MASTER

R-4355

A Guide to IDAP, Version 2: An Interactive Decision Analysis Procedure

M. J. Jusko and R. G. Whitfield

ARGONNE NATIONAL LABORATORY
Energy and Environmental Systems Division

prepared for
U. S. DEPARTMENT OF ENERGY
under Contract W-31-109-Eng-38

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) among the U. S. Department of Energy, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona
Carnegie-Mellon University
Case Western Reserve University
The University of Chicago
University of Cincinnati
Illinois Institute of Technology
University of Illinois
Indiana University
The University of Iowa
Iowa State University

The University of Kansas
Kansas State University
Loyola University of Chicago
Marquette University
The University of Michigan
Michigan State University
University of Minnesota
University of Missouri
Northwestern University
University of Notre Dame

The Ohio State University
Ohio University
The Pennsylvania State University
Purdue University
Saint Louis University
Southern Illinois University
The University of Texas at Austin
Washington University
Wayne State University
The University of Wisconsin-Madison

NOTICE

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government or any agency thereof, nor any of their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

This informal report presents preliminary results of ongoing work or work that is more limited in scope and depth than that described in formal reports issued by the Energy and Environmental Systems Division.

Printed in the United States of America. Available from National Technical Information Service,
U. S. Department of Commerce, 5285 Port Royal Road, Springfield, Virginia 22161

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

ANL/EES-TM-126

A GUIDE TO IDAP, VERSION 2:
AN INTERACTIVE DECISION ANALYSIS PROCEDURE

by

Mark J. Jusko and Ronald G. Whitfield

Energy and Environmental Systems Division
Integrated Assessments and Policy Evaluation Group

November 1980

DISCLAIMER

This book was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

work sponsored by

U.S. DEPARTMENT OF ENERGY

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MGW

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

CONTENTS

EXECUTIVE SUMMARY.	vii
1 INTRODUCTION.	1
1.1 Background	1
1.2 Developing the IDAP	1
1.3 Use of this Document	2
2 SUMMARY OF DECISION ANALYSIS.	4
2.1 Preamalysis.	4
2.2 Structural Analysis.	5
2.3 Uncertainty Analysis	5
2.4 Utility Analysis	5
2.5 Optimization Analysis	6
3 CAPABILITIES OF THE IDAP.	7
3.1 Tree Structures	7
3.2 Calculating Utility Values.	9
3.2.1 Multiattribute Utility Functions.	11
3.2.1.1 The Additive MUF	11
3.2.1.2 The Multiplicative MUF	12
3.2.2 Marginal Functions.	12
4 THE IDAP - A USER'S GUIDE	17
4.1 List of Commands	17
4.2 RAWDAT Input File Format	22
4.3 Node Labeling.	26
4.4 Example 1.	27
4.4.1 Input	27
4.4.2 Annotated IDAIN Procedure for Example 1	31
4.4.3 Input: JCL and RAWDAT	38
4.4.4 Output.	40
4.5 Example 2.	53
4.5.1 Input: JCL and RAWDAT	53
4.5.2 Output.	55
4.6 Example 3.	60
4.6.1 Input: JCL and RAWDAT	60
4.6.2 Output.	62
5 THE IDAP - A PROGRAMMER'S GUIDE	67
5.1 General Background	67
5.2 IDA's Procedures	70
5.2.1 Main Procedure	70
5.2.2 ALTER	83
5.2.3 BUILD	88

CONTENTS (Cont'd)

5.2.4	CALCK.	91
5.2.5	LISTM.	98
5.2.6	LISTU.	101
5.2.7	LOCATE	103
5.2.8	LODOWN	105
5.2.9	LONODE	108
5.2.10	LOPOINT.	110
5.2.11	PEEL	113
5.2.12	POINT.	116
5.2.13	PRINT.	128
5.2.14	PROCESS.	131
5.2.15	READRAW.	146
5.2.16	STRIP.	158
5.2.17	SUBTOL	159
5.2.18	UTILOF	163
5.3	System Requirements	169
5.4	JCL at Argonne.	171
5.5	The IDAIN Procedure	172
6	IDA'S ERROR MESSAGES	181
	REFERENCES.	185
	ACKNOWLEDGMENTS	187
	APPENDIX 1 - RECENT CHANGES IN THE IDAP	189

FIGURES

1	A Decision Tree Representation	viii
1.1	The Creation of the IDAP	3
3.1	Sample Decision Tree	8
3.2	IDA's Marginal Functions	13
4.1	Example 1 Decision Tree.	28
4.2	Example 1 Marginal Functions	30
5.1	IDAP Procedure Flow.	67
5.2	The MAIN Procedure	71
5.3	The GETCOM Module.	72
5.4	The LIST Module.	72
5.5	The LISTMAR Module	73
5.6	The LISTUTI Module.. . . .	73
5.7	The PREPRO Module.	74

FIGURES (Cont'd)

5.8	The ERROR Procedure	74
5.9	The PAR@SET Module	75
5.10	The BUILD Procedure	89
5.11	Calculating K for a Multiplicative MUF	92
5.12	The CALCK Procedure	93
5.13	The FIND Module	94
5.14	The OPPOSE Module	95
5.15	The Y Module	95
5.16	The LISTM Procedure	99
5.17	The LISTU Procedure	101
5.18	The LOCATE Procedure	103
5.19	The LODOWN Procedure	108
5.20	The LONODE Procedure	111
5.21	The LOPOINT Procedure	114
5.22	The PEEL Procedure	117
5.23	The POINT Procedure	118
5.24	The DAD@LEN Module	119
5.25	The R@SIBS Module	129
5.26	The PRINT Procedure	132
5.27	The PROCESS Procedure	132
5.28	Part A of the COUNT Module	133
5.29	Part B of the COUNT Module	133
5.30	Part A of the SUMCOST Module	134
5.31	Part B of the SUMCOST Module	134
5.32	The UPSTART Module	135
5.33	The P@ONLY Module	136
5.34	Part A of the UPMORE Module	137
5.35	Part B of the UPMORE Module	137
5.36	The PRESENT Module	137
5.37	The READRAW Procedure	147
5.38	The STRT@ND Module	147
5.39	The MARGINAL Module	148
5.40	The GETGEN Module	148
5.41	The GETUTIL Module	149

FIGURES (Cont'd)

5.42	The STRIP Procedure	158
5.43	The SUBTOL Procedure	160
5.44	The L@SUBT Module	160
5.45	The UTILOF Procedure	164
5.46	The CALC@M Module	165

TABLES

4.1	Nodes and Parameters of Sample Decision Problems	29
6.1	IDAP's Error Messages.	181

EXECUTIVE SUMMARY

This is a description of the interactive decision analysis computer program (IDAP). IDAP is a tool for creating, manipulating, and evaluating decision trees, which are representations of decision problems. Decision analysis, a branch of decision theory, provides the theoretical foundation.

Decision analysis is an approach to solving decision problems characterized by many alternatives, complex tradeoffs among conflicting attributes, and uncertainty.

The decision analysis paradigm may be summarized by the following five steps:

1. Preanalysis - the decision maker identifies a problem and defines a set of alternatives.
2. Structural analysis - the structural details of the problem are assembled.
3. Uncertainty analysis - probabilities for chance outcomes are quantified.
4. Utility analysis - value tradeoffs among outcome attributes are quantified using the decision maker's preferences.
5. Optimization analysis - after structuring the problem, assigning probabilities, and assigning utilities, the optimal strategy is calculated on the basis of expected utility.

Decision problems may be depicted with a decision tree, an example of which is illustrated in Fig. 1. The figure depicts decision nodes (e.g., the A_1 , A_2 fork representing a choice between alternatives A_1 and A_2 , and probability nodes (such as the one leading to outcomes O_1 , O_2 , and O_3 with probabilities p_1 , p_2 , and p_3 , respectively, with $p_1 + p_2 + p_3 = 1$). Outcomes are defined in terms of the three attributes X_1 , X_2 , and X_3 . Amounts of attributes are represented by lower case letters with subscripts identifying attribute/outcome pairs.

Uncertainty is represented by probability distributions at probability nodes. Value tradeoffs (or preferences) among the attributes are represented with utility functions.

Multiattribute utility functions are functions of single-attribute utility functions, sometimes called marginal functions. Marginal utility functions represent the decision maker's preferences for various amounts of an attribute under conditions of uncertainty. The marginal utility functions are combined into a multiattribute utility function, consistent with the decision maker's preferences.

The IDAP deals mainly with the last step of the decision analysis paradigm - optimization analysis. The data from problem structuring, uncertainty analysis, and utility assessment are managed using this program. Capabilities of IDAP include:

Fig. 1 A Decision Tree Representation of a Multiattribute Decision Problem with Uncertainty

- Handling multiple decision nodes, including sequential decisions,
- Handling discrete probability nodes with the number of branches virtually unlimited, so that a continuous probability function can be closely approximated,
- Handling many attributes,
- Accepting a wide variety of functional forms for marginal utility functions, including both discrete and continuous values,
- Treating multiattribute utility functions of either the additive or multiplicative forms,
- Calculating expected utility for each decision alternative,
- Identifying the optimal choice (that which maximizes expected utility), and
- Performing sensitivity analyses (that is, the probabilities associated with chance nodes, attribute level tolls, and scaling factors may be changed).

In addition to a listing of the computer code and a description of the program, this document provides an overview of decision analysis and explains the usefulness of the IDAP in performing a decision analysis. Examples are used to illustrate these points.

Appendix 1 presents some details of the IDAP as it exists on an interactive system. This section will be updated to reflect any changes in that version.

1 INTRODUCTION

1.1 BACKGROUND

A number of efforts in the Energy and Environmental Systems Division at Argonne National Laboratory (ANL) involve decision analysis. We realized that a computer program to facilitate such efforts would be useful, and, as a result, IDAP* was conceived. With the aid of this procedure the user is freed from many of the tedious calculations that must be performed when selecting the best path in a decision tree; the IDAP selects the "best choice" of options available, on the basis of well-defined calculations performed on the user's input.

There are two independent procedures that together define the IDAP. One is a small WYLBUR exec procedure**, IDAIN, that allows the user to interactively enter input for the main program. The other is the main IDAP program, written in PL/1, that operates on the user's input previously formatted by IDAIN.

Work on the IDAP procedure was performed at ANL, which has a number of powerful computing systems available to users. The development of the IDAP is an ongoing process, and the procedure as presented in this document should not be viewed as a final version. A version of the IDAP is now available to users at ANL.

1.2 DEVELOPING THE IDAP

The IDAP is undergoing almost continuous changes in its design, primarily because users make suggestions as they become familiar with the procedure. During the initial work on the IDAP only a few individuals made such suggestions which, however, generated major design specifications. Later, as the IDAP became more refined, we realized that suggestions should be sought from a larger group of individuals. As a result, the latest working version of the IDAP is herein documented and is distributed with the hope that as users become more familiar with the program, more suggestions will be submitted.

The first version of IDAP was not truly interactive, since it was implemented for batch mode only. However, it served to define the major fundamental procedures. Version 2 is presently functional; it works on both batch and interactive systems, and includes a crude procedure for sensitivity analysis. Version 3 is currently being developed; it will have a refined sensitivity analysis, and may include computer graphics (e.g., to draw decision trees). A fourth version may be created, depending on suggestions

*IDAP is an acronym for Interactive Decision Analysis Procedure. At the time of this writing, the IDAP is being implemented on CMS (an interactive system at ANL), and although this document is intended for batch users, the general theory holds for CMS users as well. Appendix 1 describes the IDAP on CMS. The current CMS version of IDAP is available from the National Energy Software Center.

**WYLBUR is an interactive text editor with limited programming capabilities.

and comments from users of Versions 2 and 3. This report documents the functions and use of Version 2. Figure 1.1 shows a flow chart of the ongoing creation of the IDAP.

1.3 USE OF THIS DOCUMENT

The function of this document is twofold. It is intended

1. To serve as both a programmer's and user's guide to the current version of the IDAP; and
2. To prompt interested individuals into making suggestions for the future development of IDAP.

The majority of the sections of this document pertain to the main IDA program rather than to the IDAIN procedure.

Section 2 of this document presents a brief discussion of the theory of decision analysis. The aspects of decision analysis that are relevant to the IDAP are discussed in Sec. 3. Section 4 provides a complete list and description of the commands used in the IDAP program and includes three complete examples. This section may be considered a user's guide to the IDAP. Section 5 may be considered a programmer's guide to the IDAP; it discusses the various technical aspects of the programs, and may be skipped by users not involved with programming the IDAP. Section 6 presents a list of the error messages generated by the IDAP. As the program is developed, error handling and messages will improve. Appendix 1 discusses recent changes made in the IDAP, and its implementation on CMS at ANL. Future versions of the IDAP will be documented and the appendix modified when appropriate.

Figure 1.1 The Creation of the IDAP

2 SUMMARY OF DECISION ANALYSIS

Decision analysis is a systematic approach that may be used to search for an optimal solution to a problem. Decision making in the energy/environmental sector is often complicated by conflicting objectives or uncertain events, and these features of the problem are handled formally by decision analysis. Conflicting objectives, such as plant design vs. plant safety, are dealt with through the use of mathematical utility functions, while uncertainties are handled by probability distributions. The theory of decision analysis has a firm mathematical basis and is thoroughly documented in available literature (see References).

Keeney¹ describes a five-step paradigm for applying decision analysis to a specific problem. These steps are outlined below and described in more detail later in this section.

1. Preanalysis: define the problem and establish viable alternatives to be examined.
2. Structural analysis: once the problem has been defined, its qualitative aspects should be formally structured into a decision tree. Such a tree, as shown in Fig. 3.1, should define the structure of the decision maker's problem and should allow one to visualize any possible outcome as a single path in that tree.
3. Uncertainty analysis: probabilities are assigned to probability nodes in a decision tree, that is, to those nodes initiating a set of outcomes that are determined by chance.
4. Utility analysis: assign utility values to the consequences associated with the outcomes represented by the various paths of the decision tree.
5. Optimization analysis: once a problem has been structured, and probabilities and utility values assigned within the tree, a strategy that maximizes the overall expected utility of the problem can be calculated.

It is principally with this last step, the optimization analysis, that IDAP is concerned. The remainder of this section briefly describes each of the above steps; the next section expands the concepts of decision analysis that apply to the IDAP.

2.1 PREANALYSIS

In general, decisions can be described as the choice between two or more alternatives. Each alternative can in turn be a choice among other alternatives, and so on. Frequently, however, an alternative is a point from which multiple potential outcomes may arise. It is chance that determines exactly which of these alternatives will occur, according to a particular probability distribution.

When a decision problem is being defined, it is vital that the decision maker have at least a rough idea of viable options, probabilities associated with chance outcomes, and the values of each outcome. It is not always necessary, however, to have a complete knowledge of outcome values or probability distributions; if estimates are available, sensitivity analyses can be performed to evaluate the final decision and determine the significance of an optimal strategy.

In defining a problem, it may be useful for the decision maker to keep in mind the structural requirements of decision analysis, as explained briefly in the next section.

2.2 STRUCTURAL ANALYSIS

A decision tree is an information structure that is essential to decision analysis algorithms. Such a tree is composed of decision nodes (represented by squares) and chance nodes (represented by circles). The goal of a decision analysis is to obtain an optimal strategy, a rule for selecting the best path through the entire tree beginning at its root.

While following a path through a tree, the decision maker may encounter a decision node or a chance node. If a decision node is encountered, the choice of which path to follow among those branching from that node is under complete control by the decision maker: there is a decision to be made. If the node encountered is a chance node, the decision maker has no control over the path that will be selected; it is chance that selects one of these paths, based on their probabilities of occurrence.

When structuring a problem to conform to a decision tree, the decision maker should examine the problem in a sequential manner, placing the decisions that must be made immediately prior to those decisions that may be deferred, or before nodes whose paths are selected by chance. Frequently a decision may be whether or not to experiment, or whether or not to purchase information. Similarly, information that may be required further down a tree is often obtained during the normal course of traversing the tree.

2.3 UNCERTAINTY ANALYSIS

Once a tree is structured after a decision problem, probabilities must be assigned to paths leaving a chance node. This is frequently a difficult process, and may involve input from a number of sources, such as empirical data, expertise from individuals familiar with the decision problem (taking into account personal biases and viewpoints, as in Walsh²), and subjective and objective evaluation by the decision maker.

2.4 UTILITY ANALYSIS

After a decision problem has been structured into a tree, the decision maker must assign utility values to all paths within the tree. Multiple objectives in a decision problem are captured by the use of multiple attributes. Each attribute, whether psychological, medical, environmental, or

mechanical, represents an aspect of the problem being analyzed. Each path in a decision tree has associated with it a specific value or level for each attribute under consideration. The attribute levels are based on a normalized scaling system.

It is up to the the decision maker to determine these attribute levels, and to mathematically combine them into a multiattribute utility function. Generating attribute levels and utility values is frequently difficult, as seen in Keeney.³ Such values are necessarily subjective, and although the techniques for obtaining such values are well defined, their application approaches an art.

2.5 OPTIMIZATION ANALYSIS

Once a decision tree has been completely structured to represent a decision problem, probabilities have been assigned to chance nodes, and utility values assigned to all paths in the tree, the decision maker can search for an optimal strategy. Such a strategy will instruct the decision maker as to which alternatives to select, and will eventually yield the maximum expected utility for the problem in question. The averaging-out-and-folding-back algorithm described in Raiffa⁴ and other decision analysis literature is commonly used to obtain this maximum expected utility, and it is the basis of the IDAP computer program.

Sensitivity analysis can often be performed once a strategy is suspected to be optimal. If the strategy remains optimal over wide ranges of probabilities for chance nodes, and over wide ranges of utility values for decision tree paths, the decision maker may be reassured that a good strategy has been selected. If the various available strategies are phased into and out of optimality when probabilities or utility values are changed, it may be advisable for the decision maker to review the basic assumptions used in solving the decision problem, or to narrow down the ranges of possible values of these probabilities or utility values. In many energy/environmental decision problems, strategies are narrowly divided, and an optimal strategy must be selected with care.

3 CAPABILITIES OF THE IDAP

Decision analysis is a complex topic - too complex to be discussed here in detail.* However, this section will develop the concepts of decision analysis introduced in Sec. 2 that form the basis of IDAP. Specifically, the various aspects of decision trees and the functions that operate on them will be discussed.

3.1 TREE STRUCTURES

The problems to which the procedures of decision analysis may be applied are often complex**. However, in applying these procedures, individuals are forced into precisely defining the problem to be examined. The IDAP user must organize the problem into components of well-defined options and uncontrollable probabilistic events, and define the relationships and probability distributions of these components. In addition, the user must formulate several functions that are assumed to define totally the preferences to be used in solving the problem. When these data are input to IDAP in the form of a decision tree, utility values are calculated for each of the various components, and the user is at last presented with a strategy that optimizes the expected utility of a solution to the problem.

One type of information structure is a tree structure. As the name suggests, there are branches that may be traversed in getting from the trunk (starting node) to the leaves (terminal nodes). Each leaf uniquely determines a path from the trunk of the tree to that leaf, as seen in Fig. 3.1.

Decisions, often viewed as the choice of one of two or more branches along a path, are often conveniently broken down into such tree structures if they are complex. Likewise, a tree structure is well suited for representing events that are not options, but have a probability distribution associated with them. A tree that is used to model such a series of decisions and events is called a decision tree.

The concept of a decision tree is important; it is used in the IDAP procedure and is discussed in this document. Figure 3.1 illustrates such a tree, which contains a number of uniquely labeled nodes. A number of paths or branches link these nodes together; each path leads to a terminal node. There is also a starting node of the tree, at the far left side of the figure, from which all paths originate.

There are three different types of nodes in such a tree: an option node, a probability node, and a terminal node. Option nodes are represented by squares, probability nodes by circles, and terminal nodes by triangles. The labeling of nodes is important to IDAP; each node must have a unique label.

