+12);
  FILLSCREEN(BLACK);
  VIEWPORT(0,279,0,191)

END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE SYSBLK(X,Y:INTEGER);

(* ************************************************************
*                                                              *
*  PROCEDURE SYSBLK                                            *
*  USE:  DRAWS, IN BAR GRAPH FORMAT, THE VALUES OF THE         *
*        ALTERNATIVES FOR A SPECIFIED CHILD NODE OF THE        *
*        INPUTTED NODE (SPECIFIED BY VARIABLE J).              *
*        THE INPUT ARGUMENTS ARE THE LOCATION THE LOWER LEFT   *
*        HAND CORNER WHERE THE FIRST BAR IS TO BE DRAWN. THE   *
*        PROCEDURE THEN DRAWS THE REMAINING BARS (UP TO FOUR)  *
*        RELATIVE TO THAT COORDINATE.                          *
*        IN ADDITION, THE PROCEDURE WRITES THE LEGEND WHICH    *
*        CONSISTS OF COLOR BARS AND THE FIRST FOUR LETTERS OF  *
*        EACH ALTERNATIVE IN THE UPPER RIGHT CORNER OF THE     *
*        MONITOR SCREEN.                                       *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY2)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: SYSTEMS:SYSTEMNAME,NSYS,VRAY,LVL                 *
*                     (see UNIT DASSA)                         *
*             X,Y,FLAG2  (see PROCEDURE DISPLAY2)              *
*       MODIFIED: ANSWER (see UNIT DASSA)                      *
*                 J (see PROCEDURE DISPLAY2)                   *
*                 CONTROL1,Y1 (see PROCEDURE SYSBLK)           *
*                                                              *
************************************************************* *)

  VAR
   CONTROL1,Y1:INTEGER;


  BEGIN
     FOR CONTROL1:=1 TO NSYS DO
      BEGIN
        CASE FLAG2  OF
        1: Y1:=TRUNC(VRAY[J,CONTROL1]*0.45);
        2: Y1:=TRUNC(VRAY[LVL,CONTROL1]*0.45);
        3: Y1:=45
        END;  (* CASE OF FLAG2  *)


        CASE CONTROL1 OF
        1: BEGIN  VIEWPORT(X,X+10,Y-45,Y-45+Y1); FILLSCREEN(ORANGE) END;
        2: BEGIN  VIEWPORT(X+15,X+25,Y-45,Y-45+Y1); FILLSCREEN(VIOLET) END;
        3: BEGIN  VIEWPORT(X+30,X+40,Y-45,Y-45+Y1); FILLSCREEN(BLUE) END;
        4: BEGIN  VIEWPORT(X+45,X+55,Y-45,Y-45+Y1); FILLSCREEN(GREEN) END;
        5: BEGIN  VIEWPORT(X+60,X+70,Y-45,Y-45+Y1); FILLSCREEN(WHITE1) END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *);

        END; (* END CASE *)

        VIEWPORT(0,279,0,191);

        IF (FLAG2=3)
          THEN
           BEGIN
            SEEK(SYSTEMS,CONTROL1-1);
            GET(SYSTEMS);
            WITH SYSTEMS^ DO ISTR:=SYSTEMNAME;
            FOR J:=1 TO 4 DO
             BEGIN
              ANSWER:=COPY(ISTR,J,1);
              MOVETO(7+(15*(CONTROL1-1)),191-(10*J));
              VSTRING(ANSWER)
             END
           END
      END

END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

(* ************************************************************
*                                                              *
*          BEGIN   PROCEDURE   DISPLAY2                        *
*                                                              *
************************************************************* *)

BEGIN
  INITTURTLE;
  FLAG2:=1;
  MOVETO(83,153); WSTRING('(REL WT)');
  MOVETO(142,153); WSTRING('CUM WT');


  X:=105; Y:=178; DRAWBLK(X,Y);
  ANSWER:=COPY(CONCAT(NODELABEL[LVL],TENSTRING),1,10);
  MOVETO(107,180); WSTRING(ANSWER);


  FOR CONTROL:=1 TO NCROSS DO

  BEGIN
    J:=LVL+CONTROL;
    ANSWER:=COPY(CONCAT(NODELABEL[J],TENSTRING),1,8);
    IF(IRAY[J,2]=0)
      THEN ANSWER:=CONCAT(ANSWER,'*');

    CASE CONTROL OF

  1:
  BEGIN
  X:=5; Y:=113; DRAWBLK(X,Y);
  MOVETO(7,115);  WSTRING(ANSWER);
  WTS;
  MOVETO(139,178); PENCOLOR(GREEN);
  MOVETO(139,141); MOVETO(39,141); MOVETO(39,126);
  PENCOLOR(NONE);
  SYSBLK(X+1,Y);
  END;

  2:
  BEGIN
  X:=105; DRAWBLK(X,Y);
  MOVETO(107,115); WSTRING(ANSWER);
  WTS; MOVETO(139,141);
  PENCOLOR(GREEN); MOVETO(139,126);
  PENCOLOR(NONE);
  SYSBLK(X,Y);
  END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

    3:
    BEGIN
    X:=205; DRAWBLK(X,Y);
    MOVETO(207,115); WSTRING(ANSWER);
    WTS; MOVETO(139,141); PENCOLOR(GREEN);
    MOVETO(239,141); MOVETO(239,126);
    PENCOLOR(NONE);
    SYSBLK(X,Y);
    END;

    4:
    BEGIN
    X:=55; Y:=45; DRAWBLK(X,Y);
    MOVETO(57,47);  WSTRING(ANSWER);
    WTS; MOVETO(93,141); PENCOLOR(GREEN);
    MOVETO(93,58);  PENCOLOR(NONE);
    SYSBLK(X,Y);
    END;

    5:
    BEGIN
    X:=155; DRAWBLK(X,Y);
    MOVETO(157,47); WSTRING(ANSWER);
    WTS; MOVETO(191,141); PENCOLOR(GREEN);
    MOVETO(191,58);  PENCOLOR(NONE);
    SYSBLK(X,Y);
    END;

    END; (* END CASE *)
    END; (* END OF FOR LOOP *)

    MOVETO(181,185); PENCOLOR(GREEN);
    MOVETO(200,185); PENCOLOR(NONE);

    FLAG2:=2; SYSBLK(202,191);  (* FOR NODE VALUES *)
    FLAG2:=3; SYSBLK(6,191);  (* FOR SYSNAMES *)
    J:=LVL;X:=97;Y:=150;WTS;
    REPEAT
     UNTIL KEYPRESS;
    READ(CH);

    TEXTMODE
  END; (*  END DISPLAY2  *)
```

ADA-101140

AD-A101140

```
                                                    (* SEGMENT PROCEDURE NUM *)

PROCEDURE DISPLAY0;

(* ************************************************************
*                                                              *
*  PROCEDURE DISPLAY0                                          *
*  USE: DETERMINES METHOD OF OUTPUTTING THE TABULAR DISPLAY    *
*       (CONSOLE AND PRINTER), DETERMINES THE CHILDREN OF      *
*       THE INPUTTED NODE, AND PROCEEDS TO THE DISPLAY1        *
*       ROUTINE.                                               *
*  CALLED BY: SEGMENT PROCEDURE NUM (DISPLAY)                  *
*  ROUTINES CALLED: UNIT DASSA (CROSS)                         *
*                   SEGMENT PROCEDURE NUM (OUTDEVICE,          *
*                        DISPLAY1)                             *
*  VARIABLES:                                                  *
*       USED: (none)                                           *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************** *)

   BEGIN
     OUTDEVICE;
     CROSS;
     DISPLAY1
   END;
         (* END DISPLAY0 *)
```

```
                                                (* SEGMENT PROCEDURE NUM *)

(* ************************************************** ***************
*                                                                     *
*             BEGIN  PROCEDURE  DISPLAY                               *
*                                                                     *
********************************************************************* *)

BEGIN

  IF(ICONT<>0)
    THEN
    BEGIN
      REPEAT
       BEGIN
         PAGE(OUTPUT);
         WRITE(OUTPUT,'TABULAR GRAPHIC ');
         REPEAT
           UNTIL KEYPRESS;
         READ(CH)
       END
      UNTIL((CH='T')OR(CH='G'));

      IF(CH='T')
       THEN DISPLAY0

       ELSE
        BEGIN
         CROSS;
         DISPLAY2;
         CH:='G'
        END

   END; (* END IF *)
 END;  (* DISPLAY *)
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

(* ************************************************************
*                                                              *
*          BEGIN  SEGMENT PROCEDURE  NUM                       *
*                                                              *
************************************************************** *)

BEGIN
  PRETOT;
  IF(CMD='DIS')
    THEN
      BEGIN
       REPEAT
        BEGIN
         PAGE(OUTPUT);
         WRITELN(OUTPUT,'ENTER NODE TO BE DISPLAYED...');
         NODIN;
         IF(ICONT()0)
          THEN
           IF (IRAY[LVL,2](1)
            THEN BEGIN
                 WRITELN(OUTPUT,'NODE IS A DATA NODE AND CANNOT BE DISPLAYED');
                 WRITELN(OUTPUT,'DEPRESS RETURN TO CONTINUE');
                 READLN(INPUT,ANSWER);
                 END
            ELSE ICONT:=ICONT
          ELSE ICONT:=ICONT
        END
       UNTIL((ICONT=0)OR(IRAY[LVL,2])=1))
      END

    ELSE
      BEGIN

       PRETOT;
       REPEAT
        BEGIN
         PAGE(OUTPUT);
         WRITE(OUTPUT,'A(LI  S(ELECT');
         REPEAT
          UNTIL KEYPRESS;
         READ(INPUT,CH)
        END
       UNTIL((CH='A')OR(CH='S')),
      . IF(CH='S')
         THEN BEGIN
         WRITELN(OUTPUT,' ');
         PRENEX
         END;
```

```
                                                    (* SEGMENT PROCEDURE NUM *)

        OUTDEVICE
       END;

IF(FLAG=1)
  THEN BEGIN
   PAGE(OUTPUT);
   WRITELN(OUTPUT,'WARNING . . . ');
   WRITELN(OUTPUT,'TREE NEEDS TO BE (RE)CALCULATED');
   WRITELN(OUTPUT,'OUTPUT VALUES MAY BE INCORRECT');
   WRITELN(OUTPUT);
   WRITELN(OUTPUT,'PRESS ANY KEY TO CONTINUE');
   REPEAT
    UNTIL KEYPRESS;
   READ(CH1)
   END;


  IF((CMD='NUM')OR(CMD='REV')) THEN NUMERICREVIEW
   ELSE IF (CMD='DIS') THEN DISPLAY;

 CLOSE(F);
 IF ((CH()'G')AND(ORD(CH)()27))
  THEN BEGIN
     WRITELN(OUTPUT);
     WRITELN(OUTPUT,'PRESS ANY KEY TO CONTINUE');
     REPEAT
      UNTIL KEYPRESS;
     READ(CH)
     END;
 END;  (* END  SEGMENT  NUM *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

SEGMENT PROCEDURE SENSITIVITY;

(* ************************************************************************
*                                                                         *
*  SEGMENT PROCEDURE SENSITIVITY                                          *
*  USE:  THIS SEGMENT CONTROLS ALL THE SENSITIVITY ANALYSIS THAT          *
*        IS IN THE DASS PROGRAM.  THE SEGMENT PROVIDES FOR                *
*        SENSITIVITY ON CUMULATIVE WEIGHTS (EFFECTS TO THE                *
*        TREE STRUCTURE), RELATIVE WEIGHTS (EFFECTS ON A GIVEN            *
*        SPAN AGAINST THE OVERALL TREE STRUCTURE, AND VALUE OF            *
*        A SPECIFIC ALTERNATIVE ON A SPECIFIC DATA NODE.  THE             *
*        SEGMENT PROVIDES GRAPHICAL OUTPUT ON THE MONITOR ONLY;           *
*        HOWEVER, THE SEGMENT PROVIDES TABULAR (NUMBERS) OUTPUT           *
*        TO EITHER THE CONSOLE OR THE LINE PRINTER.  THE                  *
*        SEGMENT ALSO HAS THE CAPABILITY TO PRESENT THE GRAPHICAL         *
*        OUTPUT IN TWO FORMS.  THE FIRST FORM GIVES THE RANGE OF          *
*        THE VALUES OF THE ALTERNATIVES AT THE NODE IN QUESTION           *
*        FROM 0-100, WHILE THE SECOND GIVES THE RANGE OF VALUES           *
*        OF THE ALTERNATIVES AS THE NEAREST 20 FROM THE LOWEST            *
*        AND HIGHEST VALUES OVER THE SENSITIVITY RANGE UNDER              *
*        CONSIDERATION.                                                   *
*  CALLED BY:  PROGRAM DASS                                               *
*  ROUTINES CALLED: SEGMENT PROCEDURE SENSITIVITY (WARNING,               *
*                          DETERMINENODE,CALCARRAY,TABDISPLAY,            *
*                          GRAPH)                                         *
*  VARIABLES:                                                             *
*       USED:  FLAG (see UNIT DASSA)                                      *
*       MODIFIED:  CH (see UNIT DASSA)                                    *
*                  CH1,SENS (see SEGMENT PROCEDURE SENSITIVITY)           *
*                                                                         *
************************************************************************ *)

VAR
 DELTA,W,WMIN,WMAX,WMAX1,WDELTA,MIN,MAX,X:REAL;
 SYSNUM,INDX,I,J,K,L:INTEGER;
 WHOLD: ARRAY [0..10,0..MAXSYSTEMS] OF REAL;
 SYSNAME,SENS:STRING;
 CH1:CHAR;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE ANYKEY;

(* ************************************************************
*                                                              *
*  PROCEDURE ANYKEY                                            *
*  USE:  SIMPLY WRITES OUT A MESSAGE TO DEPRESS ANY KEY        *
*        TO CONTINUE PROCESSING.  IS CALLED THOUGHOUT THIS     *
*        SEGMENT.                                              *
*  CALLED BY: SEGMENT PROCEDURE SENSITIVITY (SENVALUE,         *
*                    DETERMINENODE,WARNING)                    *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES                                                   *
*       USED: (none)                                           *
*       MODIFIED: CH (see UNIT DASSA)                          *
*                                                              *
************************************************************ *)

 BEGIN
  WRITELN(OUTPUT,'(ANY KEY) CONTINUE');
  REPEAT
   UNTIL KEYPRESS;
  READ(INPUT,CH)
 END;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE WARNING;

(* ************************************************************
*                                                              *
*  PROCEDURE WARNING                                           *
*  USE:  PRODUCES MESSAGE IF VALUES, WEIGHTS, OR               *
*        ALTERNATIVES HAVE BEEN ADDED/DELETED TO THE TREE      *
*        STRUCTURE AND THE TREE HAS NOT BEEN RECALCULATED      *
*        SINCE.                                                *
*  CALLED BY:  SEGMENT PROCEDURE SENSITIVITY                   *
*  ROUTINES CALLED: SEGMENT PROCEDURE SENSITIVITY (ANYKEY)     *
*  VARIABLES:                                                  *
*       USED: (none)                                           *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************** *)

 BEGIN
  WRITELN(OUTPUT,'WARNING . . .');
  WRITELN(OUTPUT,'TREE NEEDS TO BE (RE)CALCULATED');
  WRITELN(OUTPUT,'REPORTED VALUES MAY BE INCORRECT!');
  WRITELN(OUTPUT);
  ANYKEY
 END;
```

```
                                                (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE DETERMINENODE;

(* ************************************************************
*                                                              *
*  PROCEDURE DETERMINENODE                                     *
*  USE: SOLICITS INFORMATION FROM THE USER CONCERNING WHAT     *
*       NODE THE SENSITIVITY ANALYSIS IS TO TAKE PLACE.        *
*       THE PROCEDURE CONTAINS SAFEGUARDS THAT PREVENT         *
*       SENSITIVITY ANALYSIS BEING CONDUCTED ON NON-EXISTENT   *
*       NODES AND NON-DATA NODES WHEN VALUE SENSITIVITY IS     *
*       TO TAKE PLACE.                                         *
*       IN ADDITION, SOLICITS INFORMATION CONCERNING THE       *
*       MAXIMUM AND MINIMUM LIMITS THAT THE SENSITIVITY        *
*       ANALYSIS IS TO BE CONDUCTED                            *
*       THIS PROCEDURE HAS ONE SUBORDINATE PROCEDURE THAT      *
*       SOLICITS INFORMATION WHEN VALUE SENSITIVITY IS TO      *
*       BE CONDUCTED.                                          *
*  CALLED BY: SEGMENT PROCEDURE SENSITIVITY                    *
*  ROUTINES CALLED: UNIT DASSA (FRENEX,CROSS,STRTOREAL)        *
*                   SEGMENT PROCEDURE SENSITIVITY (SENVALUE,   *
*                          ANYKEY)                             *
*  VARIABLES                                                   *
*       USED: IRAY,ARAY,NODELABEL  (see UNIT DASSA)            *
*             SENS (see SEGMENT PROCEDURE SENSITIVITY)         *
*       MODIFIED: LVL,ICONT,ANSWER (see UNIT DASSA)            *
*                 INDX,I,WMAX,WMIN,WMAX1 (see SEGMENT          *
*                     PROCEDURE SENSITIVITY)                   *
*                                                              *
************************************************************** *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE SENVALUE;

(* ************************************************************************
*                                                                         *
*  PROCEDURE SENVALUE                                                     *
*  USE:  SUBORDINATE PROCEDURE TO DETERMINENODE.  THIS PROCEDURE          *
*        DISPLAYS ALTERNATIVES (SYSTEMS) WHICH ARE AVAILABLE              *
*        FOR VALUE SENSITIVITY ANALYSIS.  IN ADDITION, THE                *
*        PROCEDURE READS IN THE USER'S CHOICE AND DETERMINES IF           *
*        IF THE INPUT VALUE IS INDEED AN EXISTING ALTERNATIVE             *
*        (SYSTEM). AFTER A CORRECT ENTRY IS MADE, THE PROCEDURE           *
*        DISPLAYS THE CURRENT VALUE OF THE ALTERNATIVE (SYSTEM)           *
*        AT THE PARTICULAR DATA NODE.                                     *
*  CALLED BY: SEGMENT PROCEDURE SENSITIVITY (DETERMINENODE)               *
*  ROUTINES CALLED:  SEGMENT PROCEDURE SENSITIVITY (ANYKEY)               *
*  VARIABLES:                                                             *
*       USED: SYSTEMS:SYSTEMNAME, NSYS, TENSTRING, VRAY, LVL,             *
*             MAXLABELSIZE (see UNIT DASSA)                               *
*       MODIFIED: ANSWER (see UNIT DASSA)                                 *
*                 SYSNUM,SYSNAME,I (see SEGMENT PROCEDURE                 *
*                         SENSITIVITY)                                    *
*                                                                         *
************************************************************************ *)