*Raiffa⁴ presents the general theory, and a few examples, of decision analysis.

**Buehring⁵ presents one such problem and its solution by decision analysis methods.

"p=n" denotes the probability n of the son's occurrence.

"(X1,X2,X3,...,Xn)" denotes the levels of the various attributes collected along the path from the root to that leaf.

Fig. 3.1 Sample Decision Tree

consisting of a string of integers separated by periods.* These labels, supplied by the user, are critical when IDAP is constructing the tree.

A probability is associated with the sons (that is, possible outcomes or consequence states, represented by nodes) of each probability node. This numeric value (necessarily between 0 and 1) represents the probability that the event represented by the son will occur, given that its father occurs. Since it is impossible to be certain, in many cases, exactly which events may occur when a particular path is taken, it is useful to combine all the outcomes and their probabilities mathematically when evaluating the utility of that path.

Between the nodes of the tree are various familial relationships. Node 1 is the father of nodes 1.1 and 1.2, which are sons of node 1, and these sons are in turn fathers of nodes 1.1.1 and 1.1.2, and of 1.2.1 and 1.2.2, respectively. Node 1.1.1 is the first son (IDAP uses the term #1SON) of node 1.1. The labeling notation adopted for the IDAP efficiently reflects many of these familial relationships. The node labeling convention used in IDAP must be followed carefully by the user. Unpredictable results may occur if this convention is violated.

A subtree is itself a tree structure. The subtree of node 2.2 in Fig. 3.1 contains nodes 2.2, 2.2.1, and 2.2.2, that is, all nodes whose labels begin with the integer "2."

In Fig. 3.1 there are three levels of nodes. All nodes belonging to the same level have the same number of forefathers. The notation used by IDAP for each node conveniently indicates the level of that node by the number of integers in its label.

In working with trees, IDAP utilizes pointers set up during the input process, when definitions of the relationships between nodes are being established. The starting node of the tree may be interpreted as the current position or initial condition of the user, whose problem is to decide which path to choose (which option to take). In order to make the best choice, the user must evaluate all paths available; that is where IDAP comes in. IDAP evaluates all paths defined by the user, and indicates which is the best.

3.2 CALCULATING UTILITY VALUES

Multiattribute utility values (MAUV) are used to represent numerically the relative values of the given levels of attributes defined in the problem. A MAUV is calculated using a multiattribute utility function (MUF) that operates on a number of values calculated from various marginal functions. Marginal functions are also utility functions, but, unlike MUFs, they operate on levels of a single attribute.

Attributes may be interpreted as the units of the costs the user may pay. For example, "dollars" could be one attribute, "ppm of sulfides"

*The nomenclature used in Chinnis⁶ was adopted in IDAP.

another, "births per year" another, and so on. The values or costs of these attributes, which may be normalized, are called the levels of those attributes.

To find the best path, the path that provides the best utility (MAUV) to the user, IDAP must do the following three things:

1. The "tolls" (costs accumulated along paths) must be algebraically summed. This is done by traversing each path from the starting node to the terminal node of that path and adding the tolls and attribute levels for each node encountered. Traverses of trees are done by following pointers along the desired paths. This entire process is frequently called "averaging out."
2. Once all values are accumulated at the terminal nodes of the tree, IDAP calculates the utility value of each of the leaves.
3. Once all terminal nodes contain utility values (between 0 and 1), IDAP traverses each path backward from a terminal node to the first level of the "decision," assigning utility values to nodes along each path. During this "folding back" process, several factors must be taken into consideration when assigning utility values from descendant to ancestor nodes.
 - a. If a father has a number of optional sons, that is, if the father is an option node, then it is assigned the greatest utility value of any of its sons.
 - b. A probability node, that is, a node whose sons are not options but probabilistic "events," is assigned a certain percentage of all the utility values of its sons. For each son, that percentage or weight is equal to its associated probability.

Multiattribute utility functions, which operate on marginal functions that in turn operate on levels of various attributes for each node, produce values that are the basis for the selection of the best path. The detailed theory behind utility and marginal functions is too complex for this document, but the functions utilized by IDAP will be discussed.*

There are two types of MUF used by IDA: the additive form and the multiplicative form.

These functions operate on other functions called marginal functions, of which IDAP recognizes the following:

- General,
- Linear (desirable),

*The theory behind these functions is presented in Keeney⁷.

- Linear (undesirable),
- Exponential (desirable),
- Exponential (undesirable),
- Gamma, and
- Quadratic forms.

These marginal functions operate on the levels of attributes associated with each node, as specified by the user. The levels of those attributes may, in many cases, be interpreted as costs, either positive or negative. As a result of these functions operating on attribute levels, the final utility value of a node will be a number representing the costs calculated from all the attributes associated with the nodes within its subtree.

The next two sections describe the MUF and marginal functions available to the user through IDAP. Note that these sections are not intended as directions for coding relevant input to IDAP, but they are provided as a general background on these functions.

3.2.1 Multiattribute Utility Functions

Multiattribute utility functions (MUF) are used by IDAP to combine various marginal values. The user must determine which of the two types of functions (additive or multiplicative) will be used, and this preference must be entered with other input to the program.

3.2.1.1 The Additive MUF

The additive form of the MUF is expressed by the following equation.

$$U = U(X_1, X_2, X_3, \dots, X_n) = \sum_{i=1}^n k_i U_i(X_i) \quad (1)$$

where U , a function of the levels of all n attributes, is the MUF as pre-defined by the user. Each of the n weighting factors k_i must be present in the user's input to IDAP. U_i represents the marginal function for attribute i ; X_i denotes an amount of attribute i . In all cases U and each of the U_i will produce values between 0 and 1. There are seven types of marginal functions available to the user in the IDA procedure; they are described later.

In order for IDAP to be able to calculate U correctly, the user must supply, in the input to IDAP, the name of the MUF to be used and the weighting factors.

Associated with the additive form of the MUF used by IDAP is the following relationship between the k_i :

$$\sum_{i=1}^n k_i = 1 \quad (2)$$

3.2.1.2 The Multiplicative MUF

The multiplicative form of the utility function used by IDAP is shown in the following equation.

$$U = U(X_1, X_2, X_3, \dots, X_n) = \frac{1}{K} \left\{ \prod_{i=1}^n [1 + K k_i U_i(X_i)] - 1 \right\} \quad (3)$$

where U , X_i , k_i , and U_i are as defined for Eq. 1 above, and where K is a parameter that allows the user extra flexibility in defining the MUF. Associated with this multiplicative type of MUF are the following two relationships between the constants used in that MUF.

$$\sum_{i=1}^n k_i \neq 1 \quad (4)$$

$$1 + K = \prod_{i=1}^n (1 + K k_i) \quad (5)$$

Because Eq. 5 holds true whenever the multiplicative form of the MUF is employed, the user does not have to enter the constant K in the input to IDAP. The program will calculate it automatically (see the CALCK procedure description in the next section).

3.2.2 Marginal Functions

Marginal functions are used to represent mathematically the user's preferences. Frequently, these preferences are nonlinear, and they are not always continuous. They serve only as functions that indicate the relative value or "desirability" of attribute levels or events. Fig. 3.2 illustrates several examples of IDAP's marginal (single attribute) utility functions. The vertical axes of all graphs in that figure represent the utility of the attribute level represented by the horizontal axis. When a user is subject to a lottery, preferences are often used to generate a marginal function for the attribute that serves as the "prize" in that lottery. Assume, for example, that an individual is presented with the following question: "Suppose you were presented with a lottery ticket that offered a 50-50 chance at \$100. For how much money would you sell that ticket?" Some individuals may answer \$25, others \$35, others \$10 and so on. There is no "correct" answer: the utility value of the lottery (the expected value is 50% of \$100, or \$50) varies from person to person, depending on how averse to risk that individual is when it comes to the attribute of, in this case, money. An EMV person (an individual who uses "Expected Monetary Value" as a criterion in decision making) would have replied \$50 to the above question; the associated utility function would be linear. A gambler may have answered "\$55," and would therefore be considered "risk prone."

In practical situations, preferences do not always follow clean-cut functions, and a method of curve-fitting (such as the least-squares method) may be necessary to accommodate the function to the user's preferences.

GENERAL FORM

LINEAR FORMS

EXPONENTIAL FORM
(DESIRABLE ATTRIBUTES) EXPONENTIAL FORM
(UNDESIRABLE ATTRIBUTES)

GAMMA FORM

QUADRATIC FORM

Fig. 3.2 IDAP's Marginal Functions

In this section, the seven types of marginal functions (each operating on one attribute only) available in IDAP will be discussed separately.

In the equations that follow, note that X' represents the normalized level of an attribute.

$$X' = (X_i - X_{\min}) / (X_{\max} - X_{\min}) \quad (6)$$

where $X_{\max}$ and $X_{\min}$, calculated by IDAP, represent the extreme levels for attribute i only. The levels of attributes found at the terminal nodes of the tree are searched for these extremes.

The general form of a marginal function is useful when the user is not able to employ other forms conveniently. Values of this form are provided directly by the user. Typically the user breaks down the range of attribute levels into n parts and assigns a utility value to each part, as shown in the step function of Fig. 3.2. Note that attribute levels will not be normalized by IDAP before this type of function is invoked. This function allows the user extreme flexibility in assigning utilities, because there is no practical limit to the number of parts into which the range of attribute levels may be divided.

Two linear marginal functions are also available to the user. One is used for desirable attributes, and the other for undesirable attributes. Desirable attributes may include an increase in income, good will, better efficiency, and so on; undesirable attributes may include ill will, pollution, fines, theft, misinformation, cost of regulations, and the like. The desirable form of the linear marginal function may be represented by:

$$U = X' \quad (7)$$

This type of function is applicable to EMV persons (individuals who are neither risk-averse nor risk-prone when it is time to take chances). No parameters need be entered by the user when the decision is made to use either linear marginal function. Figure 3.2 illustrates these linear forms of IDAP's marginal functions.

The linear form of a marginal function that is used for undesirable attributes can be represented by the following equation.

$$U = 1 - X' \quad (8)$$

The exponential form is available to the user in two types, the form used for desirable attributes, and the form used for undesirable attributes. The exponential (desirable) form of the marginal function is represented by the following equation:

$$U = \frac{1 - \exp(-rX')}{1 - \exp(-r)} \quad (9)$$

where r , the risk parameter, must be supplied by the user and serves to determine uniquely the shape of the resulting curve, as shown in Fig. 3.2, and " $\exp(i)$ " represents e (the base of natural logarithms) raised to the power i .

Note that this function is monotonically increasing and that it may be concave upward or downward, depending on the sign of r . Maximum and minimum values of this function are reached at $X=1$ and $X=0$, respectively. If $r>0$, then the function is said to be of the risk-averse type, and the function is of the risk-prone type if $r<0$. As r approaches 0, the function mathematically reduces to the linear (desirable) form.

A marginal function that has the exponential (undesirable) form can be expressed by the following equation:

$$U = \frac{1 - \exp[-r(1 - X')]}{1 - \exp(-r)} \quad (10)$$

where r is defined as in Eq. 9 above.

As shown in Fig. 3.2, the exponential forms of the marginal functions are monotonic, and they may be concave upward or downward. Note that Eqs. 9 and 10 are identical if $1-X'$ is replaced with X' in the latter, and that the maximum and minimum values are reversed. As r of Eqs. 9 and 10 approaches 0, these equations reduce to the linear forms of Eqs. 7 and 8, respectively.

The gamma form of the marginal function, as seen below, is slightly more complex and is often useful when the optimum utility lies at neither extreme.

$$U = \left(\frac{X'}{ab}\right)^a \exp\left(-\frac{X' - ab}{b}\right), \text{ for } a > -1 \text{ and } b > 0 \quad (11)$$

where parameters a and b must be specified in the input to IDAP.

Various forms of this function are illustrated in Fig. 3.2, and it should be noted that the general shape of the function varies according to the parameters a and b . There are constraints on the two parameters, as noted in Eq. 11. Note that as a approaches 0, the form mathematically reduces to an exponential equation.

The quadratic form of the marginal function is:

$$U = a + 2bX' - c(X')^2 \quad (12)$$

where the parameters a , b , and c must be supplied by the user. This form of the marginal function is shown in Fig. 3.2.

When evaluating the three parameters for the quadratic form of the marginal function, it is often helpful to note the following points:

1. The shape of the quadratic equation is parabolic.
2. The function reaches a maximum or minimum at the (X,Y) coordinate $(b/c, a + b^2/c)$.
3. The function is concave upward for $c<0$, and concave downward for $c>0$.
4. $U(0) = a$.

5. In determining the parameters a , b , and c that will serve to define uniquely the desired quadratic (parabolic) function, the user may substitute the coordinates of three points into the equation and algebraically solve these three equations for the three unknown parameters.
6. If the user knows the value at which the function reaches its maximum (or minimum), then coordinate values of one other point may be substituted into the equation. The resulting equation may be coupled with the information provided in point 2 above when solving for the three parameters.

4 THE IDAP - A USER'S GUIDE

Up to this point, the emphasis of this document has been on the theory behind the IDAP and on the programming aspect of the two programs of the IDAP. This section is intended for the user of the IDAP; it explains the input, how to run the programs, and what to expect as output.

4.1 LIST OF COMMANDS

Once input to IDAP has been created in the RAWDAT file(s), the user may execute the IDAP program in batch mode. Control of most of the processes within this PL/1 program is through the proper use of commands.

This section provides a brief explanation of all the commands available to the user for Version 2 of IDAP. As will be explained in the examples, these commands must be placed in the proper input file. IDAP assumes that all input is in the form of punched data cards, although other types of input, such as datasets on magnetic disks, may be used. (Contact the authors for more information.)

There are several formatting rules that must be followed when creating commands for the IDAP program. They are explained along with each command listed in the following pages. The notation used in these lists will now be explained.

- o Upper case strings are literals; they should be coded exactly as shown in the following lists.
- o For strings in lower case the user should substitute the appropriate value or string as indicated by the command in question.
- o The use of the single quotation mark (') must be observed; if it is not coded, an error may result.
- o In several cases, it is necessary to code the "sharp sign" (#) in commands. This character has a specific meaning to IDAP and should be coded as shown.

Note that it is not necessary to left- or right-justify any commands, although there may be no blank spaces embedded in strings enclosed in single quotes. In all cases, only a single command may be coded on the same card, although in several cases, as explained below, there may be multiple cards containing a single command sequence.

Coding commands is simple once the user understands exactly what each command does; they are straightforward in syntax.* In several cases there is a hierarchy of commands, but again, when the user understands how each command operates, the ordering of the commands will seem logical. BUILD must be executed before PROCESS, before LIST, and before ALTER. Other than these three

*Individuals are invited to comment on the syntax of the commands used by IDA. These and any other relevant suggestions may be submitted to one of the authors.

hierarchical "rules," all other commands may be coded in any order. Note, however, that END causes the complete termination of the user's session; it should be used only when the program is to be ended.

The following is a complete listing of all commands that may be executed by the IDAP PL/1 program. Error messages produced by IDAP when an invalid command is input are listed later in this document. Note that when an invalid command is input to IDAP, the entire program terminates immediately after an appropriate error message is printed. In future versions of IDAP errors may not necessarily cause program termination, but at present they do. Questions may be addressed to the author.

*user's comment line

If IDAP encounters an "*" in column 1 of the user's command, that command will be ignored. This provides a convenient way to make notes in the user input data file.

```
ALTER MARG # number
      name
      parameter-1
      parameter-2
      parameter-3
```

The various ALTER commands explained below allow the user to change the values associated with a decision tree. Note, however, that the structure (shape) of the tree may not be altered by the user once it has been created. Only the data associated with nodes and with marginal functions may be altered.

By using the above command sequence (a command of more than one card), the user is able to change the three parameters and the name of the marginal functions that operate on attribute number "number." (For the string "number" shown above, the user should substitute the appropriate integer that is the reference number of the attribute in question.)

The user must enter values for all parameters shown above; however, a value identical to the existing one may be used, and therefore the parameter would remain unchanged.

Note the use of the character "#"; it conveys a special meaning to IDAP, principally to allow the program to distinguish this command from the following command.

```
ALTER MARG
      name
      parameter-1
      parameter-2
      parameter-3
```

(These four cards must
be repeated for all
marginal functions
defined by the user.)

This command is similar to the previous one, except that the user may, in this case, change all the names and parameters associated with all the marginal functions. Note that at least five cards are associated with this command sequence.

```
ALTER NODE 'node-id' COST # number
      new-cost
```

This allows the user to change the level (cost or value) of the specific attribute "number" associated with the node labeled "node-id" (see Sec. 3 for information about node labels). "new-cost" should be coded on the second card of this command sequence.

```
ALTER NODE 'node-id' COST
      cost-i          (Card repeated for all attributes.)
```

This is similar to the previous command except that there are additional cards for the new levels ("cost-i") of each attribute associated with the node labeled "node-id."

```
ALTER NODE 'node-id' PROB
      new-probability
```

This command allows the user to change the probability of a node labeled "node-id." The node's father must be a probability node in order that this command be valid.

```
BUILD 'filename'
```

When executed, this command reads the formatted input data located in "filename" (either RAWDAT1, RAWDAT2, or RAWDAT3) and creates a decision tree for the user.

```
END
```

This command causes IDAP to stop reading the user's commands and terminate execution of the PL/I program. It should be used at the end of the user's commands, although there is a routine for handling "end of file" errors.

```
LIST INPUT
```

This simple command lists all data associated with the user's decision tree. Each node is listed separately, in the order in which they were input by the user. For a discussion of altering the quantity of printout generated by this command, refer to the description of the variables found within IDAP's main procedure.

```
LIST MARG 'attribute-number'
```

This command lists data associated with the marginal function that operates on the attribute whose number appears in the command as "attribute-number."

LIST NODE 'node-id1' 'node-id2' 'node-id3' ... 'node-idn'

All nodes whose IDs or labels appear as 'node-idi' will be listed when this command is executed by IDAP. This command is limited to 80 characters (one input line); therefore the string of IDs may not extend to other input cards.

LIST SUBTREE 'node-id'

This command lists the subtree of the node labeled 'node-id.'

LIST UTIL 'utility-function-number'

The user is allowed to define any number of utility functions during the execution of IDAP, even though only one of these functions may be used when a tree is being processed. This command allows the user to list the data associated with one of these functions.

OUTPUT PAGE 'users-notation'

This command allows the user to insert comments into the DISPLAY output file. Such a command will immediately place user's notation into the output stream. If the keyword PAGE is included (it is optional), a form-feed signal will be issued and the user's comment will be placed at the top of the next output page.

PROCESS 'utility-function-number'

This PROCESS command is the major command of IDAP. It causes the program to "average out" and "fold back," as described previously. Only one utility function may be used each time a tree is processed, and that function is denoted by the string "utility-function-number" in the user's command.

Note that the PROCESS command can be executed more than once on the same decision tree.

SENS MARG 'att' UTIL u FROM f TO t BY b

This command allows the user to change parameter l of the marginal function operating on attribute 'att' from the value f to the value t in increments of b. IDAP processes the tree and generates a terse listing of the utility values of all initial nodes and their utility values for each iteration. Utility function number u is used in processing the tree. Note the use of the quotation marks seen in the command format above.

SENS UTIL 'u' ATT i FROM f TO t BY b

This command allows the user to change the value of k_i (MUF constant for attribute i) for utility function number 'u' (again, note the use of quotation marks) in increments of b from f to t , and process the tree for each iteration. A terse listing of the initial nodes and their utility values is generated each time the tree is processed.

When a specific k_i in a utility function is altered, several calculations must be made by IDAP. If the function is of the additive form, the ratios of the k_j (all k except k_i) are held constant. If it is of the multiplicative form, the k_j remain unchanged. If the sum of k_j and k_i is 1, the additive MUF form is used, otherwise K is calculated and the multiplicative form is used. If the form is found to be multiplicative, K is evaluated (see Eq. 5).