  BEGIN
    REPEAT
     BEGIN

      WRITELN(OUTPUT);
      WRITELN(OUTPUT,'SYSTEMS AVAILABLE');
      FOR I:=1 TO NSYS DO
        BEGIN
         SEEK(SYSTEMS,I-1);
         GET(SYSTEMS);
         WITH SYSTEMS^ DO WRITELN(OUTPUT,' ',SYSTEMNAME)
        END;
      WRITELN(OUTPUT);
      WRITELN(OUTPUT,'ENTER SYSTEM OF WHICH VALUE IS');
      WRITELN(OUTPUT,'TO BE PRETURBED. . .');
      READLN(INPUT,ANSWER);
      ANSWER:=COPY(CONCAT(ANSWER,TENSTRING),1,MAXLABELSIZE);
      SYSNUM:=0;
      FOR I:=1 TO NSYS DO
       BEGIN
        SEEK(SYSTEMS,I-1);
        GET(SYSTEMS);
        WITH SYSTEMS^ DO
          IF(ANSWER=SYSTEMNAME) THEN SYSNUM:=I;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

        END;
      SYSNAME:=ANSWER;

      IF (SYSNUM=0)
       THEN
        BEGIN
         WRITELN(OUTPUT,'SYSTEM ENTERED NOT VALID');
         ANYKEY;
         PAGE(OUTPUT)
        END;
     END
   UNTIL(SYSNUM<>0);
     WRITELN(OUTPUT,'CURRENT NODE VALUE IS ',VRAY[LVL,SYSNUM]:5:2);

END;  (* END SENVALUE *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

(* ************************************************
*                                                  *
*         BEGIN   PROCEDURE  DETERMINENODE         *
*                                                  *
************************************************** *)

BEGIN
    INOX:=0;
    REPEAT
      BEGIN
       PAGE(OUTPUT);
       WRITELN(OUTPUT,'SENSITIVITY ANALYSIS FOLLOWS:');
       WRITELN(OUTPUT,'NRN FOR WHICH ',SENS,' IS');
       WRITELN(OUTPUT,'TO BE PRETURBED');

       PRENEX;

       IF(ICONT=0)
        THEN
          WRITELN(OUTPUT,'NODE DOES NOT EXIST...')

        ELSE
         IF((COPY(SENS,1,1)='V')AND(IRAY[LVL,2]<>0))
           THEN
            BEGIN
             WRITELN(OUTPUT,'NODE NOT DATA NODE...');
             ICONT:=0
            END;

       IF (ICONT=0) THEN ANYKEY;

       END

     UNTIL(ICONT=1);

     FOR I:=1 TO LVL DO WRITE(OUTPUT,IRAY[I,1],' ');
     WRITELN(OUTPUT,NODELABEL[LVL]);
     IF(COPY(SENS,1,1)='C')
      THEN
       WRITELN(OUTPUT,'CURRENT NODE CUMWT IS',ARAY[LVL,2]:5:2)
      ELSE
       IF(COPY(SENS,1,1)='V')
        THEN SENVALUE

       ELSE
        BEGIN
         WRITELN(OUTPUT,'CURRENT NODE RELWT IS',ARAY[LVL,1]:5:2);
         INOX:=IRAY[LVL,1]; LVL:=LVL-1;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

      CROSS
    END;

  IF(SENS='VALUE') THEN WMAX:=100
                   ELSE WMAX:=1;
  REPEAT
   BEGIN
    WRITE(OUTPUT,'MINIMUM ',SENS,' (0 - ',WMAX:5:1,') IS ? ');
    READLN(INPUT,ANSWER);
    WMIN:=STRTOREAL(ANSWER)
   END
  UNTIL((WMIN)=0)AND(((WMIN<=1.0)AND(SENS()'VALUE'))OR
                    ((WMIN<=100.0)AND(SENS='VALUE'))));

  REPEAT
   BEGIN
    WRITE(OUTPUT,'MAXIMUM ',SENS,'(',WMIN:5:1,' -',WMAX:5:1,') IS ? ');
    READLN(INPUT,ANSWER);
    WMAX1:=STRTOREAL(ANSWER)
   END
  UNTIL((WMAX1)=WMIN)AND(((WMAX1<=1.0)AND(SENS()'VALUE'))OR
                        ((WMAX1<=100.0)AND(SENS='VALUE'))));

  WMAX:=WMAX1

END;    (* END PROCEDURE DETERMINENODE *)
```

```
                                                    (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE CALCARRAY;

(* ************************************************************************
*                                                                         *
*  PROCEDURE CALCARRAY                                                    *
*  USE:  THIS PROCEDURE CALCULATES THE SENSITIVITY ANALYSIS VALUES        *
*        FOR ALL OPTIONS CURRENTLY IN DASS. PLACES THE ANALYSIS           *
*        VALUES IN THE WHOLD ARRAY TO FACILITATE DISPLAY IN EITHER        *
*        THE GRAPHICAL OR TABULAR MODE. THE VARIABLE CONTROL              *
*        DETERMINES THE SENSITIVITY WHICH IS TO BE ACCOMPLISHED           *
*        BASED ON USER INPUT.                                             *
*                    CONTROL = 1  . . . CUMULATIVE SENSITIVITY            *
*                    CONTROL = 2  . . . VALUE SENSITIVITY                 *
*                    CONTROL = 3  . . . RELATIVE SENSITIVITY              *
*        THE ACTUAL CALCULATIONS OF THE VALUES AS THE SENSITIVITY         *
*        ANALYSIS IS BEING CONDUCTED IS DONE VIA THE CASE                 *
*        STATEMENT.                                                       *
*        THE RELATIVE WEIGHT SENSITIVITY ANALYSIS IS CONDUCTED            *
*        BY VARYING THE NODE DESIRED OVER THE INPUTTED RANGE AND          *
*        THE OTHER SIBLINGS RECOMPUTED TO MAINTAIN RELATIVE               *
*        IMPORTANCE AMONG THEMSELVES AND COMPLEMENTING THE                *
*        SENSITIZED NODE.                                                 *
*  CALLED BY:  SEGMENT PROCEDURE SENSITIVITY                              *
*  ROUTINES CALLED: (none)                                                *
*  VARIABLES:                                                             *
*       USED:  NCROSS,IRAY,ARAY,LVL,NSYS (see UNIT DASSA)                 *
*              SENS,INDX,SYSNUM (see SEGMENT PROCEDURE SENSITIVITY)       *
*       MODIFIED: WDELTA,WMAX,WMIN,I,X,WHOLD,L,K (see SEGMENT             *
*                       PROCEDURE SENSITIVITY)                            *
*                 SUM,VTEMP,NEWSUM,CONTROL (see PROCEDURE                 *
*                       CALCARRAY)                                        *
*                                                                         *
************************************************************************* *)

 VAR
  VTEMP,NEWSUM,SUM:REAL;
  CONTROL:INTEGER;


BEGIN

 WDELTA:=(WMAX-WMIN)/10;
 IF(SENS='CUMWT') THEN CONTROL:=1
 ELSE IF(SENS='VALUE') THEN CONTROL:=2
 ELSE CONTROL:=3;
 IF(CONTROL=3)
  THEN
    BEGIN
```

```
                                                (* SEGMENT PROCEDURE SENSITIVITY *)
        SUM:=0;
        FOR I:=1 TO NCROSS DO
         IF(INDX()IRAY[LVL+I,1])
         THEN SUM:=SUM+ARAY[LVL+I,1]
       END;

    FOR I:=0 TO 10 DO
     BEGIN
      X:=I;
      X:=X*WDELTA+WMIN;
      WHOLD[I,0]:=X;
      FOR K:=1 TO NSYS DO
      CASE CONTROL OF

      1: WHOLD[I,K]:=(VRAY[LVL,K]*X)+((1-X)*((VRAY[1,K]-(VRAY[LVL,K]*
         ARAY[LVL,2]))/(1-ARAY[LVL,2])));

      2: IF(K=SYSNUM) THEN WHOLD[I,K]:=VRAY[1,K]+ARAY[LVL,2]*(X-VRAY[LVL,K])
                      ELSE WHOLD[I,K]:=VRAY[1,K];
      3: BEGIN
          WHOLD[I,K]:=VRAY[1,K]-VRAY[LVL,K]*ARAY[LVL,2];

          FOR L:=1 TO NCROSS DO
           IF(INDX()L)
            THEN WHOLD[I,K]:=WHOLD[I,K]+(1-X)*VRAY[LVL+L,K]*ARAY[LVL+L,1]/SUM*
                 ARAY[LVL,2]

            ELSE WHOLD[I,K]:=WHOLD[I,K]+X*VRAY[LVL+L,K]*ARAY[LVL,2]

         END

      END (* END CASES AND K *)

     END;
    WRITELN(OUTPUT)
  END;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE HEADERS(I:INTEGER);

(* ************************************************************
*                                                              *
*  PROCEDURE HEADERS                                           *
*  USE:  PROVIDES HEADINGS TO THE SENSITIVITY ANALYSIS WHEN    *
*        TABULARIZED RESULTS ARE REQUESTED THESE HEADINGS      *
*        STATE THE TYPE OF SENSITIVITY (INCLUDING ALTERNATIVE  *
*        IF VALUE SENSITIVITY IS BEING DONE) AND THE NODE      *
*        REFERENCE NUMBER (NRN).                               *
*  CALLED BY:  SEGMENT PROCEDURE SENSITIVITY (TABDISPLAY,      *
*                        GRAPH)                                *
*  ROUTINES CALLED: DASSA (INTTOSTRING)                        *
*  VARIABLES:                                                  *
*       USED: LVL,ISTR,IRAY,NODELABEL (see UNIT DASSA)         *
*             SENS,INDX (see SEGMENT PROCEDURE SENSITIVITY)    *
*       MODIFIED: ANSWER (see UNIT DASSA)                      *
*                 I (see SEGMENT PROCEDURE SENSITIVITY)        *
*                                                              *
************************************************************ *)

BEGIN
   ANSWER:='';
 CASE I OF

 1: BEGIN
    IF(SENS='VALUE') THEN ANSWER:=CONCAT(' (',SYSNAME,')');
    ANSWER:=CONCAT(SENS,ANSWER,' SENSITIVITY ANALYSIS');
    END;

 2: BEGIN
    ANSWER:=CONCAT('FOR ',NODELABEL[LVL+INDX],'  NRN: ');
    FOR I:=1 TO LVL DO
      BEGIN
       INTTOSTRING(IRAY[I,11);
       ANSWER:=CONCAT(ANSWER,ISTR,' ')
      END;
    IF(INDX<>0)
     THEN
      BEGIN
       INTTOSTRING(IRAY[LVL+INDX,11);
       ANSWER:=CONCAT(ANSWER,ISTR,' ')
      END;
    END
  END  (* END CASE *)
END;  (* END PROCEDURE HEADERS *)
```

```
                                                    (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE TABDISPLAY;

(* ************************************************************************
*                                                                         *
*  PROCEDURE TABDISPLAY                                                   *
*  USE: SOLICITS WHETHER THE SENSITIVITY ANALYSIS IS TO BE                *
*       OUTPUTTED ON THE CONSOLE OR THE LINE PRINTER.  THE                *
*       PROCEDURE THEN PRINTS UP APPROPRIATE HEADERS AND LISTS            *
*       NUMERICALLY THE RESULTS OF THE SENSITIVITY WITH THE               *
*       SENSITIVITY VALUES DOWN THE ROWS AND THE OVERALL VALUES           *
*       FOR EACH ALTERNATIVE (SYSTEM) ACROSS THE COLUMNS.                 *
*       AN ASTERISK (*) IS PRINTED BY THE ALTERNATIVE WHICH HAS           *
*       THE LOWEST VALUE IF THE ATTRIBUTE CHARACTERISTIC UNDER            *
*       OPTION ATT STARTS WITH AN R.  AN ASTERISK (*) IS PRINTED          *
*       BY THE ALTERNATIVE WHICH HAS THE HIGHEST VALUE IF THE             *
*       ATTRIBUTE CHARACTERISTIC UNDER OPTION ATT STARTS WITH             *
*       ANYTHING BUT AN R.                                                *
*  CALLED BY:  SEGMENT PROCEDURE SENSITIVITY                              *
*  ROUTINES CALLED: SEGMENT PROCEDURE SENSITIVITY (ANYKEY)                *
*  VARIABLES:                                                             *
*       USED: LATT,NSYS,SYSTEMS:SYSTEMNAME (see UNIT DASSA)               *
*             SENS,WHOLD (see SEGMENT PROCEDURE SENSITIVITY)              *
*       MODIFIED:  ANSWER,CH  (see UNIT DASSA)                            *
*                  K,I  (see SEGMENT PROCEDURE SENSITIVITY)               *
*                  F,A,W,STAR (see PROCEDURE TABDISPLAY)                  *
*                                                                         *
************************************************************************ *)
 VAR
   F:FILE OF CHAR;
   A:STRING;
   W:REAL;
   STAR:INTEGER;

 BEGIN
  REPEAT
   BEGIN
    PAGE(OUTPUT);
    WRITE(OUTPUT,'TABULAR:  C(ONSOLE  P(RINTER');
    REPEAT
     UNTIL KEYPRESS;
    READ(INPUT,CH)
   END
  UNTIL((CH='C')OR(CH='P'));

  IF(CH='P')
    THEN REWRITE(F,'PRINTER:')
    ELSE REWRITE(F,'CONSOLE:');
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

A:=COPY(LATT,1,1);

PAGE(F);
HEADERS(1); WRITELN(F,ANSWER);
HEADERS(2); WRITELN(F,ANSWER);

ANSWER:=CONCAT(SENS,': ');
FOR I:=1 TO NSYS DO
 BEGIN
  SEEK(SYSTEMS,I-1);
  GET(SYSTEMS);
  WITH SYSTEMS^ DO ANSWER:=CONCAT(ANSWER,COPY(SYSTEMNAME,1,6),' ');
 END;
WRITELN(F,ANSWER);
WRITELN(F);

FOR I:=0 TO 10 DO
 BEGIN
  IF(A='R') THEN W:=1000
            ELSE W:=0;
  STAR:=0;
  FOR K:=1 TO NSYS DO
   IF(((WHOLD[I,K]>W)AND(A<>'R'))OR
     ((WHOLD[I,K]<W)AND(A='R')))
     THEN BEGIN STAR:=K; W:=WHOLD[I,K] END;

  WRITE(F,WHOLD[I,0]:4:2,' ');
  FOR K:=1 TO NSYS DO
   IF(K=STAR)
    THEN WRITE(F,WHOLD[I,K]:6:1,'*')
    ELSE WRITE(F,WHOLD[I,K]:6:1,' ');
  WRITELN(F);
 END;

 IF(CH='C')
  THEN
   BEGIN
    WRITELN(F);
    ANYKEY
   END;

 CLOSE(F),
END
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE GRAPH;

(* ************************************************************
*                                                              *
*  PROCEDURE GRAPH                                             *
*  USE:  DISPLAYS THE SENSITIVITY ANALYSIS IN GRAPHICS MODE    *
*        ON THE MONITOR.  DRAWS THE SENSITIVITY ANALYSIS       *
*        IN A GRAPHICAL FORMAT EACH ALTERNATIVE (SYSTEM)       *
*        HAVING A DISTINCTIVE COLOR:                           *
*                ALTERNATIVE 1     ORANGE                      *
*                ALTERNATIVE 2     VIOLET                      *
*                ALTERNATIVE 3     BLUE                        *
*                ALTERNATIVE 4     GREEN                       *
*                ALTERNATIVE 5     WHITE)                      *
*        PROCEDURE USES ONE SUBORDINATE PROCEDURE              *
*        (GRAPHHEADER) WHICH LABELS THE X-AXIS OF THE GRAPH    *
*  CALLED BY:  SEGMENT PROCEDURE SENSITIVITY                   *
*  ROUTINES CALLED:  UNIT DASSA (NUMTOSTRING)                  *
*                    SEGMENT PROCEDURE SENSITIVITY             *
*                         (GRAPHHEADER)                        *
*  VARIABLES:                                                  *
*       USED: NSYS,SYSTEM:SYSTEMNAME (see UNIT DASSA)          *
*       MODIFIED:  CH (see UNIT DASSA)                         *
*                  I,J (see SEGMENT PROCEDURE SENSITIVITY)     *
*                  COLOR,K,I1,J1 (see PROCEDURE GRAPH)         *
*                                                              *
************************************************************ *)

VAR
     K,I1,J1:INTEGER;
     COLOR: ARRAY [1..5] OF SCREENCOLOR;
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

PROCEDURE GRAPHHEADER;

(* ************************************************************
*                                                              *
*  PROCEDURE GRAPHHEADER                                       *
*  USE:  SUBORDINATE PROCEDURE TO PROCEDURE GRAPH.  THIS       *
*        PROCEDURE PRINTS THE VALUES OF THE LINES OF THE       *
*        GRAPH WHICH IS OUTPUTTED IN THE GRAPHICS MODE.        *
*        PROCEDURE WRITES OUT THE VALUES OF THE X-AXIS.        *
*  CALLED BY: SEGMENT PROCEDURE SENSITIVITY (GRAPH)            *
*  ROUTINES CALLED: UNIT DASSA (NUMTOSTRING)                   *
*  VARIABLES:                                                  *
*       USED: CH,NSYS (see UNIT DASSA)                         *
*             WHOLD (see SEGMENT PROCEDURE SENSITIVITY)        *
*      MODIFIED: ANSWER (see UNIT DASSA)                       *
*                MIN,MAX,I,K,DELTA,X (see SEGMENT PROCEDURE    *
*                     SENSITIVITY)                             *
*                                                              *
************************************************************ *)
BEGIN
     MIN:=100;  MAX:=0;
     IF(CH='E')
     THEN
     BEGIN
     FOR I:=0 TO 10 DO
     FOR K:=1 TO NSYS DO
      BEGIN
       IF(MIN)WHOLD[I,K]) THEN MIN:=WHOLD[I,K];
       IF(MAX(WHOLD[I,K]) THEN MAX:=WHOLD[I,K]
      END
     END

     ELSE
      BEGIN
       MIN:=0;  MAX:=100;
      END;
     MIN:=TRUNC(MIN/20)*20;  MAX:=TRUNC((MAX+19)/20)*20;
     ANSWER:='';  DELTA:=(MAX-MIN)/5;

     FOR I:=0 TO 5 DO
      BEGIN
       X:=I;    X:=MIN+X*DELTA;
       NUMTOSTRING(X);    ANSWER:=CONCAT(ANSWER,'    ',COPY(ISTR,1,3))
      END;

  PENCOLOR(NONE);    MOVETO(28,108);    WSTRING(ANSWER);
END;  (* END GRAPHHEADER *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

(* ********************************************************
*                                                          *
*            BEGIN   PROCEDURE   GRAPH                     *
*                                                          *
******************************************************** *)

BEGIN
  COLOR[1]:=ORANGE; COLOR[2]:=VIOLET;
  COLOR[3]:=BLUE;   COLOR[4]:=GREEN;
  COLOR[5]:=WHITE1;

  REPEAT
   BEGIN
    PAGE(OUTPUT);
    WRITE(OUTPUT,'GRAPHIC:  N(ORMAL  E(XPANDED ');
    REPEAT
     UNTIL KEYPRESS;
    READ(INPUT,CH)
   END
  UNTIL((CH='N')OR(CH='E'));