For each iteration in the analysis, k_i is changed to k'_i as an independent variable. Similarly, each k_j in k_i is changed to k'_j according to specific rules.

In the additive case, the MUF form remains additive and:

$$k_j/k'_j = (1 - k_i)/(1 - k'_i) \quad (13)$$

and therefore

$$k'_j = k_j(1 - k'_i)/(1 - k_i) \quad (14)$$

In the multiplicative case, the MUF form may change, but

$$k'_j = k_j \text{ for all } j \neq i. \quad (15)$$

SENS NODE 'n' ATT a UTIL u FROM f TO t BY b

This command allows the user to perform a sensitivity analysis on attribute levels. The level of attribute a is incremented in steps of b from f to t . A node's attribute level may be considered a toll in travelling from its parent to the node in question. Utility function number u will be used in processing, and the node designation 'n' must be enclosed in quotes.

(The following information may be useful when performing sensitivity analyses on attribute levels: when the averaging-out routine of the procedure is complete, IDAP searches for the maximum and minimum attribute levels stored at the terminal nodes of the user's decision tree. These extremes are used in calculations with marginal functions and influence the final utility values of nodes. When performing sensitivity analyses on attribute levels, it should be kept in mind that these extremes could change, and this change could cause misleading results. Theoretically, a marginal function is defined over an interval, the interval boundaries being the extremes of an attribute's level. If the extremes are altered when sensitivity

analysis calculations are performed, the function may be invalid. One way to avoid the potential problem of having these extremes changed, or to avoid the necessity of monitoring them, is by including two nodes that act as dummies in the decision tree. One node may contain the maximum levels for all attributes, and the other the minimum levels. As long as the attribute levels in these two dummy nodes remain unchanged, the resulting utility values should not be misleading or inconsistent with other similar data.)

Note that after all sensitivity analyses are completed, IDAP resets any of the parameters that may have changed during the calculations to their original values.

SET DEBUG 'bit-string'

In this command, the single quotation marks enclose the value that the variable DEBUG will become. This value must be a two-digit bit string (00, 01, 10, or 11). By default, DEBUG='11'B, and this value produces the maximum amount of output.

SET PROF 'bit-string'

In this command, as in the SET DEBUG command, the user is allowed to code a string that will be assigned to a variable used by IDAP. That bit string must be three characters in length (000, 001, 010, 011, 100, 101, 110, or 111). At present, PROFICI (the variable name used by IDAP) is not used much; altering that variable will not produce significant results.

4.2 RAWDAT INPUT FILE FORMAT

There is a rigid format that must be followed when coding data that will eventually be used during the construction of a decision tree. These data, as explained previously, must be placed on the input data cards of a RAWDAT file; the data may be coded directly by the user, or by executing the IDAIN procedure that formats the data automatically. Note that a single RAWDAT file must contain all the data that defines a decision tree, and only one decision tree may be defined in a single RAWDAT input file.

Data in RAWDAT files are organized into sequences. There are four types of sequences that must be present in a RAWDAT file in order to define the user's decision tree completely. A nonempty RAWDAT file is composed of:

1. A parameter-setting sequence,
2. Marginal function sequences,
3. Utility function sequences, and
4. Node sequences.

Each of these sequences will be discussed in detail later in this section. Note that a parameter-setting sequence (of only one input card) must be

first in the RAWDAT file. The other sequences may follow in any order. One marginal function sequence must be coded for each marginal function (for each attribute) desired. One utility function sequence must be coded for each such function the user wishes to define; more than one may be defined, although IDAP requires only one at a time during the processing of a tree. One node sequence must be entered for each node in the user's decision tree.

SEQUENCE 1: Parameter setting.

PURPOSE: To indicate the general form of the decision tree.

NUMBER OF CARDS: 1 (Must be the first card in a RAWDAT file.)

COLUMN 1: "P" - to indicate that this is a parameter-setting card.

COLUMNS 2-6: Number of levels in the user's tree.

COLUMNS 7-11: Maximum number of nodes in tree.

COLUMNS 12-16: Number of attributes considered.

COLUMNS 17-21: Number of MUFs defined by the user.

Note that within the fields reserved for arithmetic data, the user may left- or right-justify data. It is not necessary to place decimal points after integers.

After this first card, the user must supply the three other types of sequences in order that the decision tree may be totally defined for IDAP.

SEQUENCE 2: Marginal function.

PURPOSE: To define a marginal function.

NUMBER OF CARDS: 1 (Except as noted for the GENERAL function below).

COLUMN 1: "M"

COLUMNS 2-4: The attribute on which this function is to operate.

COLUMNS 5-8: The four-character code naming this function.

COLUMNS 10-19: Parameter one of this function. (See notes below.)

COLUMNS 20-29: Parameter two of this function.

COLUMNS 30-39: Parameter three of this function.

IDAP recognizes seven marginal functions. (Recall that a marginal function is a utility function that treats only a single attribute.) Each of these has a four-character code that is used to specify the type of function, as shown here:

GENE - The general form - refer to Sec. 3 and notes below.

LIND - The linear (desirable) form - see Eq. 7 of Sec. 3.

LINU - The linear (undesirable) form - Eq. 8.

EXPD - The exponential (desirable) form - Eq. 9.

EXPU - The exponential (undesirable) form - Eq. 10.

GAMM - The gamma form - Eq. 11.

QUAD - The quadratic form - Eq. 12.

One marginal function sequence must be coded for each attribute considered by IDAP; such a function is totally defined once the name (code) and three parameters are coded, as shown below.

TYPE (CODE)	-- PARAMETER --		
	1	2	3
-----	---	---	---
GENE	see notes below		
LINE	0	0	0
EXPD	r	0	0
EXPU	r	0	0
GAMM	a	b	0
QUAD	a	b	c

For the above parameters the user must substitute the appropriate values of the variables shown in the equations of Sec. 3. Note that a blank space may be substituted for the zero character (0) when coding this sequence in a RAWDAT file. Also, there is no need to code a decimal point for integer valued data.

IDAP requires two types of parameters when the user wishes to input the data defining the general marginal function form. The general form of IDAP's marginal functions is mathematically described as a step function. The number of steps must be coded by the user as Parameter 1 in the above table. The other two parameters in that table are zero (or blank spaces). These three parameters are the first parameters required by IDAP when constructing this function. Parameters of the second type define the intervals and the function values to be assigned to these intervals (steps).

When evaluating a general marginal function, IDAP searches the intervals that were defined by the user in a RAWDAT input file. The order in which this search is carried out is the same order in which the user specified the intervals in that RAWDAT file. These intervals are considered by IDAP to be "closed right-ended." That is, a point belongs to an interval if the value of that point is less than or equal to the maximum (right end) value of that interval. Since the intervals are searched as ordered by the user, the user must be careful when coding such data. The steps and the function values to be assigned to these steps must be coded in numerical order, from the steps (intervals) defined for the smallest abscissa to those defined for the largest abscissa values.

Parameters of the second type needed by IDAP in order to define a general marginal function are paired. The first value of a pair may be considered the right end of an interval. The second value in that pair is the functional value that will be assigned to all abscissa values less than or equal to that shown as the first value in that pair. Once these pairs are ordered as described above, IDAP will be able to formulate the correct functional value of any appropriate number.

When the user codes data for a general marginal function, the first card (as shown in the above table) must be followed by a number of other cards containing the pairs of parameters mentioned above. Each of these extra cards is formatted as follows.

SEQUENCE 2 (Cont'd): Additional cards for general marginal function.
PURPOSE: To define the steps and their function values.

NUMBER OF CARDS: $\text{Max}(\# \text{steps}/2, \# \text{steps}/2+1)$.
 COLUMNS 10-19: Maximum abscissa (attribute level) for step a.
 COLUMNS 20-29: Functional value to be assigned to step a.
 COLUMNS 30-39: Maximum abscissa (attribute level) for step b.
 COLUMNS 40-49: Functional value to be assigned to step b.

Each card may contain the parameters defining two steps in the user's general marginal function, although the last card of this sequence may contain only one pair of such parameters. Additional cards must be used until there are no more steps to be defined by the user.

A utility function sequence, when interpreted by IDAP, serves to define a utility function established by the user. Note that the user is allowed to define any number of utility functions, although only one may be used at a time when the "averaging out" and "folding back" process is executed.

```

SEQUENCE 3: Utility function.
PURPOSE: To define a utility function (MUF).
NUMBER OF CARDS: Variable; at least two.
CARD 1, COLUMN 1: "U"
CARD 1, COLUMNS 2-4: Utility function reference integer.
CARD 1, COLUMNS 5-7: "ADD" if the MUF is additive in form, or
                        "MUL" if the MUF is multiplicative in form;
CARDS n onward (n >=2):
    COLUMNS 10-19: parameter (n-2)*5 + 1
    COLUMNS 20-29: parameter (n-2)*5 + 2
    COLUMNS 30-39: parameter (n-2)*5 + 3
    COLUMNS 40-49: parameter (n-2)*5 + 4
    COLUMNS 50-59: parameter (n-2)*5 + 5

```

Note that the number of parameters must equal the number of attributes considered by IDAP. These parameters represent the k_i of the equations in Sec. 3.

A node sequence contains all the data required by IDAP to establish a fork (or terminal leaf) in a decision tree. The number of parameters that must be coded by the user in each node sequence equals the number of attributes considered; these parameters may be regarded as tolls or attribute levels attained when reaching that node in a decision process. Separate cards are used, with up to five parameters per card, to code these parameters in a node sequence.

SEQUENCE 4: Nodc.
PURPOSE: To define a fork or terminal node in a tree.
NUMBER OF CARDS: Variable; at least 2.
CARD 1, COLUMN 1: "N"
CARD 1, COLUMNS 2-66: Node label (see Sec. 4.4.3), followed by a
mandatory blank space, followed by any text
or comment of the user.
CARD 1, COLUMN 68: "1" if the sons of this node are options, or
"0" if they are probabalistic.

CARD 1, COLUMNS 70-80: The probability (between 0 and 1) of the outcome of this node; coded only if this node is a probability node of its father.

CARDS n onward ($n \geq 2$):

COLUMNS 10-19: Attribute $(n-2)*5 + 1$ level or toll.

COLUMNS 20-29: Attribute $(n-2)*5 + 2$ level or toll.

COLUMNS 30-39: Attribute $(n-2)*5 + 3$ level or toll.

COLUMNS 40-49: Attribute $(n-2)*5 + 4$ level or toll.

COLUMNS 50-59: Attribute $(n-2)*5 + 5$ level or toll.

Note that a blank character may be used in place of the zero character. Node nomenclature is discussed in Sec. 3.

One node sequence must be coded for each node in the user's decision tree. It is very important that, when coding these node sequences, the data for nodes of all ancestors must be coded before the data for their descendants. When a node sequence is interpreted, IDAP searches for the father of that node in the existing tree, and if the father is not found, an error will occur.

The IDAIN WYLBUR exec procedure does all the above formatting automatically. It prompts the user for all data required by IDAP, formats the data and then either saves the data set on a disk or on punched cards. It is suggested that the user execute the IDAIN procedure rather than code the data manually; errors will be held to a minimum. It should be noted that once an error occurs in the user's input, the appropriate error messages are printed and the job terminated. Future versions of IDAP may contain alternative error handling routines, but under the current version, an input error terminates the program.

Once all the above data have been coded, the user is ready to execute the IDAP PL/1 batch program. Simple JCL work is required to execute this phase of IDAP.

4.3 NODE LABELING

Each node is represented by a PL/1 structure in the IDAP program. The details of this structure are described in Sec. 3.1, and the variable NODE.ID will be examined in this section.

The ID of a node is very important to IDAP; it is through the use of this ID that IDAP is able to identify a node and link it, through the use of PL/1 pointers, to its relatives as discussed previously. In addition, it is through the text in the ID that the user is able to recognize each node easily.

The user is able to maintain total control of the decision tree's shape through the use of NODE.ID. This variable contains 65 characters, and there are several rules that must be followed when coding such a label. The node's "hierarchy code" must start in column 1 of the node label and must be followed by at least one blank space. The remainder of this 65-character variable may be filled with user's comments or text. When printing the data contained in a node, IDAP always prints out both the hierarchy code and the user's text in

the node label. Note that the single quotation mark (') should not be used in the user's text in a node label; interpretation of certain commands relies on the use of single quotes, and their presence in the user's comments may cause errors.

The hierarchy code, as seen in Fig. 4.1, is a string of integers separated by periods. The individual integers represent an ancestor of the node in question. For example, a node whose hierarchy code is "2.1.3" is the third son of the first son of the second initial option in the decision tree. A node whose hierarchy code is "2.3.12.17" is the seventeenth son of the twelfth son of the third son of the second initial option. Note that the last character in such a code should not be a period.

The nomenclature is straightforward, and much information is represented in very little space. Each node has a unique label. The level of a node is easily obtained: it is the number of periods in the hierarchy code plus one, or the number of separated integers in the code (see Sec. 2.1). All nodes that are descendants of the same node (call it node A), have the hierarchy code of A present at the beginning of their own hierarchy codes. The hierarchy codes of brothers (sons of the same father) vary only in their last integer.

4.4 EXAMPLE 1

These next sections provide three examples of the use of IDAP. These three examples are similar in that the basic shape of the tree remains unchanged, although utility functions and a probability distribution were changed*. The theory used to obtain the parameters for the various functions of these examples will not be discussed. A good example, including a brief discussion of the relevant theory behind decision analysis, is presented in Keeney.⁸

The session with IDAIN will be shown only in Example 1, but in all cases, the JCL, RAWDAT files, and output files are shown.

4.4.1 Input

The problem considered in the following example is simple. Should I go to Las Vegas or not? Three attributes are considered: "money," "excitement," and "frustration at money lost." If I do go to Las Vegas, I will play only two types of games: roulette and blackjack. The possibility of finding or losing, by accident, a certain amount of money will also be considered in the system. Thus, as can be seen in Table 4.1 and in Fig. 4.1, there are no option nodes in the tree except for the initial options.

The initial "tolls" associated with Node 1 may be considered airplane fare and the excitement associated with the plane trip.

The MUF used in these three examples has the additive form. The marginal functions are illustrated in Fig. 4.2. The various k_i were changed in

*By altering probability distributions, utility functions, and/or utility levels or tolls, the user may perform a sensitivity analysis on the system.

Fig. 4.1 Example 1 Decision Tree

different examples. Note that sessions with IDAIN, as seen in the following listings, may be executed only after the user is properly logged onto WYLBUR at ANL. Although IDAIN was used in all three cases, only the session from the first example is shown here along with inserted comments.

Note the presence of the RAWDAT DD card in the JCL listing; that dataset contains the user's formatted "tree data." The USER DD cards are the user's commands.

Table 4.1. Nodes and Parameters of Sample Decision Problem

Node Hier. Code	Description	Probability	Attribute Level		
			\$	Excit.	Frust.
1	Go to Las Vegas	option	-200	0.2	0.0
2	Do not go to Las Vegas	option	0	0	0
1.1	Win \$100 at roulette	0.15	100	0.7	0
1.2	Win \$200 at roulette	0.05	200	0.8	0
1.3	No gain/loss at roulette	0.5	0	0.8	0.5
1.4	Lose \$50 at roulette	0.3	-50	0.2	1.0
1.1.1	Win \$50 at blackjack	0.3	50	0.3	0
1.2.1	Win \$50 at blackJack	0.3	50	0.4	0
1.1.2	Lose \$50 at blackjack	0.2	-50	0.2	0.4
1.2.2	Lose \$50 at blackjack	0.2	-50	0.2	0.3
1.1.3	No gain/loss at blackjack	0.5	0	0.1	0.2
1.2.3	No gain/loss at blackjack	0.3	0	0.1	0.1
1.2.1.1	Find \$100 by accident!	0.2	100	0.9	0
1.2.1.2	Lose \$100 by accident	0.2	-100	0.3	0.5
1.2.1.3	No accidental gain/loss	0.6	0	0	0

Attribute 1: money

Form: linear, ENV

$$U(x) = x$$

Attribute 2: excitement

Form: exponential (desirable),
risk seeking

$$U(x) = \frac{1 - e^{-rx}}{1 - e^{-r}}$$

$$U(0.6) = 0.5$$

$$r = -0.83$$

Attribute 3: frustration at loss of money

Form: exponential (undesirable)
risk averse

$$U(x) = \frac{1 - e^{-r(1-x)}}{1 - e^{-r}}$$

$$U(0.4) = 0.7$$

$$r = 0.83$$

Fig. 4.2 Marginal Functions for Example 1

4.4.2 Annotated IDAIN Procedure for Example 1

USE IDA.EXEC CLE ----- The dataset containing IDAIN. If the user is logged onto his own badge, \$B99832.IDA.EXEC should be used. This initiates execution of the IDAIN procedure.

? EXE ACT LINES 1/1 MODEL CLE -----

THIS IS A PROC FOR CREATING AN INPUT FILE FOR THE IDA PROGRAM. YOU WILL BE PROMPTED TO ENTER A NUMBER OF THINGS FOR THIS FILE. REMEMBER TO PRESS THE CARRAIGE-RETURN KEY SO YOUR ENTRY WILL BE PROCESSED.

DO YOU HAVE A FILE TO WHICH YOU WANT TO ADD DATA? ...

IF SO, ENTER THAT FILE NAME, ELSE CR: ----- Carraige-Return.

EXACTLY HOW MANY LEVELS WILL THERE BE IN THE TREE? 4 ----- The user types in "4", as appropriate.

WHAT IS THE UPPER LIMIT OF THE NUMBER OF NODES IN THE TREE? 16 ----- The user types in "16", as appropriate.

EXACTLY HOW MANY ATTRIBUTES WILL THERE BE ASSOCIATED WITH EACH NODE? 3 ----- The user types in "3", as appropriate.

HOW MANY UTILITY FUNCTIONS (UPPER LIMIT) WILL YOU WANT? 1

EXPLANATION FOR MARGINAL FUNCTION? Y ----- Either "Y" or "N".

Marginal functions in this procedure can take any of the following forms:

NAME	CODE	FORM	PARAMETER 1	2	3
GENERAL	GENE	specify upper limits and values in the 'a' intervals.	a	0	0
LINEAR	LINE	$M = X'$	0	0	0
EXPONENTIAL (DESIREABLE)	EXPD	$M = \frac{1 - \text{EXP}(-rX')}{1 - \text{EXP}(-r)}$	r	0	0
EXPONENTIAL (UNDESIREABLE)	EXPUN	$M = \frac{1 - \text{EXP}(-r(1-X'))}{1 - \text{EXP}(-r)}$	r	0	0
GAMMA	GAMM	$M = \frac{(X')^a}{(ab)} * \text{EXP}(-\frac{X' - ab}{b})$	a	b	0
QUADRATIC	QUAD	$M = a + 2bX' - c(X')^2$	a	b	c

In all cases, $0 \leq X' \leq 1$ because

$$X' = \frac{X - \text{MIN}(X_i)}{\text{MAX}(X_i) - \text{MIN}(X_i)}$$

The following illustrates a GENERAL marginal function where PARAMETER 1 IS 3.

This function is defined on the interval [0,1] and takes on values between 0.25 and 1.0.

WHAT IS THE FOUR-CHARACTER NAME OF THE

1 TH MARGINAL FUNCTION? LINE

WHAT IS PARAMETER 1?

WHAT IS PARAMETER 2?

WHAT IS PARAMETER 3?

The user typed in "LINE" because the linear form was appropriate.

Carriage-Returns or blank spaces.

WHAT IS THE FOUR-CHARACTER NAME OF THE

2 TH MARGINAL FUNCTION? EXPD

WHAT IS PARAMETER 1? -0.83

WHAT IS PARAMETER 2?

WHAT IS PARAMETER 3?

WHAT IS THE FOUR-CHARACTER NAME OF THE

3 TH MARGINAL FUNCTION? EXPD

WHAT IS PARAMETER 1? 0.83

WHAT IS PARAMETER 2?

WHAT IS PARAMETER 3?