  INITTURTLE;

  (* PRINT THE COLOR BARS DEPICTING ALTERNATIVES *
  *  IN UPPER LEFT HAND CORNER                   *)

  FOR I:=1 TO NSYS DO
   BEGIN
    SEEK(SYSTEMS,I-1); GET(SYSTEMS);
    CASE I OF

     1: BEGIN VIEWPORT(0,76,176,191); FILLSCREEN(ORANGE);
        MOVETO (2,179);  WITH SYSTEMS^ DO WSTRING(SYSTEMNAME) END;
     2: BEGIN VIEWPORT(0,76,161,175); FILLSCREEN(VIOLET);
        MOVETO (2,164);  WITH SYSTEMS^ DO WSTRING(SYSTEMNAME) END;
     3: BEGIN VIEWPORT(77,153,176,191); FILLSCREEN(BLUE);
        MOVETO (78,179); WITH SYSTEMS^ DO WSTRING(SYSTEMNAME) END;
     4: BEGIN VIEWPORT(77,153,161,175); FILLSCREEN(GREEN);
        MOVETO (78,164); WITH SYSTEMS^ DO WSTRING(SYSTEMNAME) END;
     5: BEGIN VIEWPORT(154,231,176,191); FILLSCREEN(WHITE1);
        MOVETO (160,179); WITH SYSTEMS^ DO WSTRING(SYSTEMNAME) END

    END (* END CASES *)
   END;

  VIEWPORT(0,279,0,191);

  (*  DRAW THE GRAPH  *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

  J:=6; I:=70;
  MOVETO(I,J);
  FOR K:=1 TO 3 DO
   BEGIN
    PENCOLOR(WHITE);
    MOVETO(I,J+100);
    MOVETO(I+40,J+100);
    MOVETO(I+40,J);
    IF(K()3) THEN MOVETO(I+80,J);
    I:=I+80
   END;

  I:=70;
  FOR K:=1 TO 3 DO
   BEGIN
    MOVETO(I,J);
    MOVETO(I,J+20);
    MOVETO(I+200,J+20);
    IF(K()3) THEN MOVETO(I+200,J+40);
    J:=J+40
   END;


(*   NUMBER THE  X-AXIS   *)

GRAPHHEADER;

(*  LABEL THE GRAPH AXIS  *)

MOVETO(149,120); WSTRING('VALUE');

J:=71;
FOR I:=0 TO 4 DO
 BEGIN
  MOVETO(12,J-I*11);
  WSTRING(COPY(SENS,I+1,1))
 END;

(*  WRITE THE NUMERIC VALUES ON THE Y AXIS    *)

I:=0;
REPEAT
 BEGIN
 NUMTOSTRING(WHOLD(I,0));
 MOVETO(23,101-I*10);
 WSTRING(ISTR);
 I:=I+2
```

```
 END
UNTIL(I)10);

(*  WRITE OUT SENSITIVITY HEADINGS  *)

HEADERS(1); MOVETO(0,141); WSTRING(ANSWER);
HEADERS(2); MOVETO(0,130); WSTRING(ANSWER);

(*  DRAW THE LINES ON THE GRAPH  *)

 FOR I:=1 TO NSYS DO
  BEGIN
   J1:=106;
   X:=200/(MAX-MIN);
   FOR J:=0 TO 10 DO
    BEGIN
     I1:=TRUNC((WHOLD[J,I]-MIN)*X)+70;
     IF (J=0)
      THEN PENCOLOR(NONE)
      ELSE PENCOLOR(COLOR[I]);
     MOVETO(I1,J1);
     J1:=J1-10
    END
  END;

  REPEAT
   UNTIL KEYPRESS;
  READ(INPUT,CH);
  TEXTMODE
END; (* END GRAPH *)
```

```
                                        (* SEGMENT PROCEDURE SENSITIVITY *)

(* ************************************************
*                                                  *
*       BEGIN  SEGMENT PROCEDURE SENSITIVITY       *
*                                                  *
************************************************ ******* *)

BEGIN
 IF(FLAG=1) THEN WARNING;

 REPEAT
  BEGIN
   PAGE(OUTPUT);
   WRITE(OUTPUT,'SENSITIVITY:  C)UMWT R)ELWT V)ALUE E)XIT ');
   REPEAT
    UNTIL KEYPRESS;
   READ(INPUT,CH)
  END
 UNTIL((CH='C')OR(CH='R')OR(CH='V')OR(CH='E'));

 IF(CH()'E')
  THEN
   BEGIN
    IF(CH='C') THEN SENS:='CUMWT'
    ELSE IF(CH='R') THEN SENS:='RELWT'
    ELSE SENS:='VALUE';

    DETERMINENODE;
    CALCARRAY;

    REPEAT
     BEGIN
      PAGE(OUTPUT);
      WRITE(OUTPUT,SENS,':  T)ABULAR G)RAPHICAL E)XIT ');
      REPEAT
       UNTIL KEYPRESS;
      READ(INPUT,CH1);
      IF(CH1='T')THEN TABDISPLAY
                 ELSE IF(CH1='G') THEN GRAPH;
     END
    UNTIL(CH1='E')
   END

 END;  (* END SEGMENT SENSITIVITY *)
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

SEGMENT PROCEDURE READSYSTEMLABELS;

(* ************************************************************
*                                                              *
*  SEGMENT PROCEDURE SYSTEMLABELS                              *
*  USE:  THIS SEGMENT CONTROLS THE NUMBER AND LABELS OF THE    *
*        ALTERNATIVES USED IN THE DASS PROGRAM.  FUNCTIONS     *
*        ARE ACCOMPLISHED BY THIS SEGMENT IS THE ADDITION,     *
*        DELETION, AND CREATION OF ALTERNATIVES.  THERE ARE    *
*        THREE SUBORDINATE PROCEDURES WHICH SUPPORT THIS       *
*        SEGMENT.                                              *
*  CALLED BY: PROGRAM DASS                                     *
*  ROUTINES CALLED: UNIT DASSA (NODEDISKTOARRAY)               *
*                   SEGMENT PROCEDURE SYSTEMLABELS (NEW,       *
*                        DELSYS,ADDSYS)                        *
*  VARIABLES:                                                  *
*       USED: NODE (see UNIT DASSA)                            *
*       MODIFIED: CH,LVL (see UNIT DASSA)                      *
*                                                              *
************************************************************* *)
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

PROCEDURE NEW;

(* ************************************************************
*                                                             *
*  PROCEDURE NEW                                              *
*  USE:  THIS PROCEDURE SOLICITS NAMES FOR ALTERNATIVES       *
*        (SYSTEMS).  THIS ROUTINE IS NORMALLY SELECTED WHEN   *
*        A NEW TREE IS CONSTRUCTED AND A LIST OF ALTERNATIVES *
*        IS DESIRED TO BE ENTERED.                            *
*        THIS PROCEDURE CAN BE USED ANYTIME DURING THE        *
*        THE COURSE OF THE PROGRAM.  DATA IN THE TREE WILL    *
*        NOT BE LOST IF THE NUMBER OF LABELS ENTERED ARE THE  *
*        SAME OF THE NUMBER OF LABELS OF THE ALTERNATIVES     *
*        (SYSTEMS) IN THE ORIGINAL STRUCTURE.  IF FEWER       *
*        LABELS ARE USED, ONLY THE DATA FOR THE LABELS ENTERED*
*        WILL BE ACCESSIBLE.                                  *
*  CALLED BY: SEGMENT PROCEDURE READSYSTEMLABELS              *
*  ROUTINES CALLED: (none)                                    *
*  VARIABLES:                                                 *
*       USED: LABELSTRING (see UNIT DASSA)                    *
*       MODIFIED: NSYS,SYSTEMS:SYSTEMNAME (see UNIT DASSA)    *
*                 SYSLABEL (see PROCEDURE NEW)                *
*                                                             *
************************************************************* *)

VAR
    SYSLABEL : STRING;

BEGIN

    PAGE(OUTPUT);
    NSYS:=1;

    REPEAT

     WRITE(OUTPUT,'ENTER...SYSTEM ',NSYS,' LABEL? ');
     READLN(INPUT,SYSLABEL);
     IF((SYSLABEL()'DONE')AND(SYSLABEL()''))

       THEN
       BEGIN
         SYSLABEL:=COPY(CONCAT(SYSLABEL,LABELSTRING),1,10);
         SEEK(SYSTEMS,NSYS-1);
         WITH SYSTEMS^ DO SYSTEMNAME:=SYSLABEL;
         PUT(SYSTEMS);
         NSYS:=NSYS+1;
         IF(NSYS)5) THEN SYSLABEL:='';
       END
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

   ELSE
     IF (NSYS=1)
     THEN WRITELN(OUTPUT,'SORRY...YOU MUST ENTER AT LEAST ONE SYSTEM');

UNTIL(((SYSLABEL='DONE')OR(SYSLABEL=''))AND(NSYS>1));

NSYS:=NSYS-1;
WRITELN(OUTPUT,'NUM OF SYS ',NSYS)

END;    (* END NEW *)
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

PROCEDURE DELSYS;

(* ************************************************************
*                                                              *
*  PROCEDURE DELSYS                                            *
*  USE: DELETES USER SPECIFIED ALTERNATIVE (SYSTEM) AND        *
*       REMOVES THE LABEL AND ALL INFORMATION REGARDING THE    *
*       ALTERNATIVE (SYSTEM) FROM THE TREE STRUCTURE.          *
*  CALLED BY:  SEGMENT PROCEDURE READSYSTEMLABELS              *
*  ROUTINES CALLED:  (none)                                    *
*  VARIABLES:                                                  *
*       USED: TENSTRING,MAXLABELSIZE,MAXSYSTEMS,NNODES         *
*                  (see UNIT DASSA)                            *
*       MODIFIED: NODE:SYSTEMVALUE,SYSTEM:SYSTEMNAME,          *
*                  NSYS,I,ANSWER,ISTR,L (see UNIT DASSA)       *
*                SYS,J (see PROCEDURE DELSYS)                  *
*                                                              *
************************************************************ *)

VAR
      SYS:ARRAY [1..MAXSYSTEMS] OF STRING;
      J:INTEGER;
BEGIN
    PAGE(OUTPUT);
    WRITELN(OUTPUT,'CURRENT SYSTEMS. . .');
    SEEK(SYSTEMS,0);
    FOR I:=1 TO NSYS DO
     BEGIN
      GET(SYSTEMS);
      WITH SYSTEMS^ DO
       BEGIN
       SYS[I]:=SYSTEMNAME;
       WRITELN(OUTPUT,SYSTEMNAME)
       END
     END;
    WRITELN(OUTPUT,'ENTER SYSTEM TO BE DELETED ');
    READLN(INPUT,ANSWER);
    ANSWER:=COPY(CONCAT(ANSWER,TENSTRING),1,MAXLABELSIZE);
    ISTR:='';
    FOR I:=1 TO NSYS DO
      IF(SYS[I]=ANSWER)
       THEN BEGIN
        ISTR:=ANSWER;
        L:=I
        END;
    IF(ISTR='')
      THEN WRITELN(OUTPUT,'SYSTEM NOT FOUND')
      ELSE
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

        BEGIN
         FOR I:=L TO (MAXSYSTEMS-1) DO
          BEGIN
           SEEK(SYSTEMS,I);
           GET(SYSTEMS);
           SEEK(SYSTEMS,I-1);
           PUT(SYSTEMS);
           WITH SYSTEMS^ DO SYSTEMNAME:='NONE';
           PUT(SYSTEMS);
          END;

         FOR I:=1 TO NNODES DO
          BEGIN
           SEEK(NODE,I); GET(NODE);
           WITH NODE^ DO
            FOR J:=L TO (MAXSYSTEMS-1) DO
              BEGIN
              SYSTEMVALUE[J]:=SYSTEMVALUE[J+1];
              SYSTEMVALUE[J+1]:=0
              END;
           SEEK(NODE,I); PUT(NODE)
          END;
      NSYS:=NSYS-1
     END
END;
```

```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

PROCEDURE ADDSYS;

(* ************************************************************
*                                                              *
*  PROCEDURE ADDSYS                                            *
*  USE: ADDS AN ALTERNATIVE (SYSTEM) LABEL TO THE TREE         *
*       STRUCTURE.  CHECKS TO MAKE SURE THAT NO MORE THAN      *
*       THE MAXIMUM NUMBER OF SYSTEMS CAN BE ADDED TO THE      *
*       STRUCTURE AND PROVIDES A MESSAGE TO INSERT VALUES      *
*       FOR THE NEW ALTERNATIVE (SYSTEM).                      *
*  CALLED BY: SEGMENT PROCEDURE READSYSTEMLABELS               *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: MAXSYSTEMS,TENSTRING,MAXLABELSIZE (see UNIT      *
*                  DASSA)                                      *
*       MODIFIED: NSYS,FLAG,ANSWER,SYSTEMS:SYSTEMNAME (see     *
*                  UNIT DASSA)                                 *
*                                                              *
************************************************************* *)

BEGIN
   WRITELN(OUTPUT,' ');
   IF(NSYS=MAXSYSTEMS)
     THEN WRITELN(OUTPUT,'MAX NUMBER OF SYSTEMS EXCEEDED')
     ELSE
       BEGIN
        NSYS:=NSYS+1;
        SEEK(SYSTEMS,NSYS-1);
        PAGE(OUTPUT);
        WRITELN(OUTPUT,'ADDING SYSTEM');
        WRITE(OUTPUT,'LABEL? ');
        WITH SYSTEMS^ DO
         BEGIN
          READLN(INPUT,ANSWER);
          SYSTEMNAME:=COPY(CONCAT(ANSWER,TENSTRING),1,MAXLABELSIZE)
         END;
        PUT(SYSTEMS);
        WRITELN(OUTPUT,'USE WVC FOR ENTERING VALUES AND');
        WRITELN(OUTPUT,'RECALCULATING TREE');
        FLAG:=1;
        END
END;
```

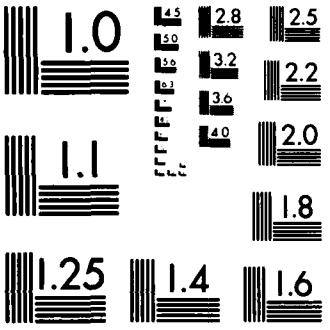
```
                                        (* SEGMENT PROCEDURE READSYSTEMLABELS *)

(* ************************************************
*                                                  *
*     BEGIN SEGMENT PROCEDURE READSYSTEMLABELS     *
*                                                  *
************************************************** *)

BEGIN
   REPEAT

   BEGIN
    PAGE(OUTPUT);
    WRITE(OUTPUT,'A(DD D(ELETE N(EW E(XIT');
    REPEAT
     UNTIL KEYPRESS;
    READ(CH);
    IF(CH='N') THEN NEW
    ELSE IF(CH='D') THEN DELSYS
    ELSE IF(CH='A') THEN ADDSYS;
    LVL:=1;
    SEEK(NODE,1);
    GET(NODE);
    NODEDISKTOARRAY(LVL)
   END

   UNTIL(CH='E');
   WRITELN(OUTPUT,' ')

END;
```

AD-A101 140 ENHANCED DECISION ANALYSIS SUPPORT SYSTEM(U) AIR FORCE 4/4
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING D B LEE MAR 81 AFIT/GST/OS/81M-8

UNCLASSIFIED F/G 9/2 NL

END

FILMED
6 82
DTIC



```
                                                    (* SEGMENT PROCEDURE MODPRU *)

SEGMENT PROCEDURE MODPRU;

(* ***************************************************************
*                                                                 *
*  SEGMENT PROCEDURE MODPRU                                       *
*  USE:  THIS SEGMENT SUPPORTS OPTIONS MOD AND PRU (MODIFY THE TREE *
*        STRUCTURE AND PRUNE THE TREE STRUCTURE, RESPECTIVELY).   *
*        PROCEDURE MODIFY HAS NO SUBORDINATE PROCEDURES, WHEREAS  *
*        PROCEDURE PRUNE HAS FOUR.                                *
*  CALLED BY: PROGRAM DASS                                        *
*  ROUTINES CALLED:  SEGMENT PROCEDURE MODPRU (PRUNE,MODIFY)      *
*  VARIABLES:                                                     *
*       USED: CMD (see UNIT DASSA)                                *
*       MODIFIED:  (none)                                         *
*                                                                 *
*************************************************************** *)
```

```
                                                    (* SEGMENT PROCEDURE MODPRU *)

PROCEDURE PRUNE;

(* ***************************************************************
*                                                                 *
*  PROCEDURE PRUNE                                                *
*  USE:  USED TO PERMANENTLY REMOVES NODES AND BRANCHES FROM THE  *
*        TREE STRUCTURE.  ACCOMPLISHES THIS TASK BY LOADING ALL   *
*        NODE LINKING INFORMATION INTO AN ARRAY (PRAY), IDENTIFIES*
*        NODE(S) TO BE ELIMINATED, RELINKS THE REMAINING STRUCTURE,*
*        REMOVES THE PRUNED NODES, COMPRESSES THE TREE STRUCTURE  *
*        SUCH THAT THE NODES PRUNED CAN NOW BE AVAILABLE FOR RE-  *
*        ASSIGNMENTS, AND REWRITES THE TREE STRUCTURE ON THE DATA *
*        FILE.                                                    *
*  CALLED BY: SEGMENT PROCEDURE MODPRU                            *
*  ROUTINES CALLED: UNIT DASSA (PRENEX,NEXT,PRETOT,NODEDISKTOARRAY)*
*                   SEGMENT PROCEDURE MODPRU (WDOT,RELINK,COMPRESS,*
*                           REWRITE)                               *
*  VARIABLES:                                                     *
*       USED:  CH,NNODES,ICONT,NODE:[CELLNUMBER,NRNDIGIT,DOWNLINK,*
*              CROSSLINK,BACKLINK],MAXARRAYSIZE                   *
*              MAXNUMBEROFNODES (see UNIT DASSA)                  *
*       MODIFIED: CH,IRAY,I,LVL,NDEEP,FLAG (see UNIT DASSA)       *
*                 PRAY,PRNUM,NEWNNODES,NEWLVL (see PROCEDURE PRUNE)*
*                                                                 *
*************************************************************** *)

 VAR
   PRAY:ARRAY [1..6,0..MAXNUMBEROFNODES] OF INTEGER;
   NEWLVL,NEWNNODES,K,FLAG1,J,PRNUM:INTEGER;
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

PROCEDURE WDOT;

(* ************************************************************
*                                                              *
*  PROCEDURE WDOT                                              *
*  USE:  THIS PROCEDURE DRAWS A DOT AFTER VARIOUS FUNCTIONS OF *
*        PRUNING TO LET THE USER KNOW THAT SOMETHING IS        *
*        HAPPENING.                                            *
*  CALLED BY: SEGMENT PROCEDURE MODPRU (PRUNE)                 *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED:  (none)                                          *
*       MODIFIED: (none)                                       *
*                                                              *
************************************************************ *)

BEGIN
  WRITE(OUTPUT,'.')
END;
```

(* SEGMENT PROCEDURE MODPRU *)