NOW THE INPUT FOR THE 3 MARGINAL FUNCTIONS IS COMPLETE.

These sequences are repeated once for each attribute considered.

EXPLANATION FOR UTILITY FUNCTION? Y "Y" or "N".

The utility function $U = U(X_1, X_2, X_3, \dots, X_n)$ may be one of two forms:

The Additive form (code: ADD):

$$(1) \quad U = \sum_{i=1}^n K_i M_i(X_i) \quad \text{when } \sum_{i=1}^n K_i = 1$$

The Multiplicative form (code: MUL):

$$(2) \quad U = \frac{1}{K} \left(\prod_{i=1}^n (1 + K_i M_i(X_i)) - 1 \right) \quad \text{when } \sum_{i=1}^n K_i = 1$$

and $1 + K = \prod_{i=1}^n (1 + K_i)$

In both cases, $M_i(X_i)$ is the i th marginal function operating on the value of attribute i .

In both cases, $0 \leq U \leq 1$.

WHAT IS THE NUMBER OF THE NEXT UTILITY FUNCTION ...
 TYPE 'END' TO STOP: 1 ----- Utility Function reference number.
 WHAT IS THE THREE CHARACTER NAME OF THE
 UTILITY FUNCTION? ADD
 WHAT IS THE 1 TH PARAMETER IN
 THE UTILITY FUNCTION? 0.4
 WHAT IS THE 2 TH PARAMETER IN
 THE UTILITY FUNCTION? 0.4
 WHAT IS THE 3 TH PARAMETER IN
 THE UTILITY FUNCTION? 0.2
 WHAT IS THE NUMBER OF THE NEXT UTILITY FUNCTION ...
 TYPE 'END' TO STOP: END ----- Only one Utility Function was needed
 NOW THE UTILITY FUNCTION DATA INPUT IS COMPLETE. by the user.
 EXPLANATION FOR NODE INPUT? Y

IN THE REMAINDER OF THIS SESSION, THE TERM 'NODE
 DESIGNATION' WILL BE USED FREQUENTLY. YOU WILL BE
 PROMPTED TO ENTER A CHARACTER STRING TO BE USED IN
 THE DESIGNATION OF EACH NODE. THIS STRING WILL BE
 USED BY THE COMPUTER FOR THE IDENTIFICATION OF THE
 NODE IN QUESTION, AS WELL AS BY YOU THE USER. IT
 MUST CONTAIN THE FOLLOWING:

- THE HEIRARCHICAL REPRESENTATION OF THE
 NODE (SUCH AS 1.3.2) STARTING IN COLUMN
 ONE, WHICH MAY NOT CONTAIN BLANKS.
- AT LEAST ONE BLANK FOLLOWING THIS
 HEIRARCHICAL REPRESENTATION.
- ANY NOTATION DESIRED BY YOU THE USER,
 WHICH WILL BE IGNORED BY THE COMPUTER.
 IT MAY NOT CONTAIN QUOTATION MARKS.

IN ALL CASES, THIS NODE DESIGNATION WILL BE
 TRUNCATED TO 65 CHARACTERS.

*This string is the Node Label; it includes
 the Node Heirarchy Code.*

WHAT IS THE NEXT NODE DESIGNATION? 1 GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?

1 GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?
 REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P ----- Either "p" or "o".
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR':
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? -200
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.200
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? 2 STAY AT HOME - DONT YOU DARE GO TO LAS VEGAS

2 STAY AT HOME - DONT YOU DARE GO TO LAS VEGAS
 REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER --- This sequence is
 THAT PROBABILITY, OTHERWISE PRESS 'CR': repeated for each
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 0 node.
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? 1.1 WIN \$100 AT ROULETTE

1.1 WIN \$100 AT ROULETTE
 REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.15
 73 CHARACTERS IN LINE 510.

REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 100
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.700
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.000

WHAT IS THE NEXT NODE DESIGNATION? 1.2 WIN \$200 AT ROULETTE

1.2 WIN \$200 AT ROULETTE

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.05
 73 CHARACTERS IN LINE 512.
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 200.000
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.800
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.000

-- This sequence is
 repeated for
 each node.

WHAT IS THE NEXT NODE DESIGNATION? 1.3 NO GAIN AT ROULETTE - OR LOSS!

1.3 NO GAIN AT ROULETTE - OR LOSS!

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.5
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.800
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.500

WHAT IS THE NEXT NODE DESIGNATION? 1.4 LOSE \$50 AT ROULETTE

1.4 LOSE \$50 AT ROULETTE

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.3
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? -50.000
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.200
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 1.000

WHAT IS THE NEXT NODE DESIGNATION? 1.1.1 WIN \$50 AT BLACK JACK

1.1.1 WIN \$50 AT BLACK JACK

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.3
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 50
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.300
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? 1.2.1 WIN \$50 AT B.J.

1.2.1 WIN \$50 AT B.J.

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.4
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 50
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.400
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? 1.1.2 LOSE \$50 AT B.J.

1.1.2 LOSE \$50 AT B.J.

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': .2
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? -50.000
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.200
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.400

WHAT IS THE NEXT NODE DESIGNATION? 1.2.2 LOSE \$50 AT B.J.

1.2.2 LOSE \$50 AT B.J.

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.3
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? -50
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.200
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.300

-- This sequence is repeated
 for each node.

WHAT IS THE NEXT NODE DESIGNATION? 1.1.3 NO GAIN/LOSS AT B.J.

1.1.3 NO GAIN/LOSS AT B.J.

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.5
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.100
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.200

WHAT IS THE NEXT NODE DESIGNATION? 1.2.3 NO GAIN OR LOSS AT B.J.

1.2.3 NO GAIN OR LOSS AT B.J.

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.3
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.100
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.100

WHAT IS THE NEXT NODE DESIGNATION? 1.2.1.1 FIND \$100 BY ACCIDENT!!

1.2.1.1 FIND \$100 BY ACCIDENT!!

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? 0
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.2
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 100
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.900
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? 1.2.1.2 LOSE \$100 BY ACCIDENT

1.2.1.2 LOSE \$100 BY ACCIDENT

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? P
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.2
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? -100.000
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0.300
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0.500

WHAT IS THE NEXT NODE DESIGNATION? 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

REGARDING THE ABOVE NODE, ARE ITS DESCENDANTS PROB. OR OPT.? 0
 REGARDING THE ABOVE NODE, IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
 THAT PROBABILITY, OTHERWISE PRESS 'CR': 0.6
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 1? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 2? 0
 REGARDING THE ABOVE NODE, WHAT IS VALUE OF ATTRIBUTE 3? 0

WHAT IS THE NEXT NODE DESIGNATION? END ----- No more nodes are to be placed in
COMPLETED CONSTRUCTION OF NODE DATA CARDS. the user's decision tree.

COMPLETED NODE DATA INPUT.

NOW YOU HAVE TWO OPTIONS.

ENTER '1' IF YOU WISH TO TERMINATE THIS SESSION.

'2' IF YOU WISH TO MODIFY OR SAVE THIS DATASET.

ENTER 1 OR 2: 2 -----

The user chose to modify or examine
this dataset.

TYPE IN 'END' IF YOU WISH TO END THIS MODIFICATION
PROCESS AND CONTINUE ON.

DO YOU WANT TO SEE THIS DATASET? Y -----

Either "Y" or "N".

500.	P4	16	3	1		
501.	M1	LINE				
502.	M2	EXPD	-0.83			
503.	M3	EXPU	0.83			
504.	U1	ADD				
505.			0.4	0.4	0.2	
506.	N1	GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?				0
507.			-200	0.200	0	
508.	N2	STAY AT HOME - DONT YOU DARE GO TO LAS VEGAS				0
509.			0	0	0	
510.	N1.1	WIN \$100 AT ROULETTE				0 0.15
511.			100	0.700	0.000	
512.	N1.2	WIN \$200 AT ROULETTE				0 0.05
513.			200.000	0.800	0.000	
514.	N1.3	NO GAIN AT ROULETTE - OR LOSS!				0 0.5
515.			0	0.800	0.500	
516.	N1.4	LOSE \$50 AT ROULETTE				0 0.3
517.			-50.000	0.200	1.000	
518.	N1.1.1	WIN \$50 AT BLACK JACK				0 0.3
519.			50	0.300	0	
520.	N1.2.1	WIN \$50 AT B.J.				0 0.4
521.			50	0.400	0	
522.	N1.1.2	LOSE \$50 AT B.J.				0 .2
523.			-50.000	0.200	0.400	
524.	N1.2.2	LOSE \$50 AT B.J.				0 0.3
525.			-50	0.200	0.300	
526.	N1.1.3	NO GAIN/LOSS AT B.J.				0 0.5
527.			0	0.100	0.200	
528.	N1.2.3	NO GAIN OR LOSS AT B.J.				0 0.3
529.			0	0.100	0.100	
530.	N1.2.1.1	FIND \$100 BY ACCIDENT!!				1 0.2
531.			100	0.900	0	
532.	N1.2.1.2	LOSE \$100 BY ACCIDENT				0 0.2
533.			-100.000	0.300	0.500	
534.	N1.2.1.3	LOSE NOR GAIN \$ BY ACCIDENT.				1 0.6
535.			0	0	0	

TYPE IN 'END' IF YOU WISH TO END THIS MODIFICATION
PROCESS AND CONTINUE ON.

DO YOU WANT TO SEE THIS DATASET? N

DO YOU WISH TO MOVE LINES OF THIS DATASET
TO OTHER POSITIONS? N

DO YOU WISH TO DELETE LINES? N

TYPE IN 'END' IF YOU WISH TO END THIS MODIFICATION
PROCESS AND CONTINUE ON.

DO YOU WANT TO SEE THIS DATASET? END

The user chose to end "modifications".

DONE WITH MODIFICATIONS.

DO YOU WANT TO SAVE THIS DATASET? Y
TO BE SAVED AS WHAT, AND VOLUME: IDADATA ON TST502
IDADATA SAVED ON TST502

----- The user chose to save this dataset
(on a magnetic disk).

DO YOU WANT THIS DATASET PUNCHED? Y
JOB WYL100 SUBMITTED

----- The user also chose to punch this dataset
on cards..

DONE!!
EXEC END
?

----- This is the job that will produce the
punched cards. Note the name of the
job well!

----- This indicates the end of the IDAIN
session.

4.4.3 Input: JCL and RAWDAT

```

P4      16      3      1
M1 LINE
M2 EXPD -0.83
M3 EXPD 0.83
U1 ADD
      0.4      0.4      0.2
N1 GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?      0
      -200      0.200      0
N2 STAY HOME - DON'T YOU DARE GO TO LAS VEGAS!      0
      0      0      0
N1.1 WIN $100 AT ROULETTE      0 0.15
      100      0.700      0.000
N1.2 WIN $200 AT ROULETTE      0 0.05
      200.000      0.800      0.000
N1.3 NO GAIN AT ROULETTE - OR LOSS!      0 0.5
      0      0.800      0.500
N1.4 LOSE $50 AT ROULETTE      0 0.3
      -50.000      0.200      1.000
N1.1.1 WIN $50 AT BLACK JACK      0 0.3
      50      0.300      0
N1.2.1 WIN $50 AT B.J.      0 0.4
      50      0.400      0
N1.1.2 LOSE $50 AT B.J.      0 0.2
      -50.000      0.200      0.400
N1.2.2 LOSE $50 AT BLACK JACK      0 0.3
      -50      0.200      0.300
N1.1.3 NO GAIN/LOSS AT B.J.      0 0.5
      0      0.100      0.200
N1.2.3 NO GAIN OR LOSS AT B.J.      0 0.3
      0      0.100      0.100
N1.2.1.1 FIND $100 BY ACCIDENT!!      1 0.2
      100      0.900      0
N1.2.1.2 LOSE $100 BY ACCIDENT      0 0.2
      -100.000      0.300      0.500
N1.2.1.3 LOSE NOR GAIN $ BY ACCIDENT      1 0.6
      0      0      0

```

JCL FOR EXAMPLE 1

```

//IDA JOB (L99832,5,,2),REGION=250K,CLASS=A,PRTY=N
          99832          8J411-30190
//*FORMAT PR,DDNAME=,DEST=3800
//*
//*
//*
//STEP1 EXEC PGM=IDA
//STEPLIB DD DSN=C190.899832.LOADERS,DISP=SHR
//*
//*
//*
//***** DATA FOR NODES OF TREE
//*
//GO,RAWDAT1 DD DSN=899832.IDA.NODEDATA,DISP=SHR
//GO,RAWDAT2 DD DUMMY
//GO,RAWDAT3 DD DUMMY
//*
//*
//*
//***** USERS COMMANDS
//*
//GO.USER DD *
BUILD 'RAWDAT1'
LIST INPUT
PROCESS '1'
END
//GO.DISPLAY DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//*

```


4.4.4 Output

The following is the DISPLAY output file that was produced after the IDAP PL/1 program was executed. The last two pages of that printout show the final conclusion.

WELCOME TO IDA!

AN INTERACTIVE PROGRAM
FOR DECISION ANALYSIS.

TODAY IS MARCH 1, 1979. THE TIME IS 9:29:23:284 AM.

**** COMMAND:BUILD 'RAWDAT1'

RAW DATA CARDS BEING INPUT ...
TREE CONSTRUCTION INITIATED ...

INITIATE PROCESSING OF PARMETER CARD ...
LEVELS IN TREE = 4
NODES IN TREE <= 16
ATTRIBUTES = 3
UTILITY FUNCTIONS = 1

INITIATE PROCESSING OF MARGINAL FUNCTION CARD ...
THIS IS A LISTING OF THE MARGINAL FUNCTION
FOR ATTRIBUTE NUMBER 1
THE NAME OF THE FUNCTION IS: LINE
PARAMETER 1: 0.00000E+00
PARAMETER 2: 0.00000E+00
PARAMETER 3: 0.00000E+00

INITIATE PROCESSING OF MARGINAL FUNCTION CARD ...
THIS IS A LISTING OF THE MARGINAL FUNCTION
FOR ATTRIBUTE NUMBER 2
THE NAME OF THE FUNCTION IS: EXPD
PARAMETER 1: -8.30000E-01
PARAMETER 2: 0.00000E+00
PARAMETER 3: 0.00000E+00

INITIATE PROCESSING OF MARGINAL FUNCTION CARD ...
THIS IS A LISTING OF THE MARGINAL FUNCTION
FOR ATTRIBUTE NUMBER 3
THE NAME OF THE FUNCTION IS: EXPD
PARAMETER 1: 8.30000E-01
PARAMETER 2: 0.00000E+00
PARAMETER 3: 0.00000E+00

INITIATE PROCESSING OF UTILITY FUNCTION CARD ...
UTILITY FUNCTION HAS THE FORM: ADD
UTILITY FUNCTION ADD
PARAMETER 1: 4.00000E-01
PARAMETER 2: 4.00000E-01
PARAMETER 3: 2.00000E-01
UTILITY CONSTANT K: 0.00000E+00

INITIATE PROCESSING OF NODE-ID CARD ...

INITIATE PROCESSING OF NODE-ID CARD ...

INITIATE PROCESSING OF NODE-ID CARD ...

INITIATE PROCESSING OF NODE-ID CARD ...

INITIATE PROCESSING OF NODE-ID CARD ...

INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...
 INITIATE PROCESSING OF NODE-ID CARD ...

END OF RAW DATA CARDS ... 15 CARDS PROCESSED.
 LOOKING FOR FATHER OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 LOOKING FOR FATHER OF 2 STAY HOME - DON'T YOU DARE GO TO LA
 LOOKING FOR FATHER OF 1.1 WIN \$100 AT ROULETTE
 FOUND FATHER
 LOOKING FOR FATHER OF 1.2 WIN \$200 AT ROULETTE
 FOUND FATHER
 LOOKING FOR FATHER OF 1.3 NO GAIN AT ROULETTE - OR LOSS!
 FOUND FATHER
 LOOKING FOR FATHER OF 1.4 LOSE \$50 AT ROULETTE
 FOUND FATHER
 LOOKING FOR FATHER OF 1.1.1 WIN \$50 AT BLACK JACK
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.2.1 WIN \$50 AT B.J.
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.1.2 LOSE \$50 AT B.J.
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.2.2 LOSE \$50 AT BLACK JACK
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.1.3 NO GAIN/LOSS AT B.J.
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.2.3 NO GAIN OR LOSS AT B.J.
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 DIDN'T FIND FATHER
 FOUND FATHER
 LOOKING FOR FATHER OF 1.2.1.1 FIND \$100 BY ACCIDENT!!

DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
FOUND FATHER

LOOKING FOR FATHER OF 1.2.1.2 LOSE \$100 BY ACCIDENT

DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
FOUND FATHER

LOOKING FOR FATHER OF 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
DIDN'T FIND FATHER
FOUND FATHER

*** RASIBLING OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN	IS 2 STAY HOME - DON'T YOU DARE GO TO LA
*** RASIBLING OF 1.1 WIN \$100 AT ROULETTE	IS 1.2 WIN \$200 AT ROULETTE
*** RASIBLING OF 1.2 WIN \$200 AT ROULETTE	IS 1.3 NO GAIN AT ROULETTE - OR LOSS!
*** RASIBLING OF 1.3 NO GAIN AT ROULETTE - OR LOSS!	IS 1.4 LOSE \$50 AT ROULETTE
*** RASIBLING OF 1.1.1 WIN \$50 AT BLACK JACK	IS 1.1.2 LOSE \$50 AT B.J.
*** RASIBLING OF 1.2.1 WIN \$50 AT B.J.	IS 1.2.2 LOSE \$50 AT BLACK JACK
*** RASIBLING OF 1.1.2 LOSE \$50 AT B.J.	IS 1.1.3 NO GAIN/LOSS AT B.J.
*** RASIBLING OF 1.2.2 LOSE \$50 AT BLACK JACK	IS 1.2.3 NO GAIN OR LOSS AT B.J.
*** RASIBLING OF 1.2.1.1 FIND \$100 BY ACCIDENT!!	IS 1.2.1.2 LOSE \$100 BY ACCIDENT
*** RASIBLING OF 1.2.1.2 LOSE \$100 BY ACCIDENT	IS 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

* POINTER LISTING OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

FATHER: NULL

#12SON: 1.1 WIN \$100 AT ROULETTE

RASIBLING: 2 STAY HOME - DON'T YOU DARE GO TO LA

DOWN: 2 STAY HOME - DON'T YOU DARE GO TO LA

* POINTER LISTING OF 1.1 WIN \$100 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#12SON: 1.1.1 WIN \$50 AT BLACK JACK

RASIBLING: 1.2 WIN \$200 AT ROULETTE

DOWN: 1.2 WIN \$200 AT ROULETTE

* POINTER LISTING OF 1.1.1 WIN \$50 AT BLACK JACK

FATHER: 1.1 WIN \$100 AT ROULETTE

#12SON: NULL

RASIBLING: 1.1.2 LOSE \$50 AT B.J.

DOWN: 1.2.1 WIN \$50 AT B.J.

* POINTER LISTING OF 1.1.2 LOSE \$50 AT B.J.

FATHER: 1.1 WIN \$100 AT ROULETTE

#12SON: NULL

RASIBLING: 1.1.3 NO GAIN/LOSS AT B.J.

DOWN: 1.2.2 LOSE \$50 AT BLACK JACK

* POINTER LISTING OF 1.1.3 NO GAIN/LOSS AT B.J.

FATHER: 1.1 WIN \$100 AT ROULETTE

#12SON: NULL

RASIBLING: 1.2.1 WIN \$50 AT B.J.