```
PROCEDURE RELINK;

(* ****************************************************************
*                                                                 *
*  PROCEDURE RELINK                                               *
*  USE:  THIS PROCEDURE IS USED TO RELINK THE TREE STRUCTURE      *
*        AFTER DELETING THE BRANCH OF THE TREE DESIGNATED BY THE  *
*        USER.  THE ONLY LINKS THAT ARE RELINKED ARE THE          *
*        DOWNLINKS, CROSSLINKS, AND THE BACKLINKS.  THE NRN       *
*        DIGITS ARE RE-DESIGNATED IN THE MAIN BODY OF THE PRUNE   *
*        PROGRAM.                                                 *
*        CASES ADDRESSED:                                         *
*            1)  PRUNING THE DESCENDENTS ONLY OPTION IS           *
*                REQUESTED.  THE DOWN LINK(S) OF THE NODE         *
*                INPUTTED ARE REMOVED.                            *
*            2)  OPTION N IS EXECUTED. THE NODE INPUTTED IS THE   *
*                LAST NODE ON THE SPAN.                           *
*            3)  OPTION N IS EXECUTED. THE NODE INPUTTED IS THE   *
*                FIRST NODE ON THE SPAN.                          *
*            4)  OPTION N IS EXECUTED. THE NODE INPUTTED IS IN    *
*                THE MIDDLE OF THE SPAN.                          *
*  CALLED BY: SEGMENT PROCEDURE MODPRU (PRUNE)                    *
*  ROUTINES CALLED: (none)                                        *
*  VARIABLES:                                                     *
*       USED: CH (see UNIT DASSA)                                 *
*             PRNUM (see PROCEDURE PRUNE)                         *
*       MODIFIED: PRAY (see PROCEDURE PRUNE)                      *
*                                                                 *
***************************************************************** *)
BEGIN
      IF(CH='D')
         THEN PRAY[4,PRNUM]:=0

         ELSE
           IF ((PRAY[5,PRNUM]=0)AND(PRAY[3,PRNUM]>1))
             THEN
               PRAY[5,PRAY[6,PRNUM]]:=0
             ELSE
               IF(PRAY[3,PRNUM]=1)
                 THEN
                  BEGIN
                   PRAY[4,PRAY[6,PRNUM]]:=PRAY[5,PRNUM];
                   PRAY[6,PRAY[5,PRNUM]]:=PRAY[6,PRNUM]
                  END

                 ELSE
                  BEGIN
                   PRAY[6,PRAY[5,PRNUM]]:=PRAY[6,PRNUM];
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

                    PRAY[S,PRAY[4,PRNUM]]:=PRAY[S,PRNUM]
                  END;

END;      (* END RELINK *)
```

```
                                                    (* SEGMENT PROCEDURE MODPRU *)

PROCEDURE COMPRESS;

(* ************************************************************
*                                                              *
*  PROCEDURE COMPRESS                                          *
*  USE:  THIS PROCEDURE REASSIGNS NODE NUMBERS TO THOSE NODES  *
*        WHICH REMAIN IN THE TREE AFTER THE INPUTTED NODE AND  *
*        DESCENDENTS ARE ZEROED.  REASSIGNMENT RESULTS IN THE  *
*        PRUNED NODES BEING FREED TO TAKE ON NEW NODE          *
*        DEFINITIONS.  THIS COMPRESSION IS USED TO MAKE THE    *
*        MOST USE OF THE LIMITED NUMBER OF NODES (CURRENTLY 100)*
*        AVAILABLE IN THE PROGRAM.                             *
*  CALLED BY:  SEGMENT PROCEDURE MODPRU (PRUNE)                *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED:  NNODES   (see UNIT DASSA)                       *
*       MODIFIED: I (see UNIT DASSA)                           *
*                 PRAY,J,K,FLAG1,NEWNNODES (see PROCEDURE PRUNE)*
*                                                              *
************************************************************** *)

 BEGIN

      FOR I:=1 TO NNODES DO
       BEGIN
        IF(PRAY[2,I]=0)
         THEN
          BEGIN
           J:=I+1;
           FLAG1:=0;
           WHILE((J<=NNODES)AND(FLAG1=0)) DO
            IF(PRAY[2,J]=0)
             THEN J:=J+1
             ELSE
               FOR K:=2 TO 6 DO
                BEGIN
                 PRAY[K,I]:=PRAY[K,J];
                 PRAY[K,J]:=0;
                 FLAG1:=1
                END
          END
        END;


      (* RELABEL THE LINKS *)

       I:=0;
       FLAG1:=0;
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

        REPEAT
         BEGIN
         I:=I+1;
         IF(PRAY[2,I]=0) THEN FLAG1:=1
         END
        UNTIL(FLAG1=1);

        NEWNNODES:=I-1;

        FOR I:=1 TO NEWNNODES DO

         FOR J:=1 TO NEWNNODES DO
          FOR K:=4 TO 6 DO

           IF (PRAY[K,J]=PRAY[2,I])
             THEN PRAY[K,J]:=PRAY[1,I];
END;
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

PROCEDURE REWRITE;

(* ************************************************************
*                                                              *
*  PROCEDURE REWRITE                                           *
*  USE:  REWRITES THE PRUNED TREE DATA BACK TO THE DISK FILE.  *
*        CONSERVES INFORMATION THAT WAS PRESENT WITH THE NODE  *
*        PRIOR TO PRUNING.  RATIONALES ARE ALSO CONSERVED.     *
*        ALL NODES NOT IN THE NEW TREE STRUCTURE ARE ZEROED AS *
*        IN A NEW FILE.                                        *
*  CALLED BY: SEGMENT PROCEDURE MODPRU (PRUNE)                 *
*  ROUTINES CALLED:  (none)                                    *
*  VARIABLES:                                                  *
*       USED:  NNODES,MAXSYSTEMS (see UNIT DASSA)              *
*       MODIFIED: I,NODE[CELLNUMBER,NRNDIGIT,DOWNLINK,         *
*                 CROSSLINK,BACKLINK,NODETITLE,RELWEIGHT,      *
*                 CUMWEIGHT,SYSTEMVALUES,RATIONALE]            *
*                                 (see UNIT DASSA)             *
*                 NEWNNODES,J (see PROCEDURE PRUNE)            *
*                                                              *
************************************************************** *)
BEGIN
    FOR I:=1 TO NEWNNODES DO
     BEGIN
      SEEK(NODE,PRAY[2,I]);
      GET(NODE);
      WITH NODE^ DO
       BEGIN
        CELLNUMBER:=I;  NRNDIGIT:=PRAY[3,I];
        DOWNLINK:=PRAY[4,I];  CROSSLINK:=PRAY[5,I];  BACKLINK:=PRAY[6,I]
       END;
      SEEK(NODE,I);
      PUT(NODE)
     END;

    WITH NODE^ DO
     BEGIN
     NODETITLE:='BLANK      ';
     CELLNUMBER:=0;  NRNDIGIT:=0;  DOWNLINK:=0;
     CROSSLINK:=0;   BACKLINK:=0;  RELWEIGHT:=0.0;
     CUMWEIGHT:=0.0;

     FOR J:=1 TO MAXSYSTEMS DO SYSTEMVALUES[J]:=0;

     RATIONALE:='NO COMMENT'
     END;

    FOR J:=NEWNNODES+1 TO NNODES DO PUT(NODE);
```

```
                                                (* SEGMENT PROCEDURE MODPRU *)

END;  (* END REWRITE *)
```

```
                                                (* SEGMENT PROCEDURE MODPRU *)

(* ************************************************
*                                                  *
*       BEGIN   PROCEDURE   PRUNE                  *
*                                                  *
************************************************** *)

BEGIN

   REPEAT
     BEGIN
      PAGE(OUTPUT);
      WRITE(OUTPUT,'N(ODE+DOWN  D(OWN ONLY  E(XIT ');
      REPEAT
        UNTIL KEYPRESS;
      READ(INPUT,CH)
     END
   UNTIL ((CH='N')OR(CH='D')OR(CH='E'));

   IF(CH()'E')
     THEN
      BEGIN

       FOR I:=1 TO NNODES DO
        BEGIN
         SEEK(NODE,I);
         GET(NODE);
         WITH NODE^ DO
          BEGIN
           PRAY[1,I]:=I;
           PRAY[2,I]:=CELLNUMBER;
           PRAY[3,I]:=NRNDIGIT;
           PRAY[4,I]:=DOWNLINK;
           PRAY[5,I]:=CROSSLINK;
           PRAY[6,I]:=BACKLINK;
          END
        END;


       WRITELN(OUTPUT);
       PRENEX;


       IF((IRAY[LVL,2]=0)AND(CH='D'))
         THEN
          BEGIN
           WRITELN(OUTPUT,'YOU CANNOT PRUNE A DATA NODE WITH');
           WRITELN(OUTPUT,'A DOWN ONLY OPTION. . . ');
           WRITELN(OUTPUT,'PRUNE TERMINATED . . .');
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

      WRITELN(OUTPUT,'(ANY KEY) CONTINUE');
      REPEAT
       UNTIL KEYPRESS;
      READ(INPUT,CH);
      CH:='E'
     END;

IF(CH()'E')
  THEN
   BEGIN


  WRITE(OUTPUT,'PRUNING.');
  PRNUM:=IRAY[LVL,0];


  RELINK;
  WDOT;


  IF(CH='N') THEN PRAY[2,PRNUM]:=0;
  NEXT;
  WDOT;
  WHILE (ICONT()0) DO
   BEGIN
    PRAY[2,IRAY[LVL,0]]:=0;
    NEXT
   END;


  WDOT;

  (* NRN ADJUSTMENT *)