DOWN: 1.2.3 NO GAIN OR LOSS AT B.J.
 * POINTER LISTING OF 1.2.1 WIN \$50 AT B.J.
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1@SON: 1.2.1.1 FIND \$100 BY ACCIDENT!!
 R@SIBLING: 1.2.2 LOSE \$50 AT BLACK JACK
 DOWN: 1.1.2 LOSE \$50 AT B.J.
 * POINTER LISTING OF 1.2.1.1 FIND \$100 BY ACCIDENT!!
 FATHER: 1.2.1 WIN \$50 AT B.J.
 #1@SON: NULL
 R@SIBLING: 1.2.1.2 LOSE \$100 BY ACCIDENT
 DOWN: 1.2.1.2 LOSE \$100 BY ACCIDENT
 * POINTER LISTING OF 1.2.1.2 LOSE \$100 BY ACCIDENT
 FATHER: 1.2.1 WIN \$50 AT B.J.
 #1@SON: NULL
 R@SIBLING: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT
 DOWN: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

**** COMMAND:LIST INPUT

**** NODE LISTING ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 4
 COSTS:
 -2.00000E+02 2.00000E-01 0.00000E+00
 * POINTER LISTING OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 FATHER: NULL
 #1@SON: 1.1 WIN \$100 AT ROULETTE
 R@SIBLING: 2 STAY HOME - DON'T YOU DARE GO TO LA
 DOWN: 2 STAY HOME - DON'T YOU DARE GO TO LA
 THIS IS AN INITIAL OPTION.
 SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 0.00000E+00 0.00000E+00
 * POINTER LISTING OF 2 STAY HOME - DON'T YOU DARE GO TO LA
 FATHER: NULL
 #1@SON: NULL
 R@SIBLING: NULL
 DOWN: 1.1 WIN \$100 AT ROULETTE
 THIS IS AN INITIAL OPTION.
 THIS IS LEAF OF TREE.

1.1 WIN \$100 AT ROULETTE
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 3
 COSTS:
 1.00000E+02 7.00000E-01 0.00000E+00
 * POINTER LISTING OF 1.1 WIN \$100 AT ROULETTE
 FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 #1@SON: 1.1.1 WIN \$50 AT BLACK JACK
 R@SIBLING: 1.2 WIN \$200 AT ROULETTE
 DOWN: 1.2 WIN \$200 AT ROULETTE

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.15
SONS ARE PROBABILISTIC.

1.2 WIN \$200 AT ROULETTE

UTILITY: 0.00000E+00

NUMBER OF SONS: 3

COSTS:

2.00000E+02 8.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2 WIN \$200 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1SON: 1.2.1 WIN \$50 AT B.J.

RASIBLING: 1.3 NO GAIN AT ROULETTE - OR LOSS!

DOWN: 1.3 NO GAIN AT ROULETTE - OR LOSS!

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.05
SONS ARE PROBABILISTIC.

1.3 NO GAIN AT ROULETTE - OR LOSS!

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 8.00000E-01 5.00000E-01

* POINTER LISTING OF 1.3 NO GAIN AT ROULETTE - OR LOSS!

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1SON: NULL

RASIBLING: 1.4 LOSE \$50 AT ROULETTE

DOWN: 1.4 LOSE \$50 AT ROULETTE

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.50
THIS IS LEAF OF TREE.

1.4 LOSE \$50 AT ROULETTE

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 1.00000E+00

* POINTER LISTING OF 1.4 LOSE \$50 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1SON: NULL

RASIBLING: NULL

DOWN: 1.1.1 WIN \$50 AT BLACK JACK

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
THIS IS LEAF OF TREE.

1.1.1 WIN \$50 AT BLACK JACK

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

5.00000E+01 3.00000E-01 0.00000E+00

* POINTER LISTING OF 1.1.1 WIN \$50 AT BLACK JACK

FATHER: 1.1 WIN \$100 AT ROULETTE

#1SON: NULL

RASIBLING: 1.1.2 LOSE \$50 AT B.J.

DOWN: 1.2.1 WIN \$50 AT B.J.

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
THIS IS LEAF OF TREE.

1.2.1 WIN \$50 AT B.J.

UTILITY: 0.00000E+00

NUMBER OF SONS: 3

COSTS:

5.00000E+01 4.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2.1 WIN \$50 AT B.J.
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1SON: 1.2.1.1 FIND \$100 BY ACCIDENT!!
 RASIBLING: 1.2.2 LOSE \$50 AT BLACK JACK
 DOWN: 1.1.2 LOSE \$50 AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.40
 SONS ARE PROBABILISTIC.

1.1.2 LOSE \$50 AT B.J.
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 4.00000E-01
 * POINTER LISTING OF 1.1.2 LOSE \$50 AT B.J.
 FATHER: 1.1 WIN \$100 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.1.3 NO GAIN/LOSS AT B.J.
 DOWN: 1.2.2 LOSE \$50 AT BLACK JACK
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.20
 THIS IS LEAF OF TREE.

1.2.2 LOSE \$50 AT BLACK JACK
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 3.00000E-01
 * POINTER LISTING OF 1.2.2 LOSE \$50 AT BLACK JACK
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.2.3 NO GAIN OR LOSS AT B.J.
 DOWN: 1.1.3 NO GAIN/LOSS AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
 THIS IS LEAF OF TREE.

1.1.3 NO GAIN/LOSS AT B.J.
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 1.00000E-01 2.00000E-01
 * POINTER LISTING OF 1.1.3 NO GAIN/LOSS AT B.J.
 FATHER: 1.1 WIN \$100 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.2.1 WIN \$50 AT B.J.
 DOWN: 1.2.3 NO GAIN OR LOSS AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.50
 THIS IS LEAF OF TREE.

1.2.3 NO GAIN OR LOSS AT B.J.
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 1.00000E-01 1.00000E-01
 * POINTER LISTING OF 1.2.3 NO GAIN OR LOSS AT B.J.
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1SON: NULL
 RASIBLING: NULL
 DOWN: 1.2.1.1 FIND \$100 BY ACCIDENT!!
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
 THIS IS LEAF OF TREE.

1.2.1.1 FIND \$100 BY ACCIDENT!!

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

1.00000E+02 9.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2.1.1 FIND \$100 BY ACCIDENT!!

FATHER: 1.2.1 WIN \$50 AT B.J.

#1\$ON: NULL

R\$SIBLING: 1.2.1.2 LOSE \$100 BY ACCIDENT

DOWN: 1.2.1.2 LOSE \$100 BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20

THIS IS LEAF OF TREE.

1.2.1.2 LOSE \$100 BY ACCIDENT

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

-1.00000E+02 3.00000E-01 5.00000E-01

* POINTER LISTING OF 1.2.1.2 LOSE \$100 BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1\$ON: NULL

R\$SIBLING: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

DOWN: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20

THIS IS LEAF OF TREE.

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

* POINTER LISTING OF 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1\$ON: NULL

R\$SIBLING: NULL

DOWN: NULL

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.60

THIS IS LEAF OF TREE.

**** NODE LISTING COMPLETE ****

**** COMMAND:PROCESS '1'

COUNTING IN LEVEL 4

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

COUNTING IN LEVEL 3

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

COUNTING IN LEVEL 2

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

EXAMINING NEXT NODE.

```

EXAMINING NEXT NODE.
COUNTING IN LEVEL 1
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
#3LEAVE = 11 #OPT2NODES = 2
COUNTING IN LEVEL 4
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
COUNTING IN LEVEL 3
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
COUNTING IN LEVEL 2
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
COUNTING IN LEVEL 1
EXAMINING NEXT NODE.
EXAMINING NEXT NODE.
1 MAX: 1.50000E+02 MIN: -2.50000E+02
2 MAX: 2.30000E+00 MIN: 0.00000E+00
3 MAX: 1.00000E+00 MIN: 0.00000E+00

```

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

```

VALUE OF 1.00000E+00
HAS RETURNED FOR MARGINAL NUMBER 1
VALUE OF 1.00000E+00
HAS RETURNED FOR MARGINAL NUMBER 2
VALUE OF 1.00000E+00
HAS RETURNED FOR MARGINAL NUMBER 3

```

VALUE FOR UTILITY IS 1.00000E+00

1.2.1.1 FIND \$100 BY ACCIDENT!!

UTILITY: 1.00000E+00

NUMBER OF SONS: 0

COSTS:

1.00000E+02 9.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2.1.1 FIND \$100 BY ACCIDENT!!

FATHER: 1.2.1 WIN \$50 AT B.J.

#12SON: NULL

R2SIBLING: 1.2.1.2 LOSE \$100 BY ACCIDENT

DOWN: 1.2.1.2 LOSE \$100 BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

```

VALUE OF 5.00000E-01
HAS RETURNED FOR MARGINAL NUMBER 1
VALUE OF 6.54783E-01
HAS RETURNED FOR MARGINAL NUMBER 2
VALUE OF 6.02286E-01

```


HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 5.82370E-01

1.2.1.2 LOSE \$100 BY ACCIDENT

UTILITY: 5.82370E-01

NUMBER OF SONS: 0

COSTS:

-1.00000E+02 3.00000E-01 5.00000E-01

* POINTER LISTING OF 1.2.1.2 LOSE \$100 BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1SON: NULL

R0SIBLING: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

DOWN: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.20

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 7.50000E-01

HAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 5.08263E-01

HAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 1.00000E+00

HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 7.03305E-01

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

UTILITY: 7.03305E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

* POINTER LISTING OF 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1SON: NULL

R0SIBLING: NULL

DOWN: NULL

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.60

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 5.00000E-01

HAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 4.19033E-01

HAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 1.00000E+00

HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 5.67613E-01

1.1.1 WIN \$50 AT BLACK JACK

UTILITY: 5.67613E-01

NUMBER OF SONS: 0

COSTS:

5.00000E+01 3.00000E-01 0.00000E+00

* POINTER LISTING OF 1.1.1 WIN \$50 AT BLACK JACK

FATHER: 1.1 WIN \$100 AT ROULETTE

#1SON: NULL

R0SIBLING: 1.1.2 LOSE \$50 AT B.J.

DOWN: 1.2.1 WIN \$50 AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30
 THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 2.50000E-01
 WAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 3.76776E-01
 WAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 6.95548E-01
 WAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 3.89820E-01

1.1.2 LOSE \$50 AT B.J.

UTILITY: 3.89820E-01

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 4.00000E-01

* POINTER LISTING OF 1.1.2 LOSE \$50 AT B.J.

FATHER: 1.1 WIN \$100 AT ROULETTE

#1SON: NULL

RASIBLING: 1.1.3 NO GAIN/LOSS AT B.J.

DOWN: 1.2.2 LOSE \$50 AT BLACK JACK

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 3.75000E-01
 WAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 3.36017E-01
 WAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 8.60380E-01
 WAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 4.56483E-01

1.1.3 NO GAIN/LOSS AT B.J.

UTILITY: 4.56483E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 2.00000E-01

* POINTER LISTING OF 1.1.3 NO GAIN/LOSS AT B.J.

FATHER: 1.1 WIN \$100 AT ROULETTE

#1SON: NULL

RASIBLING: 1.2.1 WIN \$50 AT B.J.

DOWN: 1.2.3 NO GAIN OR LOSS AT B.J.

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.50

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 5.00000E-01
 WAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 4.19033E-01
 WAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 7.81383E-01

HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 5.23890E-01

1.2.2 LOSE \$50 AT BLACK JACK

UTILITY: 5.23890E-01

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 3.00000E-01

* POINTER LISTING OF 1.2.2 LOSE \$50 AT BLACK JACK

FATHER: 1.2 WIN \$200 AT ROULETTE

#1SON: NULL

R2SIBLING: 1.2.3 NO GAIN OR LOSS AT B.J.

DOWN: 1.1.3 NO GAIN/LOSS AT B.J.

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 6.25000E-01

HAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 3.76776E-01

HAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 9.33085E-01

HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 5.87327E-01

1.2.3 NO GAIN OR LOSS AT B.J.

UTILITY: 5.87327E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 1.00000E-01

* POINTER LISTING OF 1.2.3 NO GAIN OR LOSS AT B.J.

FATHER: 1.2 WIN \$200 AT ROULETTE

#1SON: NULL

R2SIBLING: NULL

DOWN: 1.2.1.1 FIND \$100 BY ACCIDENT!!

THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30

THIS IS LEAF OF TREE.

UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 1.25000E-01

HAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 3.36017E-01

HAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 6.02286E-01

HAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 3.04864E-01

1.3 NO GAIN AT ROULETTE - OR LOSS!

UTILITY: 3.04864E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 8.00000E-01 5.00000E-01

* POINTER LISTING OF 1.3 NO GAIN AT ROULETTE - OR LOSS!

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1SON: NULL

R2SIBLING: 1.4 LOSE \$50 AT ROULETTE

DOWN: 1.4 LOSE \$50 AT ROULETTE
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.50
 THIS IS LEAF OF TREE.
 UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 0.00000E+00
 WAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 1.20068E-01
 WAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 0.00000E+00
 WAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 4.80271E-02

1.4 LOSE \$50 AT ROULETTE
 UTILITY: 4.80271E-02
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 1.00000E+00
 * POINTER LISTING OF 1.4 LOSE \$50 AT ROULETTE
 FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 #1SON: NULL
 RASIBLING: NULL
 DOWN: 1.1 WIN \$50 AT BLACK JACK
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30
 THIS IS LEAF OF TREE.
 UTILITY CALCULATIONS PROCEEDING ACCORDING TO ADDITIVE FORM ...

VALUE OF 6.25000E-01
 WAS RETURNED FOR MARGINAL NUMBER 1

VALUE OF 0.00000E+00
 WAS RETURNED FOR MARGINAL NUMBER 2

VALUE OF 1.00000E+00
 WAS RETURNED FOR MARGINAL NUMBER 3

VALUE FOR UTILITY IS 4.50000E-01

2 STAY HOME - DON'T YOU DARE GO TO LA
 UTILITY: 4.50000E-01
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 0.00000E+00 0.00000E+00
 * POINTER LISTING OF 2 STAY HOME - DON'T YOU DARE GO TO LA
 FATHER: NULL
 #1SON: NULL
 RASIBLING: NULL
 DOWN: 1.1 WIN \$100 AT ROULETTE
 THIS IS AN INITIAL OPTION.
 THIS IS LEAF OF TREE.

**** THE FOLLOWING IS A LIST OF INITITIAL OPTIONS ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 UTILITY: 2.69751E-01
 NUMBER OF SONS: 4

COSTS:

-2.00000E+02 2.00000E-01 0.00000E+00

* POINTER LISTING OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
FATHER: NULL

#1@SON: 1.1 WIN \$100 AT ROULETTE

R@SIBLING: 2 STAY HOME - DON'T YOU DARE GO TO LA

DOWN: 2 STAY HOME - DON'T YOU DARE GO TO LA

THIS IS AN INITIAL OPTION.

SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA

UTILITY: 4.50000E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

* POINTER LISTING OF 2 STAY HOME - DON'T YOU DARE GO TO LA

FATHER: NULL

#1@SON: NULL

R@SIBLING: NULL

DOWN: 1.1 WIN \$100 AT ROULETTE

THIS IS AN INITIAL OPTION.

THIS IS LEAF OF TREE.

THE ABOVE LISTING INDICATES THAT THE BEST OPTION IS:

2 STAY HOME - DON'T YOU DARE GO TO LA

IT HAS AN EXPECTED UTILITY OF 4.50000E-01

**** COMMAND:END

TOTAL NUMBER OF COMMANDS PROCESSED: 4

TODAY IS MARCH 1, 1979. THE TIME IS 9:29:25:759 AM.

THIS SESSION WITH IDA HAS BEEN SUCESSFULLY COMPLETED.

4.5 EXAMPLE 2

The following example is similar to the preceeding one, except that the probability distribution of the sons of node 1 were altered. Note the effect this change had on the final outcome, as shown in the conclusions listed in Sec. 4.5.2

4.5.1 Input: JCL and RAWDAT

Note the use of the SET DEBUG command seen in the JCL listing, and the difference in the volume of output compared to that of the previous example.

EXAMPLE 2 RAWDAT INPUT FILE

1.	P4	16	3	1		
2.	M1	LINE				
3.	M2	EXPD	-0.83			
4.	M3	EXPU	0.83			
5.	U1	ADD				
6.			0.4	0.4	0.2	
7.	N1	GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?				0
8.		-200	0.200	0		
9.	N2	STAY HOME - DON'T YOU DARE GO TO LAS VEGAS!				0
10.		0	0	0		
11.	N1.1	WIN \$100 AT ROULETTE				0 0.25
12.		100	0.700	0.000		
13.	N1.2	WIN \$200 AT ROULETTE				0 0.15
14.		200.000	0.800	0.000		
15.	N1.3	NO GAIN AT ROULETTE - OR LOSS!				0 0.3
16.		0	0.800	0.500		
17.	N1.4	LOSE \$50 AT ROULETTE				0 0.3
18.		-50.000	0.200	1.000		
19.	N1.1.1	WIN \$50 AT BLACK JACK				0 0.3
20.		50	0.300	0		
21.	N1.2.1	WIN \$50 AT B.J.				0 0.4
22.		50	0.400	0		
23.	N1.1.2	LOSE \$50 AT B.J.				0 .2
24.		-50.000	0.200	0.400		
25.	N1.2.2	LOSE \$50 AT BLACK JACK				0 0.3
26.		-50	0.200	0.300		
27.	N1.1.3	NO GAIN/LOSS AT B.J.				0 0.5
28.		0	0.100	0.200		
29.	N1.2.3	NO GAIN OR LOSS AT B.J.				0 0.3
30.		0	0.100	0.100		
31.	N1.2.1.1	FIND \$100 BY ACCIDENT!!				1 .2
32.		100	0.900	0		
33.	N1.2.1.2	LOSE \$100 BY ACCIDENT				0 .2
34.		-100.000	0.300	0.500		
35.	N1.2.1.3	LOSE NOR GAIN \$ BY ACCIDENT				1 0.6
36.		0	0	0		

JCL FOR EXAMPLE 2

```

//IDA JOB (L99832,5,,2),REGION=250K,CLASS=A,PRTY=N
          99832      8J411-30190
//*FORMAT PR,DDNAME*,DEST=3800
//*
//*
//*
//STEP1 EXEC PGM=IDA
//STEPLIB DD DSN=C190.B99832.LOADERS,DISP=SHR
//*
//*
//*
//***** DATA FOR NODES OF TREE
//*
//GO,RAWDAT1 DD DSN=B99832.IDA.NODEDATA,DISP=SHR
//GO,RAWDAT2 DD DUMMY
//GO,RAWDAT3 DD DUMMY
//*
//*
//*
//***** USERS COMMANDS
//*
//GO,USER DD *
SET DEBUG '00'
BUILD 'RAWDAT1'
LIST INPUT
PROCESS '1'
LIST INPUT
END
//GO,DISPLAY DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
/*

```

4.5.2 Output

The set DEBUG command was used in this example; the size of the DISPLAY output file was greatly reduced.

WELCOME TO IDA!

AN INTERACTIVE PROGRAM
FOR DECISION ANALYSIS.

TODAY IS MARCH 2, 1979. THE TIME IS 11:45:44:697 AM.

**** COMMAND:SET DEBUG '00'

RAW DATA CARDS BEING INPUT ...
TREE CONSTRUCTION INITIATED ...

END OF RAW DATA CARDS ... 15 CARDS PROCESSED.

**** NODE LISTING ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN

UTILITY: 0.00000E+00

NUMBER OF SONS: 4

COSTS:

-2.00000E+02 2.00000E-01 0.00000E+00

THIS IS AN INITIAL OPTION.

SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

THIS IS AN INITIAL OPTION.

THIS IS LEAF OF TREE.

1.1 WIN \$100 AT ROULETTE

UTILITY: 0.00000E+00

NUMBER OF SONS: 3

COSTS:

1.00000E+02 7.00000E-01 0.00000E+00

THE PROBABILITY OF THIS NODES OCCURRENCE IS

0.25

SONS ARE PROBABILISTIC.

1.2 WIN \$200 AT ROULETTE

UTILITY: 0.00000E+00

NUMBER OF SONS: 3

COSTS:

2.00000E+02 8.00000E-01 0.00000E+00

THE PROBABILITY OF THIS NODES OCCURRENCE IS

0.15

SONS ARE PROBABILISTIC.

1.3 NO GAIN AT ROULETTE - OR LOSS!

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 8.00000E-01 5.00000E-01

THE PROBABILITY OF THIS NODES OCCURRENCE IS

0.30

THIS IS LEAF OF TREE.

1.4 LOSE \$50 AT ROULETTE

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 1.00000E+00

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.30

1.1.1 WIN \$50 AT BLACK JACK

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

5.00000E+01 3.00000E-01 0.00000E+00

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.30

1.2.1 WIN \$50 AT B.J.

UTILITY: 0.00000E+00

NUMBER OF SONS: 3

COSTS:

5.00000E+01 4.00000E-01 0.00000E+00

THE PROBABILITY OF THIS NODES OCCURRENCE IS
SONS ARE PROBABILISTIC.

0.40

1.1.2 LOSE \$50 AT B.J.

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 4.00000E-01

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.20

1.2.2 LOSE \$50 AT BLACK JACK

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 3.00000E-01

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.30

1.1.3 NO GAIN/LOSS AT B.J.

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 2.00000E-01

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.50

1.2.3 NO GAIN OR LOSS AT B.J.

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 1.00000E-01

THE PROBABILITY OF THIS NODES OCCURRENCE IS
THIS IS LEAF OF TREE.

0.30

1.2.1.1 FIND \$100 BY ACCIDENT!!

UTILITY: 0.00000E+00

NUMBER OF SONS: 0

COSTS:

1.00000E+02 9.00000E-01 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20
 THIS IS LEAF OF TREE.

1.2.1.2 LOSE \$100 BY ACCIDENT
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 -1.00000E+02 3.00000E-01 5.00000E-01
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20
 THIS IS LEAF OF TREE.

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT
 UTILITY: 0.00000E+00
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 0.00000E+00 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.60
 THIS IS LEAF OF TREE.

**** NODE LISTING COMPLETE ****

**** THE FOLLOWING IS A LIST OF INITITIAL OPTIONS ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 UTILITY: 3.19302E-01
 NUMBER OF SONS: 4
 COSTS:
 -2.00000E+02 2.00000E-01 0.00000E+00
 THIS IS AN INITIAL OPTION.
 SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA
 UTILITY: 4.50000E-01
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 0.00000E+00 0.00000E+00
 THIS IS AN INITIAL OPTION.
 THIS IS LEAF OF TREE.

THE ABOVE LISTING INDICATES THAT THE BEST OPTION IS:
 2 STAY HOME - DON'T YOU DARE GO TO LA
 IT HAS AN EXPECTED UTILITY OF 4.50000E-01

**** NODE LISTING ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 UTILITY: 3.19302E-01
 NUMBER OF SONS: 4
 COSTS:

-2.00000E+02 2.00000E-01 0.00000E+00
 THIS IS AN INITIAL OPTION.
 SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA

UTILITY: 4.50000E-01
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 0.00000E+00 0.00000E+00
 THIS IS AN INITIAL OPTION.
 THIS IS LEAF OF TREE.

1.1 WIN \$100 AT ROULETTE

UTILITY: 4.76489E-01
 NUMBER OF SONS: 3
 COSTS:
 1.00000E+02 7.00000E-01 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.25
 SONS ARE PROBABILISTIC.

1.2 WIN \$200 AT ROULETTE

UTILITY: 6.28748E-01
 NUMBER OF SONS: 3
 COSTS:
 2.00000E+02 8.00000E-01 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.15
 SONS ARE PROBABILISTIC.

1.3 NO GAIN AT ROULETTE - OR LOSS!

UTILITY: 3.04864E-01
 NUMBER OF SONS: 0
 COSTS:
 0.00000E+00 8.00000E-01 5.00000E-01
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
 THIS IS LEAF OF TREE.

1.4 LOSE \$50 AT ROULETTE

UTILITY: 4.80271E-02
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 1.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
 THIS IS LEAF OF TREE.

1.1.1 WIN \$50 AT BLACK JACK

UTILITY: 5.67613E-01
 NUMBER OF SONS: 0
 COSTS:
 5.00000E+01 3.00000E-01 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
 THIS IS LEAF OF TREE.

1.2.1 WIN \$50 AT B.J.

UTILITY: 7.38457E-01
 NUMBER OF SONS: 3
 COSTS:
 5.00000E+01 4.00000E-01 0.00000E+00
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.40
 SONS ARE PROBABILISTIC.

1.1.2 LOSE \$50 AT B.J.

UTILITY: 3.89820E-01

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 4.00000E-01

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.20

1.2.2 LOSE \$50 AT BLACK JACK

UTILITY: 5.23890E-01

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 3.00000E-01

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.30

1.1.3 NO GAIN/LOSS AT B.J.

UTILITY: 4.56483E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 2.00000E-01

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.50

1.2.3 NO GAIN OR LOSS AT B.J.

UTILITY: 5.87327E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 1.00000E-01

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.30

1.2.1.1 FIND \$100 BY ACCIDENT!!

UTILITY: 1.00000E+00

NUMBER OF SONS: 0

COSTS:

1.00000E+02 9.00000E-01 0.90000E+00

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.20

1.2.1.2 LOSE \$100 BY ACCIDENT

UTILITY: 5.82370E-01

NUMBER OF SONS: 0

COSTS:

-1.00000E+02 3.00000E-01 5.00000E-01

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.20

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

UTILITY: 7.03305E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

THE PROBABILITY OF THIS NODES OCCURRANCE IS
THIS IS LEAF OF TREE.

0.60

**** NODE LISTING COMPLETE ****

4.6 EXAMPLE 3

The error discovered when this example was run has since been corrected, but the user should note how IDAP handles such errors.

Note the multiple use of the SET DEBUG command, utility function input sequences, and PROCESS commands.

4.6.1 Input: JCL and RAWDAT

Five utility functions were defined for this example, with the intention that a crude form of sensitivity analysis would be performed. Only the K_i in each utility function were altered.

EXAMPLE 3 RAWDAT INPUT FILE

1.	P4	16	3	5	
2.	M1	LINE			
3.	M2	EXPD	-0.83		
4.	M3	EXPU	0.83		
5.	U1	ADD			
6.			0.334	0.333	0.333
7.	U2	ADD			
8.			0.4	0.4	0.2
9.	U3	ADD			
10.			0.3	0.5	0.2
11.	U4	ADD			
12.			0.2	0.5	0.3
13.	U5	ADD			
14.			0.1	0.2	0.7
15.	N1	GO TO LAS VEGAS! HAVE A LOT OF FUN ... MAYBE!?			0
16.			-200	0.200	0
17.	N2	STAY HOME - DON'T YOU DARE GO TO LAS VEGAS!			0
18.			0	0	0
19.	N1.1	WIN \$100 AT ROULETTE			0 0.25
20.			100	0.700	0.000
21.	N1.2	WIN \$200 AT ROULETTE			0 0.15
22.			200.000	0.800	0.000
23.	N1.3	NO GAIN AT ROULETTE - OR LOSS!			0 0.3
24.			0	0.800	0.500
25.	N1.4	LOSE \$50 AT ROULETTE			0 0.3
26.			-50.000	0.200	1.000
27.	N1.1.1	WIN \$50 AT BLACK JACK			0 0.3
28.			50	0.300	0
29.	N1.2.1	WIN \$50 AT B.J.			0 0.4
30.			50	0.400	0
31.	N1.1.2	LOSE \$50 AT B.J.			0 .2
32.			-50.000	0.200	0.400
33.	N1.2.2	LOSE \$50 AT BLACK JACK			0 0.3
34.			-50	0.200	0.300
35.	N1.1.3	NO GAIN/LOSS AT B.J.			0 0.5
36.			0	0.100	0.200
37.	N1.2.3	NO GAIN OR LOSS AT B.J.			0 0.3
38.			0	0.100	0.100
39.	N1.2.1.1	FIND \$100 BY ACCIDENT!!			1 .2
40.			100	0.900	0
41.	N1.2.1.2	LOSE \$100 BY ACCIDENT			0 .2
42.			-100.000	0.300	0.500
43.	N1.2.1.3	LOSE NOR GAIN \$ BY ACCIDENT			1 0.6
44.			0	0	0

EXAMPLE 3 JCL

```

1.      //IDAE3A JOB (L99832,5,,2),REGION=250K,CLASS=A,PTY=N
2.      99832      8J411-38190
3.2     //MAIN ORG=RADS14
4.      //*
5.      //*
6.      //*
7.      //STEP1 EXEC PGM=IDA
8.      //STEPLIB DD DSN=C190.B99832.LOADERS,DISP=SHR
9.      //*
10.     //*
11.     //*
12.     //***** DATA FOR NODES OF TREE
13.     //*
14.     //GO.RAWDAT1 DD DSN=B99832.IDA.NODEDATA,DISP=SHR
15.     //GO.RAWDAT2 DD DUMMY
16.     //GO.RAWDAT3 DD DUMMY
17.     //*
18.     //*
19.     //*
20.     //***** USERS COMMANDS
21.     //*
22.     //GO.USER DD *
23.     SET DEBUG '00'
24.     BUILD 'RAWDAT1'
25.     PROCESS '1'
25.1    SET DEBUG '11'
26.     LIST INPUT
27.     PROCESS '2'
28.     LIST INPUT
29.     PROCESS '3'
30.     LIST INPUT
31.     PROCESS '4'
32.     LIST INPUT
33.     PROCESS '5'
34.     LIST INPUT
35.     END
36.     //GO.DISPLAY DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
37.     /*

```

4.6.2 Output

Note the final utility values of the initial options of this decision tree; they are different, as expected, from the ones on the previous two examples.

Note also the effect of the use of the SET DEBUG command; the variable DEBUG was maximized intentionally in order to study the error encountered. Refer to the next section for a discussion of IDAP's error messages.

The error handling seen in this printout is typical for IDAP.

```

      WELCOME TO IDA!

      AN INTERACTIVE PROGRAM
      FOR DECISION ANALYSIS.

TODAY IS MARCH  6, 1979.  THE TIME IS  2:32:25:435 PM.

**** COMMAND:SET DEBUG '00'

RAW DATA CARDS BEING INPUT ...
TREE CONSTRUCTION INITIATED ...

END OF RAW DATA CARDS ...      15 CARDS PROCESSED.

**** THE FOLLOWING IS A LIST OF INITIAL OPTIONS ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
  UTILITY:  3.54358E-01
  NUMBER OF SONS:      4
  COSTS:
    -2.00000E+02  2.00000E-01  0.00000E+00
  THIS IS AN INITIAL OPTION.
  SONS ARE PROBABILISTIC.

2 STAY HOME - DON'T YOU DARE GO TO LA
  UTILITY:  5.41750E-01
  NUMBER OF SONS:      0
  COSTS:
    0.00000E+00  0.00000E+00  0.00000E+00
  THIS IS AN INITIAL OPTION.
  THIS IS LEAF OF TREE.

THE ABOVE LISTING INDICATES THAT THE BEST OPTION IS:
2 STAY HOME - DON'T YOU DARE GO TO LA
IT HAS AN EXPECTED UTILITY OF  5.41750E-01

**** COMMAND:LIST INPUT

**** NODE LISTING ****

1 GO TO LAS VEGAS! HAVE A LOT OF FUN
  UTILITY:  3.54358E-01
  NUMBER OF SONS:      4
  COSTS:
    -2.00000E+02  2.00000E-01  0.00000E+00
  * POINTER LISTING OF 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
    FATHER: NULL
    $1$SON: 1.1 WIN $100 AT ROULETTE
    $2$SIBLING: 2 STAY HOME - DON'T YOU DARE GO TO LA
    DOWN: 2 STAY HOME - DON'T YOU DARE GO TO LA
  THIS IS AN INITIAL OPTION.
```

SONS ARE PROBABILISTIC.

2. STAY HOME - DON'T YOU DARE GO TO LA

UTILITY: 5.41750E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

* POINTER LISTING OF 2 STAY HOME - DON'T YOU DARE GO TO LA

FATHER: NULL

#1@SON: NULL

R@SIBLING: NULL

DOWN: 1.1 WIN \$100 AT ROULETTE

THIS IS AN INITIAL OPTION.

THIS IS LEAF OF TREE.

1.1 WIN \$100 AT ROULETTE

UTILITY: 5.41803E-01

NUMBER OF SONS: 3

COSTS:

1.00000E+02 7.00000E-01 0.00000E+00

* POINTER LISTING OF 1.1 WIN \$100 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1@SON: 1.1.1 WIN \$50 AT BLACK JACK

R@SIBLING: 1.2 WIN \$200 AT ROULETTE

DOWN: 1.2 WIN \$200 AT ROULETTE

THE PROBABILITY OF THIS NODES OCCURRANCE IS

0.25

SONS ARE PROBABILISTIC.

1.2 WIN \$200 AT ROULETTE

UTILITY: 6.71010E-01

NUMBER OF SONS: 3

COSTS:

2.00000E+02 8.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2 WIN \$200 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1@SON: 1.2.1 WIN \$50 AT B.J.

R@SIBLING: 1.3 NO GAIN AT ROULETTE - OR LOSS!

DOWN: 1.3 NO GAIN AT ROULETTE - OR LOSS!

THE PROBABILITY OF THIS NODES OCCURRANCE IS

0.15

SONS ARE PROBABILISTIC.

1.3 NO GAIN AT ROULETTE - OR LOSS!

UTILITY: 3.54205E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 8.00000E-01 5.00000E-01

* POINTER LISTING OF 1.3 NO GAIN AT ROULETTE - OR LOSS!

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN

#1@SON: NULL

R@SIBLING: 1.4 LOSE \$50 AT ROULETTE

DOWN: 1.4 LOSE \$50 AT ROULETTE

THE PROBABILITY OF THIS NODES OCCURRANCE IS

0.30

THIS IS LEAF OF TREE.

1.4 LOSE \$50 AT ROULETTE

UTILITY: 3.99825E-02

NUMBER OF SONS: 0

COSTS:

-5.00000E+01 2.00000E-01 1.00000E+00

* POINTER LISTING OF 1.4 LOSE \$50 AT ROULETTE

FATHER: 1 GO TO LAS VEGAS! HAVE A LOT OF FUN
 #1SON: NULL
 RASIBLING: NULL
 DOWN: 1.1.1 WIN \$50 AT BLACK JACK
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30
 THIS IS LEAF OF TREE.

1.1.1 WIN \$50 AT BLACK JACK
 UTILITY: 6.39538E-01
 NUMBER OF SONS: 0
 COSTS:
 5.00000E+01 3.00000E-01 0.00000E+00
 * POINTER LISTING OF 1.1.1 WIN \$50 AT BLACK JACK
 FATHER: 1.1 WIN \$100 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.1.2 LOSE \$50 AT B.J.
 DOWN: 1.2.1 WIN \$50 AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30
 THIS IS LEAF OF TREE.

1.2.1 WIN \$50 AT B.J.
 UTILITY: 7.68771E-01
 NUMBER OF SONS: 3
 COSTS:
 5.00000E+01 4.00000E-01 0.00000E+00
 * POINTER LISTING OF 1.2.1 WIN \$50 AT B.J.
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1SON: 1.2.1.1 FIND \$100 BY ACCIDENT!!
 RASIBLING: 1.2.2 LOSE \$50 AT BLACK JACK
 DOWN: 1.1.2 LOSE \$50 AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.40
 SONS ARE PROBABILISTIC.

1.1.2 LOSE \$50 AT B.J.
 UTILITY: 4.40584E-01
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 4.00000E-01
 * POINTER LISTING OF 1.1.2 LOSE \$50 AT B.J.
 FATHER: 1.1 WIN \$100 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.1.3 NO GAIN/LOSS AT B.J.
 DOWN: 1.2.2 LOSE \$50 AT BLACK JACK
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.20
 THIS IS LEAF OF TREE.

1.2.2 LOSE \$50 AT BLACK JACK
 UTILITY: 5.66738E-01
 NUMBER OF SONS: 0
 COSTS:
 -5.00000E+01 2.00000E-01 3.00000E-01
 * POINTER LISTING OF 1.2.2 LOSE \$50 AT BLACK JACK
 FATHER: 1.2 WIN \$200 AT ROULETTE
 #1SON: NULL
 RASIBLING: 1.2.3 NO GAIN OR LOSS AT B.J.
 DOWN: 1.1.3 NO GAIN/LOSS AT B.J.
 THE PROBABILITY OF THIS NODES OCCURRANCE IS 0.30
 THIS IS LEAF OF TREE.

1.1.3 NO GAIN/LOSS AT B.J.

UTILITY: 5.23650E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 2.00000E-01

* POINTER LISTING OF 1.1.3 NO GAIN/LOSS AT B.J.

FATHER: 1.1 WIN \$100 AT ROULETTE

#1@SON: NULL

R@SIBLING: 1.2.1 WIN \$50 AT B.J.

DOWN: 1.2.3 NO GAIN OR LOSS AT B.J.

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.50
THIS IS LEAF OF TREE.

1.2.3 NO GAIN OR LOSS AT B.J.

UTILITY: 6.44934E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 1.00000E-01 1.00000E-01

* POINTER LISTING OF 1.2.3 NO GAIN OR LOSS AT B.J.

FATHER: 1.2 WIN \$200 AT ROULETTE

#1@SON: NULL

R@SIBLING: NULL

DOWN: 1.2.1.1 FIND \$100 BY ACCIDENT!!

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.30
THIS IS LEAF OF TREE.

1.2.1.1 FIND \$100 BY ACCIDENT!!

UTILITY: 1.00000E+00

NUMBER OF SONS: 0

COSTS:

1.00000E+02 9.00000E-01 0.00000E+00

* POINTER LISTING OF 1.2.1.1 FIND \$100 BY ACCIDENT!!

FATHER: 1.2.1 WIN \$50 AT B.J.

#1@SON: NULL

R@SIBLING: 1.2.1.2 LOSE \$100 BY ACCIDENT

DOWN: 1.2.1.2 LOSE \$100 BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.20
THIS IS LEAF OF TREE.

1.2.1.2 LOSE \$100 BY ACCIDENT

UTILITY: 5.85604E-01

NUMBER OF SONS: 0

COSTS:

-1.00000E+02 3.00000E-01 5.00000E-01

* POINTER LISTING OF 1.2.1.2 LOSE \$100 BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1@SON: NULL

R@SIBLING: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

DOWN: 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.20
THIS IS LEAF OF TREE.

1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

UTILITY: 7.52751E-01

NUMBER OF SONS: 0

COSTS:

0.00000E+00 0.00000E+00 0.00000E+00

* POINTER LISTING OF 1.2.1.3 LOSE NOR GAIN \$ BY ACCIDENT

FATHER: 1.2.1 WIN \$50 AT B.J.

#1@SON: NULL

R@SIBLING: NULL

DOWN: NULL
 THE PROBABILITY OF THIS NODES OCCURRENCE IS 0.60
 THIS IS LEAF OF TREE.

**** NODE LISTING COMPLETE ****

```
**** COMMAND:PROCESS '2'
COUNTING IN LEVEL 4
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 3
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 2
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 1
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  #@LEAVE = 22 #OPT@NODES = 2
COUNTING IN LEVEL 4
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 3
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 2
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
COUNTING IN LEVEL 1
  EXAMINING NEXT NODE.
  EXAMINING NEXT NODE.
```

PL/1 ERROR

**** ERROR NUMBER 20.100
 ONCODE = 8095

TOTAL NUMBER OF COMMANDS PROCESSED: 6

5 A PROGRAMMER'S GUIDE TO THE IDAP

5.1 GENERAL BACKGROUND

Figure 5.1 illustrates the flow, at the system analysis level, of the entire IDAP. The IDAIN program is used to format input for the IDAP PL/1 program, which the user may execute a number of times. Note that, for all procedures within IDAP, complete source code listings and illustrations of their internal modular flow are provided. Both of these procedures will be discussed, but emphasis of this document is placed on the larger IDAP PL/1 program.