  IF(CH='N')
  THEN
  BEGIN
  I:=PRAY[5,PRNUM];
  WHILE(I()0) DO
   BEGIN
    PRAY[3,I]:=PRAY[3,I]-1;
    I:=PRAY[5,I]
   END;
  END;

  WDOT;

  COMPRESS;
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)


      WDOT;
      REWRITE;
      WDOT;
    NNODES:=NEWNNODES

  END;

 END;

 NEWLVL:=1;
 PRETOT;
 REPEAT
  BEGIN
  IF(LVL)NEWLVL)THEN NEWLVL:=LVL;
  NEXT
  END
 UNTIL(ICONT=0);
 NDEEP:=NEWLVL;
 WDOT;
 FOR I:=1 TO MAXARRAYSIZE DO IRAY[I,0]:=0;
 LVL:=1;
 SEEK(NODE,1);
 GET(NODE);
 NODEDISKTOARRAY(LVL);

 FLAG:=1


END;  (* END OF PRUNE *)
```

```
                                        (* SEGMENT PROCEDURE MODPRU *)

PROCEDURE MODIFY;

(* ************************************************************** *
*                                                                  *
*  PROCEDURE MODIFY                                                *
*  USE:  PROCEDURE SUPPORTS OPTION MOD.  ELICITS NODE REFERENCE    *
*        NUMBER FROM THE USER.  IF THE NODE EXISTS, ONLY THE       *
*        LABEL OF THE NODE IS CHANGED.  IF THE NODE DOES NOT       *
*        EXIST, A NEW BRANCH IS CREATED TO THE NODE.  IF NO        *
*        SPANS EXIST TO THE INPUTTED NODE, A SINGLE NODE SPAN      *
*        IS CREATED AND HAS A NRN NUMBER OF ONE ON THE LEVEL       *
*        REGARDLESS OF WHAT WAS INPUTTED (THIS IS PRIMARILY        *
*        TO INSURE AGAINST PROGRAM FAILURE IN SUBSEQUENT           *
*        PRUNING OPERATIONS).                                      *
*  CALLED BY: SEGMENT PROCEDURE MODPRU                             *
*  ROUTINES CALLED: UNIT DASSA (NODEARRAYTODISK,NODIN)             *
*  VARIABLES:                                                      *
*       USED: NLVLS,TENSTRING,ICONT (see UNIT DASSA)               *
*       MODIFIED:  IRAY,LVL,NDEEP,L,IFADD,IFIND,NNODES,            *
*                  ANSWER,NODELABEL,ARAY,VRAY (see UNIT DASSA)     *
*                  IQUIT (see PROCEDURE MODIFY)                    *
*                                                                  *
************************************************************** *)

VAR
   IQUIT:INTEGER;

 BEGIN
   IQUIT:=0;
   REPEAT
     BEGIN
     NODIN;
     IF(NLVLS(1)
       THEN IQUIT:=1
       ELSE
       BEGIN
        WRITE(OUTPUT,'LABEL? ');
        READLN(INPUT,ANSWER);
        IF((ANSWER='')OR(ANSWER='DONE'))
         THEN IQUIT:=1
         ELSE
         BEGIN
           ANSWER:=COPY(CONCAT(ANSWER,TENSTRING),1,10);
           IF(ICONT=1)
            THEN
            BEGIN
             NODELABEL[LVL]:=ANSWER;
             SEEK(NODE,IRAY[LVL,0]);
```

```
                                                (* SEGMENT PROCEDURE MODPRU *)

                NODEARRAYTODISK(LVL);
                PUT(NODE)
               END

               ELSE
               BEGIN
                L:=3-IFADD;
                IFIND:=IRAY[LVL,0];
                REPEAT
                 BEGIN
                 NNODES:=NNODES+1;
                 IRAY[LVL,IFADD]:=NNODES;
                 SEEK(NODE,IRAY[LVL,0]);
                 NODEARRAYTODISK(LVL);
                 PUT(NODE);
                 LVL:=LVL+L;
                 IRAY[LVL,0]:=NNODES;

                 IF(L=0) THEN IRAY[LVL,1]:=IRAY[LVL,1]+1
                         ELSE IRAY[LVL,1]:=1;

                 IRAY[LVL,2]:=0;
                 IRAY[LVL,3]:=0;
                 IRAY[LVL,4]:=IFIND;
                 IRAY[LVL,5]:=IFIND;
                 FOR I:=1 TO 2 DO ARAY[LVL,I]:=0;
                 FOR I:=1 TO NSYS DO VRAY[LVL,I]:=0.0;
                 NODELABEL[LVL]:=ANSWER;
                 SEEK(NODE,IRAY[LVL,0]);
                 NODEARRAYTODISK(LVL);
                 PUT(NODE);
                 IFADD:=2;
                 IFIND:=NNODES;
                 L:=1
                 END
                UNTIL(LVL)=NLVLS)
              END
            END
          END
        END
      UNTIL(IQUIT)0);
      IF(LVL)NDEEP) THEN NDEEP:=LVL;
    END;  (* END MODIFY *)
```

```
                                                    (* SEGMENT PROCEDURE MODPRU *)

(* ************************************************
*                                                  *
*     BEGIN  SEGMENT  PROCEDURE  MODPRU            *
*                                                  *
************************************************** *)

BEGIN
 IF(CMD='PRU') THEN PRUNE
 ELSE IF(CMD='MOD') THEN MODIFY
END;
```

```
                                                    (* PROGRAM DASS *)

PROCEDURE STAT;

(* ************************************************************
*                                                              *
*  PROCEDURE STAT                                              *
*  USE:  PRESENTS INFORMATION ON THE CONSOLE CONCERNING THE    *
*        NUMBER OF NODES IN THE SYSTEM, THE NUMBER OF LEVELS   *
*        IN THE SYSTEM, AND THE NUMBER OF ALTERNATIVES IN THE  *
*        SYSTEM.                                               *
*  CALLED BY: PROGRAM DASS                                     *
*  ROUTINES CALLED: (none)                                     *
*  VARIABLES:                                                  *
*       USED: NNODES,NDEEP,NSYS (see UNIT DASSA)               *
*       MODIFIED: CH (see UNIT DASSA)                          *
*                                                              *
************************************************************ *)

  BEGIN
   PAGE(OUTPUT);
   WRITELN(OUTPUT,'NUMBER OF NODES ',NNODES);
   WRITELN(OUTPUT,'NUMBER OF LEVELS ',NDEEP);
   WRITELN(OUTPUT,'NUMBER OF SYSTEMS ',NSYS);
   WRITELN(OUTPUT);
   WRITELN(OUTPUT,'(ANY KEY) CONTINUE');
   REPEAT
    UNTIL KEYPRESS;
   READ(CH)
  END;
```

```
                                                        (* PROGRAM DASS *)
(* ************************************************************
*                                                              *
*            BEGIN    PROGRAM    DASS                         *
*                                                              *
************************************************************** *)
BEGIN
  COMMENTSTRING:=CONCAT(TENSTRING,TENSTRING,TENSTRING,
                  TENSTRING,TENSTRING,TENSTRING);
  LABELSTRING:=TENSTRING;
  FLAG:=0; NSYS:=2; NFLAG:=0; NDEEP:=0;
  IRAY(1,3):=2;  CMD:='SEL';DUMMY;

  REPEAT
   BEGIN
   PAGE (OUTPUT);
   WRITELN(OUTPUT,'ATT DIS DON MOD NEW NUM PRU REV SEL ');
   WRITELN(OUTPUT,'SEN SPA STA SYS TTL WVC');
   WRITE(OUTPUT,'OPTION? ');
   READLN(INPUT,CMD);
   IF(CMD='ATT') THEN DUMMY
   ELSE IF(CMD='DIS') THEN NUM
   ELSE IF(CMD='MOD') THEN MODPRU
   ELSE IF(CMD='NEW') THEN DUMMY
   ELSE IF(CMD='NUM') THEN NUM
   ELSE IF(CMD='PRU') THEN MODPRU
   ELSE IF(CMD='REV') THEN NUM
   ELSE IF(CMD='SEL') THEN DUMMY
   ELSE IF(CMD='SEN') THEN SENSITIVITY
   ELSE IF(CMD='SPA') THEN DUMMY
   ELSE IF(CMD='STA') THEN STAT
   ELSE IF(CMD='SYS') THEN READSYSTEMLABELS
   ELSE IF(CMD='TTL') THEN DUMMY
   ELSE IF(CMD='WVC') THEN WVLOAD
   END
  UNTIL(CMD='DON');
  SEEK(NODE,0);
  GET(NODE);
  WITH NODE^ DO
    BEGIN
     RATIONALE:=TITLE;     NOUETITLE:=LATT;
     CELLNUMBER:=NNODES;   NRNDIGIT:=NDEEP;
     DOWNLINK:=NSYS;       CROSSLINK:=FLAG
    END;
  SEEK(NODE,0);          PUT(NODE);
  CLOSE(NODE, LOCK);    CLOSE(SYSTEMS, LOCK);
END.    (* END OF PROGRAM DASS *)
```

VITA

David B. Lee was born and raised in Chicago, Illinois. He received a Bachelor of Science degree from the Illinois Institute of Technology in Chemical Engineering in June 1972. He received his Air Force commission through ROTC, and served from September 1972 to May 1976 in the MINUTEMAN Missile Combat Crew Force at Whiteman Air Force Base, Missouri. During this time, he received his Masters in Business Administration from the University of Missouri. He then served from May 1976 until his entry into the School of Engineering, Air Force Institute of Technology as a project manager and section chief at the Air Force Rocket Propulsion Laboratory, Edwards Air Force Base, California.

END

FILMED

1-84

DTIC