Fig. 5.1 IDAP Procedure Flow.- System Analysis Level

The IDAIN procedure is written in WYLBUR exec language, which is easily learned and simple to write. Its interactive programming capabilities run quickly and do not tie up many resources, as do TSO procedures. The simple I/O with the large amounts of time spent waiting for the user's responses during input sessions made WYLBUR a logical choice for an interactive procedure to format input data.

The IDAP program is written in PL/1. The advantages of using PL/1 rather than another language such as FORTRAN or APL include the following.

- PL/1 is more readable than many languages. Frequent modifications to the program are more efficient if the programmer understands quickly what is read in the source code.
- PL/1 allows the programmer to structure the source code. Modifications are more easily made when the source code is modular.
- PL/1 allows many types of I/O.
- Without PL/1's pointers or data structure facilities, IDAP's source code would be more complex, and execution time would probably increase.
- PL/1 is supported by IBM on their larger machines; IDAP's storage requirements are relatively large.

Currently, IDAIN runs through WYLBUR, which is available to the user at ANL. The IDAP program, through batch mode, may be executed on either an IBM 370/195 or an IBM 3033 computer. Execution statistics of the IDAP program are presented in Sec. 5.3 of this document.

Before the procedures of the IDAP program are presented, the dynamic PL/1 structure used to implement the user's decision tree will be examined. The heart of this implementation is based on the following PL/1 structure.

```

1 NODE          BASED,
  2 #I          REAL FIXED BINARY(15),
  2 FATHER      POINTER,
  2 #1SON       POINTER,
  2 RSIBLING    POINTER,
  2 #SONS       REAL FIXED BINARY(15),
  2 OPTFLAG     BIT(1),
  2 ID          CHAR(65),
  2 COST(#ATTRI REFER(NODE.#I))
                REAL FLOAT BINARY(21),
  2 PROB        REAL FLOAT BINARY(21),
  2 UTILITY     REAL FLOAT BINARY(21),
  2 DOWN        POINTER

```

In the above structure, NODE is the general name of the based structure pointed to by a PL/1 pointer variable. Each node contains all data that completely define a fork in the user's decision tree. By linking the various

nodes together through the FATHER, #1SON, RSIBLING, and DOWN pointer variables, the user's entire decision tree may be dynamically constructed. Pointers allow for quick traverses of the structure, and traverses are one of the main logical processes in the IDAP procedure.

The variables in the above structure are briefly described below.

#I contains the number of attributes to be used in the IDA procedure.

FATHER contains the pointer variable to the "address" of the father of the node in question.
(Refer to Sec. 3 for the familial terminology used in this document.)

#1SON points to the first son of the node in question.

RSIBLING points to the node that is adjacent to the node in question, where the two are on the same level.

#SONS is a variable containing the exact number of sons of which the node in question is a father.

OPTFLAG is a bit switch; it is on if the node in question is an option node, and is off if the node in question is a probability node.

ID is the user-supplied name of the node in question. The string in this ID, up to the first blank space, is the unique label used by IDAP for identification of the node in question.

COST is an array whose dimension equals the number of attributes considered in the IDAP procedure. #ATTRI is another variable in the program, and its value is stored in #I in the data structure for the node in question.
COST contains the levels of attributes as input by the user.

PROB is the probability (between 0 and 1) of the occurrence of the event represented by the node in question. This value is used by IDAP only if the father of the node in question is a probability node.

UTILITY is the utility value of the node in question. It is calculated from the terminal nodes of its subtree upward to the node in question.

DOWN is a pointer assigned by IDAP when the nodes are initially read in from the user's input. It is necessary only because it serves as the initial link that is used in traverses between nodes while the tree is still under construction.

5.2 IDAP'S PROCEDURES

The IDAP PL/1 program is broken down into a number of external procedures, each of which will be discussed below. Although it is assumed the reader is familiar with PL/1 list processing, the basic logic behind many of the concepts implemented in IDAP will be discussed.

5.2.1 Main Procedure

The main procedure, that is the code where execution of the program is initiated, serves to initialize the entire program. For the declarations of variables, as shown in the main procedure's source code listings provided at the end of this section, note the following explanations.

BATCH: A bit switch used by IDAP to distinguish between sessions that are executed in batch mode or interactive mode. At the present time, BATCH=1.

COMMAND: The basic character string that is used to transfer data from the user to IDAP.

DEBUG: A variable used to indicate the level of debugging desired by the user/ programmer. By altering its value, the user may control the amount of output generated by IDAP, although there is a minimum amount of output that IDAP will always produce. If DEBUG equals '11'B, IDAP will generate the maximum amount of output, while assigning it a value of '00'B will produce the least.

FILENAM: A file variable used as the name of the user's input file, (either RAWDAT1, RAWDAT2, or RAWDAT3), that contains data defining decision tree nodes and functions.

ONCODE: A built-in PL/1 function that produces the "error number" when such an event occurs.

PMARG: An array containing variables pointing to the based structure for the various marginal functions defined by the user.

PUTIL: An array containing pointers to the based structures for the various utility functions (MUF) defined by the user.

PLACE: A variable that may be used to indicate the place in the program where an error occurred. When used together with the ONCODE message, this may supply the user/programmer with the means of correcting the error. This document provides a list of error messages that may be produced by IDAP.

PROFICI: A variable that the user may change during the execution of IDAP. It serves to indicate the proficiency of the user. This variable is not used much in the current version of IDAP.

RAWDATi files: The input files for the user. Their names, as explained above, are stored in FILENAM when they are being used by IDAP for input.

SYSPRINT: The standard PL/1 output file. Under normal conditions, IDAP will produce no output on SYSPRINT.

GEORGE: A time function that formats and returns the current date and time variables passed to it.

LEVEL: A controlled array that points to the first node in each level of the user's decision tree.

Fig. 5.2 The MAIN Procedure

MARGIN: A structure containing the parameters of a marginal function as defined by the user.

GENERAL: A dynamic structure containing the parameters of a user-defined marginal function of the general form.

The main procedure prints out the time and date and then calls GETCOM, which serves as a branching point for the program. From GETCOM, depending on the user's input, control passes to the procedure that will process this command. The modular flow of this main procedure is illustrated in Fig. 5.2.

Following the listing of the main procedure is the source code of the ERROR procedure. Note that use of the "* PROCESS" card is necessary when not using the link editor or IDAP's load modules before execution*. This error procedure is not well developed at present; improvements will be made in future versions of IDAP.

*See Sec. 4.3 for an explanation of the JCL needed to run IDAP at ANL.

Fig. 5.3 The GETCOM Module -
Within MAIN

Fig. 5.4 The LIST Module -
Within MAIN

Fig. 5.5 The LISTMAR Module -
Within MAIN

Fig. 5.6 The LISTUTI Module -
Within MAIN

Fig. 5.7 The PREPRO
Module -
Within MAIN

Fig. 5.8 The ERROR
Procedure

Fig. 5.9 The PAR@SET Module - Within MAIN

```

/*****
IDA: PROCEDURE OPTIONS(MAIN);
*****/

```

```

/*      AN INTERACTIVE DECISION ANALYSIS PROGRAM
        WRITTEN BY MARK J. JUSKO
        ARGONNE NATIONAL LABORATORY

```

```

        LAST UPDATE: 9-22-80

```

```

*/

```

```

DECLARE

```

```

#@COMS      REAL FIXED BINARY(15) INIT(0) EXTERNAL,
#@ATTRI     REAL FIXED BINARY(15) INIT(1) EXTERNAL,
#@LEVEL     REAL FIXED BINARY(15) INIT(12) EXTERNAL,
#@NODES     REAL FIXED BINARY(15) INIT(10) EXTERNAL,
#UFS        REAL FIXED BINARY(15) EXTERNAL,
BATCH       BIT(1) INIT('1'B) EXTERNAL,
CARD@IM     CHAR(80) INIT(' ') EXTERNAL,
COMMAND     CHAR(80) EXTERNAL,
DATE        BUILTIN,
DEBUG       BIT(2) INIT('11'B) EXTERNAL,
DISPLAY     STREAM OUTPUT PRINT EXTERNAL,
FILENAM     FILE VARIABLE EXTERNAL,
HEADER      POINTER EXTERNAL,
ID@LENG     REAL FIXED BINARY(15) INIT(65) EXTERNAL,
INDEX       BUILTIN,
MAX(@ATTRI) REAL FLOAT BINARY(21) CTL EXTERNAL,
MIN(@ATTRI) REAL FLOAT BINARY(21) CTL EXTERNAL,
NO@INCR     REAL FIXED BINARY(15) EXTERNAL,
NULL        BUILTIN,
NULLO       POINTER EXTERNAL,
ONCODE      BUILTIN,
P@MARG(@ATTRI) POINTER CTL EXTERNAL,
P@UTIL(@UFS) POINTER CTL EXTERNAL,
PLACE       REAL FIXED DECIMAL(7,3) EXTERNAL,
PROFICI     BIT(3) INIT('000'B) EXTERNAL,
P1          POINTER EXTERNAL,
RAWDAT1     STREAM INPUT EXTERNAL,
RAWDAT2     STREAM INPUT EXTERNAL,
RAWDAT3     STREAM INPUT EXTERNAL,
SENSE       BIT(1) EXTERNAL INIT('0'B),
SHO@CST     BIT(1) INIT('1'B) EXTERNAL,
SUBSTR      BUILTIN,
SYSPRINT    STREAM OUTPUT PRINT EXTERNAL,
THE@END     LABEL EXTERNAL,
TIME        BUILTIN,
USER        STREAM INPUT EXTERNAL,
UTILNO      REAL FIXED BINARY(15) EXTERNAL,

```

```

ALTER      EXTERNAL ENTRY,
BUILD      EXTERNAL ENTRY,
CALCK      EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),

```

```

ERROR          EXTERNAL ENTRY,
GEORGE         EXTERNAL ENTRY RETURNS(CHAR(80)),
LODOWN        EXTERNAL ENTRY,
LONODE        EXTERNAL ENTRY,
LOPOINT       EXTERNAL ENTRY,
LISTM         EXTERNAL ENTRY,
LISTU         EXTERNAL ENTRY,
LOCATE        EXTERNAL ENTRY RETURNS(POINTER),
PEEL          EXTERNAL ENTRY RETURNS(CHAR(80)),
POINT         EXTERNAL ENTRY,
PRINT         EXTERNAL ENTRY,
PROCESS       EXTERNAL ENTRY,
READRAW       EXTERNAL ENTRY,
SENSIT        EXTERNAL ENTRY,
STRIP         EXTERNAL ENTRY RETURNS(FIXED BINARY(15)),
SUBTOL        EXTERNAL ENTRY,
UTILOF        EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),

```

```

LEVEL(#@LEVEL)  POINTER CTL EXTERNAL,
1 MARGIN        BASED,
    2 NAME      CHAR(4),
    2 PARM(3)   REAL FLOAT BINARY(21),
1 GENERAL       BASED,
    2 NAME      CHAR(4),
    2 #@INCR    REAL FIXED BINARY(15),
    2 UP@LIM(NO@INCR REFER(GENERAL.#@INCR))
                REAL FLOAT BINARY(21),
    2 VALUE(NO@INCR REFER(GENERAL.#@INCR))
                REAL FLOAT BINARY(21),
1 NODE          BASED,
    2 #@I       REAL FIXED BINARY(15),
    2 FATHER    POINTER,
    2 #1@SON    POINTER,
    2 R@SIBLING POINTER, /* SAME LEV, NOT ONLY BRO.*/
    2 #@SONS    REAL FIXED BINARY(15),
    2 OPT@FLAG  BIT(1),
    2 ID        CHAR(65),
    2 COST(#@ATTRI REFER(NODE.#@I))
                REAL FLOAT BINARY(21),
    2 PROB      REAL FLOAT BINARY(21),
    2 UTILITY   REAL FLOAT BINARY(21),
    2 DOWN      POINTER,
1 UTILITY       BASED,
    2 NAME      CHAR(3),
    2 #@I       REAL FIXED BINARY(15),
    2 PARM(#@ATTRI REFER(UTILITY.#@I))
                REAL FLOAT BINARY(21),
    2 K         REAL FLOAT BINARY(21),
CST@CRD        CHAR(80) CTL EXTERNAL;

```

```

PUT FILE(DISPLAY) EDIT
('WELCOME TO IDA!',

```

```

      'AN INTERACTIVE PROGRAM',
      'FOR DECISION ANALYSIS.',
      GEORGE(TIME,DATE,'  '))
      (COL(24),A,
      SKIP,2 (SKIP,COL(20),A),
      SKIP(3),A(80));
PUT FILE(DISPLAY) EDIT
  ('BEGIN ENTERING COMMANDS ...')
  (SKIP(5),A);
NULLO = NULL;
THE@END = QUIT;

ON ERROR BEGIN;
      PUT FILE(DISPLAY) EDIT
      ('      PL/1 ERROR')
      (SKIP(3),A);
      CALL ERROR;
      END;

CALL GETCOM;

/*****
GETCOM: PROCEDURE;
*****/

/* INITIATES THE RESPONSES TO COMMAND INPUT. */

ON ENDFILE(USER) GO TO THE@END;

GET: DO WHILE('1'B);
  IF BATCH
    THEN;
    ELSE PUT FILE(DISPLAY) EDIT
      ('COMMAND:')(COL(1),A);
  GET FILE(USER) EDIT
    (COMMAND)(COL(1),A(80));
  IF BATCH
    THEN #@COMS = #@COMS + 1;
    ELSE PUT FILE(DISPLAY) EDIT
      ('COMMAND ACCEPTED.')(COL(1),A);
  IF INDEX(COMMAND,'OUTPUT') = 0 & SUBSTR(COMMAND,1,1)
    THEN PUT FILE(DISPLAY) EDIT
      ('**** COMMAND:',COMMAND)
      (SKIP(2),A,A(80));
  ELSE;

```

```

PLACE = .5;
SELECT;
    WHEN(SUBSTR(COMMAND,1,1)='*');
    WHEN(INDEX(COMMAND,'OUTPUT') > 0) CALL OUTPUTT;
    WHEN(INDEX(COMMAND,'SENS') > 0) CALL SENSIT;
    WHEN(INDEX(COMMAND,'AVER') > 0) CALL PREPRO;
    WHEN(INDEX(COMMAND,'LIST ') > 0) CALL LIST;
    WHEN(INDEX(COMMAND,'BUILD ') > 0) CALL BUILD;
    WHEN(INDEX(COMMAND,'PROCESS') > 0) CALL PREPRO;
    WHEN(INDEX(COMMAND,'SET ') > 0) CALL PAR@SET;
    WHEN(INDEX(COMMAND,'END ') > 0) GO TO THE@END;
    WHEN(INDEX(COMMAND,'ALTER ') > 0) CALL ALTER;
    OTHERWISE CALL ERROR;
END;
END GET;

END GETCOM;

```

```

/*****/
OUTPUTT: PROCEDURE;
/*****/

```

```

/* PRINTS COMMENTS DESIRED BY THE USER */

```

```

DECLARE COMMENT CHAR(80);

COMMENT = PEEL(COMMAND);
IF INDEX(COMMAND,' PAGE ') > 0
    THEN PUT FILE(DISPLAY) EDIT
        (COMMENT)(PAGE,A(80));
    ELSE PUT FILE(DISPLAY) EDIT
        (COMMENT)(SKIP(2),A(80));

```

```

END OUTPUTT;

```

```

/*****/
LIST: PROCEDURE;
/*****/

```

```

PLACE = 0.51;
SELECT;
    WHEN(INDEX(COMMAND,' INPUT') > 0) CALL LODOWN;
    WHEN(INDEX(COMMAND,' SUBTREE') > 0) CALL SUBTOL;
    WHEN(INDEX(COMMAND,' NODE') > 0) CALL LONODE;
    WHEN(INDEX(COMMAND,' MARG') > 0) CALL LISTMAR;
    WHEN(INDEX(COMMAND,' UTIL') > 0) CALL LISTUTI;

```



```

        OTHERWISE CALL ERROR;
END;

END LIST;

```

```

/*****/
LISTMAR: PROCEDURE;
/*****/

```

```

/* INVOKES LISTM(NUM). NUM FROM COMMAND. */

```

```

DECLARE NUM          REAL FIXED BINARY(15);

```

```

NUM = PEEL(COMMAND);
IF NUM > #@ATTRI | NUM < 1
    THEN DO;
        PLACE = 0.7;
        CALL ERROR;
    END;
ELSE CALL LISTM(NUM);
END LISTMAR;

```

```

/*****/
LISTUTI: PROCEDURE;
/*****/

```

```

/* INVOKES LISTU(NUM). NUM FROM COMMAND. */

```

```

DECLARE NUM          REAL FIXED BINARY(15);

```

```

NUM = PEEL(COMMAND);
IF NUM > #@ATTRI | NUM < 1
    THEN DO;
        PLACE = 0.8;
        CALL ERROR;
    END;
ELSE CALL LISTU(NUM);
END LISTUTI;

```

```

/*****/
PREPRO: PROCEDURE;
/*****/

```

```

/* UTILNO = THE UTILITY FUNCTION TO BE USED IN PROCESSING */

```

```

DECLARE PASS      CHAR(1) INIT('A');

```

```

IF INDEX(COMMAND, 'PROCESS') > 0

```

```

THEN DO;
    PASS = 'P';
    UTILNO = PEEL(COMMAND);
    PLACE = 0.6;
    IF UTILNO > #UFS | UTILNO < 1
        THEN CALL ERROR;
    ELSE;
        END;
    ELSE;
        CALL PROCESS(PASS);
END PREPRO;

```

```

/*****
PAR@SET: PROCEDURE;
*****/

```

```

DECLARE  A          CHAR(80),
        NEWCOM      CHAR(80),
        MAXVAL      FLOAT BINARY(21),
        MINVAL      FLOAT BINARY(21),
        MAXLOC      FIXED BINARY(15),
        MINLOC      FIXED BINARY(15),
        I          FIXED BINARY(15);

```

```

PLACE = 0.61;
A = PEEL(COMMAND);
IF A = ' ' & INDEX(COMMAND, ' LEVEL') = 0
    THEN DO;
        CALL ERROR;
        RETURN;
    END;
    ELSE;
        PLACE = 0.62;
        SELECT;
            WHEN (INDEX(COMMAND, ' DEBUG') > 0) DEBUG = A;
            WHEN (INDEX(COMMAND, ' PROF') > 0) PROFICI = A;
            WHEN (INDEX(COMMAND, ' SHOWC') > 0) SHO@CST = A;
            OTHERWISE CALL ERROR;
        END;

```

```

END PAR@SET;

```

```

QUITS:

```

```

    IF BATCH = '1'B

```

```

      THEN PUT FILE(DISPLAY) EDIT
        ('TOTAL NUMBER OF COMMANDS PROCESSED: ',#@COMS)
        (SKIP(3),A,F(5,0));
    ELSE;
    PUT FILE(DISPLAY) EDIT
      (GEORGE(TIME,DATE,' '),
      'THIS SESSION WITH IDA HAS BEEN ',
      'SUCESSFULLY COMPLETED.')
      (SKIP(5),A(80),
      SKIP(2),A,A);
END IDA;

* PROCESS;

/*****
ERROR: PROCEDURE;
*****/

DECLARE  DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
         ONCODE       BUILTIN,
         THE@END      LABEL EXTERNAL,
         PLACE        REAL FIXED DECIMAL(7,3) EXTERNAL,
         PROFICI      BIT(3) EXTERNAL;

PUT FILE(DISPLAY) EDIT
  ('**** ERROR NUMBER ',PLACE,'ONCODE = ',ONCODE)
  (SKIP(2),A,F(8,3),COL(6),A,F(5,0));
IF PROFICI <= '011'B
  THEN CALL MESSAGE;
IF PLACE = 1 THEN GO TO THE@END;
IF PLACE = 2 THEN GO TO THE@END;
GO TO THE@END;

/*****
MESSAGE: PROCEDURE;
*****/

/* IF ((PLACE > 3)&(PLACE < 10))
  THEN PUT FILE(DISPLAY) EDIT
    ('COST DATA CARD:',CST@CRD)
    (COL(6),A,A(80)); */
PUT FILE(DISPLAY) EDIT
  ('PLACE THE ERROR MESSAGE TEXT HERE.')
  (COL(6),A);

END MESSAGE;

END ERROR;

```

5.2.2 ALTER

This procedure has not been thoroughly tested. Eventually the procedure will modify, according to the user's commands, data defining the user's nodes, MUFs, or marginal functions. Note that the structure of the decision tree may not be altered; only parameters related to the various nodes and functions may be changed.

The source code of this procedure is provided on the following pages.

```

/*****/
ALTER: PROCEDURE;
/*****/

/* ACCEPTS THREE COMMANDS TO ALTER NODE, UTILITY OR MARG. FUNC. */

DECLARE
  #@ATTRI      REAL FIXED BINARY(15) INIT(1) EXTERNAL,
  BATCH        BIT(1) EXTERNAL,
  STRING       CHAR(80),
  #UFS         REAL FIXED BINARY(15) EXTERNAL,
  COMMAND      CHAR(80) EXTERNAL,
  DEBUG        BIT(2) INIT('11'B) EXTERNAL,
  DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
  ID@LENG      FIXED BINARY(15) EXTERNAL,
  INDEX        BUILTIN,
  LISTM        EXTERNAL ENTRY,
  NO@INCR      REAL FIXED BINARY(15) EXTERNAL,
  NULLO        POINTER EXTERNAL,
  ONCODE       BUILTIN,
  P@MARG(#@ATTRI) POINTER CTL EXTERNAL,
  P@UTIL(#UFS)  POINTER CTL EXTERNAL,
  SUBSTR       BUILTIN,
  USER        STREAM INPUT EXTERNAL,
  UTILNO       REAL FIXED BINARY(15) EXTERNAL,
  PEEL         EXTERNAL ENTRY RETURNS(REAL FIXED BINARY(15)),
  CARD         CHAR(80),
  LB           REAL FIXED BINARY(15),
  VAL          REAL FLOAT BINARY(21),
  LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
  STRIP        EXTERNAL ENTRY RETURNS(REAL FIXED BINARY(15)),
  PLACE       REAL FIXED DECIMAL(7,3) EXTERNAL,
  ERROR        EXTERNAL ENTRY,
  NAME         CHAR(80),
  1 MARGIN     BASED,
  2 NAME       CHAR(4),
  2 PARM(3)    REAL FLOAT BINARY(21),
  1 GENERAL    BASED,
  2 NAME       CHAR(4),

```

```

2 #@INCR      REAL FIXED BINARY(15),
2 UP@LIM(NO@INCR REFER(GENERAL.#@INCR))
              REAL FLOAT BINARY(21),
2 VALUE(NO@INCR REFER(GENERAL.#@INCR))
              REAL FLOAT BINARY(21),
1 NODE        BASED,
2 #@I         REAL FIXED BINARY(15),
2 FATHER      POINTER,
2 #1@SON      POINTER,
2 R@SIBLING   POINTER, /* SAME LEV ONLY, NOT BRO.*/
2 #@SONS      REAL FIXED BINARY(15),
2 OPT@FLAG    BIT(1),
2 ID          CHAR(65),
2 COST(#@ATTRI REFER(NODE.#@I))
              REAL FLOAT BINARY(21),
2 PROB        REAL FLOAT BINARY(21),
2 UTILITY     REAL FLOAT BINARY(21),
2 DOWN        POINTER,
1 UTILITY     BASED,
2 NAME        CHAR(3),
2 #@I         REAL FIXED BINARY(15),
2 PARM(#@ATTRI REFER(UTILITY.#@I))
              REAL FLOAT BINARY(21),
2 K           REAL FLOAT BINARY(21),
NUM           REAL FIXED BINARY(15),
P1            POINTER,
(I,J,LEN)     REAL FIXED BINARY(15);

```

```

ON CONVERSION BEGIN;
              PLACE = 1.1;
              CALL ERROR;
              GO TO STOPSA;
END;

PLACE = 80.1;
SELECT;
  WHEN(INDEX(COMMAND,' NODE') > 0) CALL ALTNOD;
  WHEN(INDEX(COMMAND,' MARG') > 0) CALL ALTMARG;
  OTHERWISE CALL ERROR;
END;

```

```

/*****
ALTNOD: PROC;
*****/

```

```

/* TO MODIFY A NODE; COST(I) OR PROB ONLY */

```

```

CARD = PEEL(COMMAND);
IF CARD = ' '
  THEN DO;
    PLACE = 1.01;
    CALL ERROR;
    GO TO STOPSA;
  END;

```

```

ELSE P1 = LOCATE(CARD);
IF P1 = NULLO
THEN DO;
    PLACE = 1.2;
    CALL ERROR;
    GO TO STOPSA;
END;

```

```

IF INDEX(COMMAND,'PROB') > 0
THEN GO TO ALTPRO;
ELSE;
I = INDEX(COMMAND,'COST');
IF I <= 0
THEN;
ELSE COSTS: DO;

```

```

/* ALTER COSTS(I) */
LB = INDEX(COMMAND,'#') + 1;
IF LB <= 1
THEN CALL MODCSTS;
ELSE SOMECS: DO;

```

```

/* ALTER SELECTED COSTS(I) */

LEN = 81 - LB;
CARD = SUBSTR(COMMAND,LB,LEN);
LB = STRIP(CARD);
IF LB < 1
THEN LEAVE SOMECS;
ELSE LEN = 81 - LB;
I = SUBSTR(CARD,1,LEN);
IF I < 0 | I > P1->NODE.#@I
THEN DO;
    PLACE = 1.4;
    CALL ERROR;
    GO TO STOPSA;
END;
ELSE;
IF BATCH = '0'B
THEN PUT FILE(DISPLAY) EDIT
    ('VALUE OF ATTRIBUTE ',I,' IS ',
     P1->NODE.COST(I),
     'ENTER NEW VALUE :')
    (SKIP(2),A,F(5,0),A,E(12,5),
     COL(20),A);
ELSE;
GET FILE(USER) EDIT
    (STRING)(COL(1),A(80));
IF STRING = ' '
THEN;
ELSE P1->NODE.COST(I) = STRING;
GO TO SOMECS;
END SOMECS;

```

```

        RETURN;
    END COSTS;
CALL MODCSTS;

```

```
/* ALTER PROB */
```

```
ALTPRO:
```

```

    IF BATCH = '0'B
    THEN PUT FILE(DISPLAY) EDIT
        ('VALUE OF PROBABILITY IS ',
         P1->NODE.PROB,
         'ENTER NEW VALUE :')
        (SKIP(2),A,E(12,5),
         COL(20),A);

    ELSE;
    GET FILE(USER) EDIT
        (STRING)(COL(1),A(80));
    IF STRING = ' '
    THEN;
    ELSE P1->NODE.PROB = STRING;

```

```

/*****/
MODCSTS: PROCEDURE;
/*****/

```

```
/* MODIFY ALL COSTS(I) */
```

```

DO I = 1 TO P1->NODE.#@1;
    IF BATCH = '0'B
    THEN PUT FILE(DISPLAY) EDIT
        ('VALUE OF ATTRIBUTE ',I,' IS ',
         P1->NODE.COST(I),
         'ENTER NEW VALUE :')
        (SKIP(2),A,F(5,0),A,E(12,5),
         COL(20),A);

    ELSE;
    GET FILE(USER) EDIT
        (STRING)(COL(1),A(80));
    IF STRING = ' '
    THEN;
    ELSE P1->NODE.COST(I) = STRING;

```

```
END;
```

```
END MODCSTS;
```

```
END ALTNOD;
```

```

/*****/
ALTMARG: PROC;
/*****/

```

```
/* ALTERS THE MARGINAL FUNCTIONS */
```

```
LB = INDEX(COMMAND,'#') + 1;
```

```
IF LB <= 1
```

```
/* ALTER ALL MARG FUNCTIONS */
```

```
THEN DO I = 1 TO #@ATTRI;
```

```
CALL ALMAR(I);
```

```
END;
```

```
/* ALTER SELECTED MARGINAL FUNCTIONS */
```

```
ELSE SOMEM: DO WHILE('1'B);
```

```
LEN = 81 - LB;
```

```
CARD = SUBSTR(COMMAND, LB, LEN);
```

```
LB = STRIP(CARD);
```

```
IF LB <= 1
```

```
THEN LEAVE SOMEM;
```

```
ELSE LEN = 81 - LB;
```

```
I = SUBSTR(CARD, 1, LEN);
```

```
IF I < 0 | I > #@ATTRI
```

```
THEN DO;
```

```
PLACE = 22.1;
```

```
CALL ERROR;
```

```
RETURN;
```

```
END;
```

```
ELSE CALL ALMAR(I);
```

```
END SOMEM;
```

```
/******
```

```
ALMAR: PROC(I);
```

```
/******
```

```
/* MODIFIES MARGIN(I), THE MARGINAL FUNCTION DATA */
```

```
DECLARE I REAL FIXED BINARY(15);
```

```
CALL LISTM(I);
```

```
IF P@MARG(I)->MARGIN.NAME = 'GENE'
```

```
THEN DO;
```

```
PLACE = 22.2;
```

```
GO TO STOPSA;
```

```
RETURN;
```

```
END;
```

```
ELSE;
```

```
/* ALTER NAME */
```

```
IF BATCH
```

```
THEN;
```

```
ELSE PUT FILE(DISPLAY) EDIT
```

```
('ENTER NEW 4 CHARACTER NAME:')
```

```
(SKIP(2),A);
```

```
GET FILE(USER) EDIT
```

```
(NAME)(COL(1),A(4));
```

```
IF NAME = 'GENE'
```

```
THEN DO;
```

```
PLACE = 22.3;
```



```

        CALL ERROR;
        RETURN;
    END;
ELSE P@MARG(I)->MARGIN.NAME = NAME;

/* ALTER PARM(I), I = 1,2,3 */
DO J = 1 TO 3;
    IF BATCH
        THEN;
        ELSE PUT FILE(DISPLAY) EDIT
            ('ENTER NEW VALUE OF PARAMETER ',J,':')
            (SKIP,A,F(5,0),A);
        GET FILE(USER) EDIT
            (NAME)(COL(1),A(80));
        P@MARG(I)->MARGIN.PARM(J) = NAME;
    END;
END ALMAR;

END ALTMARG;

STOPSA: END ALTER;

```

5.2.3 BUILD

IDAP's BUILD procedure is simple. It serves as a branching point, passing control of the program to READRAW and POINT, in that order. This procedure is invoked after a command to build a tree from the user's formatted input file.

Both a flow chart (Fig. 5.10) of the internal modular flow of BUILD and its source code are provided on the following pages.

Fig. 5.10 The BUILD Procedure

```

/*****
BUILD: PROCEDURE;
*****/

```

```

DECLARE   FIL      CHAR(80) VARYING;
DECLARE
  COMMAND      CHAR(80) EXTERNAL,
  FILENAM      FILE VARIABLE EXTERNAL,
  PLACE        REAL FIXED DECIMAL(7,3) EXTERNAL,
  RAWDAT1      STREAM INPUT EXTERNAL,
  RAWDAT2      STREAM INPUT EXTERNAL,
  RAWDAT3      STREAM INPUT EXTERNAL,
  SUBSTR       BUILTIN,
  CST@CRD      CHAR(80) CTL EXTERNAL;
DECLARE
  PROCESS      EXTERNAL ENTRY,
  SUBTOL       EXTERNAL ENTRY,
  GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOPOINT      EXTERNAL ENTRY,
  LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
  LONODE       EXTERNAL ENTRY,
  PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
  PRINT        EXTERNAL ENTRY,
  LODOWN       EXTERNAL ENTRY,
  ERROR        EXTERNAL ENTRY,
  READRAW      EXTERNAL ENTRY,
  UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT        EXTERNAL ENTRY;

```

```

PLACE = 0.52;
FIL = PEEL(COMMAND);
IF FIL = ' '
  THEN CALL ERROR;
  ELSE DO;
    PLACE = 0.53;
    FIL=SUBSTR(FIL,1,7);
    SELECT(FIL);
      WHEN('RAWDAT1') FILENAM = RAWDAT1;
      WHEN('RAWDAT2') FILENAM = RAWDAT2;
      WHEN('RAWDAT3') FILENAM = RAWDAT3;
      OTHERWISE CALL ERROR;
    END;
    OPEN FILE(FILENAM);
    ALLOCATE CST@CRD;
    CALL READRAW;
    FREE CST@CRD;
    CLOSE FILE(FILENAM);
    CALL POINT;
  END;
END BUILD;

```

5.2.4 CALCK

If a MUF is of the multiplicative form (see Sec. 3), the constant K of Eq. 5 is calculated by IDAP. The routine for doing this, as seen in the CALCK procedure, will now be briefly explained*.

$$y = \prod_{i=1}^n (1 + K k_i) - (1 + K) \quad (16)$$

Eq. 16, obtained by manipulating Eq. 5, is used in the CALCK procedure to calculate K. The procedure evaluates y for various values of K until it is sufficiently close to 0, and then that value of K is used in the rest of the program.

Fig. 5.11 represents the general function y(K) that was used to calculate K at the point where y(K)=0. As shown in that figure, there are two types of curves generated by Eq. 16. One is y(K) when the sum of the k_i is less than 1 (0<K), the other y(K) when the sum of the k_i is greater than 1 (-1<K<0). For the multiplicative form of IDAP's MUFs, these two cases are the only valid ones; if the sum of the k_i equals 1, the MUF form is additive.

In the CALCK algorithm, it is necessary only to search the interval (-1,0) if the sum of the k_i is greater than 1, or the interval from 0 onward if the sum of the k_i is less than 1.

IDAP divides the interval (-1,0) into halves when searching for K<0. When K>0, IDAP substitutes successive powers of 2 for K until y(K)>0. This value of K then defines the interval (0,K) on which the search may then be limited. This new interval is halved until a K is found such that y(K) is sufficiently close to 0.

The modular flow of CALCK is presented in Fig. 5.12, and its source code is also presented.

*This algorithm is a modification of the one presented in Sicherman.⁹

$$y = \prod_{i=1}^n (1 + K K_i) - (1 + K)$$

Fig. 5.11 Calculating K for a Multiplicative MUF

Fig. 5.12 The CALCK Procedure

Fig. 5.13 The FIND Module - Within CALCK

Fig. 5.14 The OPPOSE Module -
Within CALCK

NOTE: $\prod (1 - X \times \text{PARAM}(I)) - (1 + X)$

Fig. 5.15 The Y Module -
Within CALCK


```

/*****
CALCK: PROCEDURE RETURNS(REAL FLOAT BINARY(21));
*****/

```

```

/* THIS PROC CALCULATES THE PARAMETER UTILITY.K FROM
THE EQUATION Y=PRODUCT-OVER-I(1-K*PARAM(I)) - (1-K). */

```

```

DECLARE

```

```

#@ATTRI      REAL FIXED BINARY(15) EXTERNAL,
UTILNO        REAL FIXED BINARY(15) EXTERNAL,
MAXTEST       REAL FLOAT BINARY(21) INIT(1048576), /* 2**20 */
KAPPROX       REAL FLOAT BINARY(21),
DELTA         REAL FLOAT BINARY(21) INIT(0.0001),
SUMK          REAL FLOAT BINARY(21) INIT(0),
RET           REAL FLOAT BINARY(21),
TWO           REAL FLOAT BINARY(21) INIT(2.0000000),
ABS           BUILTIN,
DEBUG         BIT(2) EXTERNAL,
DISPLAY       STREAM OUTPUT PRINT EXTERNAL,
ERROR         EXTERNAL ENTRY,
I             REAL FIXED BINARY(15),
#UFS          REAL FIXED BINARY(15) EXTERNAL,
PLACE        REAL FIXED DECIMAL(7,3) EXTERNAL,
P@UTIL(#UFS)  POINTER EXTERNAL CTL,
1 UTILITY     BASED,
2 NAME        CHAR(3),
2 #@I         REAL FIXED BINARY(15),
2 PARM(#@ATTRI REFER(UTILITY.#@I))
              REAL FLOAT BINARY(21),
2 K           REAL FLOAT BINARY(21),
LOWER         REAL FLOAT BINARY(21) INIT(-1.),
UPPER         REAL FLOAT BINARY(21) INIT(-0.001);

```

```

DO I = 1 TO #@ATTRI;
  SUMK = SUMK + P@UTIL(UTILNO)->UTILITY.PARM(I);
END;
IF ABS(SUMK - 1.) < 0.00001
  THEN RETURN(-99.); /* AS A FLAG OF AN ERROR */
ELSE;
IF SUMK > 1.
  THEN CALL FIND; /* -1<K<0 */
ELSE DO; /* 0<K */
  LOWER = 0.001;
  UPPER = TWO;
  INC: DO WHILE(
    & UPPER <= MAXTEST);
    UPPER = TWO * UPPER;
  END INC;
  PLACE = 66.1;
  IF UPPER > MAXTEST
    THEN CALL ERROR;
  ELSE;
    CALL FIND;
  END;
END;

```

```
RETURN(KAPPROX);
```

```

/*****/
FIND: PROCEDURE;
/*****/

```

```
/* THIS PROC FINDS UTILITY.K +- DELTA, SUMK > 1 */
```

```

PLACE = 66.2;
IF OPPOSE(Y(UPPER),Y(LOWER))
  THEN;
  ELSE CALL ERROR;
DO WHILE(UPPER-LOWER > DELTA);
  KAPPROX = (UPPER + LOWER)/TWO;
  IF OPPOSE(Y(LOWER),Y(KAPPROX))
    THEN UPPER = KAPPROX;
    ELSE LOWER = KAPPROX;
END;

```

```
END FIND;
```

```

/*****/
OPPOSE: PROCEDURE(U,L) RETURNS(BIT(1));
/*****/

```

```
/* THIS FUNCTION RETURNS 1 IF THE SIGNS OF U AND L
ARE DIFFERENT, AND 0 IF THEY HAVE THE SAME SIGNS */
```

```

DECLARE
  SIGN      BUILTIN,
  U          REAL FLOAT BINARY(21),
  L          REAL FLOAT BINARY(21);

```

```

IF SIGN(U) = SIGN(L)
  THEN RETURN('0'B);
  ELSE RETURN('1'B);

```

```
END OPPOSE;
```

```

/*****/
Y: PROCEDURE(X) RETURNS(REAL FLOAT BINARY(21));
/*****/

```

```

/* THIS FUNCTION RETURNS THE VALUE OF THE EQUATION:
  Y = PROD-OVER-I(1-X*PARAM(I)) - (1 - X) */

```

```

DECLARE
  RET      REAL FLOAT BINARY(21) INIT(1),
  X        REAL FLOAT BINARY(21),
  ONE      REAL FLOAT BINARY(21) INIT(1.00000),
  I        REAL FIXED BINARY(15);

```

```

DO I = 1 TO #@ATTRI;
  RET = RET * (ONE + X*P@UTIL(UTILNO)->UTILITY.PARM(I));
END;
RET = RET - (ONE + X);
IF DEBUG = '11'B
  THEN PUT FILE(DISPLAY) EDIT
        ('Y VALUE OF ',X,' IS ',RET)
        (SKIP, COL(20),A,E(12,5),A,E(12,5));
  ELSE;
RETURN(RET);

END Y;

END CALCK;

```

5.2.5 LISTM

This procedure prints out the data that define a marginal function provided by the user. It is passed a number that represents the number of the attribute on which that marginal function operates.

The flow of this procedure is provided in Fig. 5.16, and the source code follows.

Fig. 5.16 The LISTM Procedure

```

/*****
LISTM: PROCEDURE(NO);
*****/

```

```

/* LISTS DATA ABOUT MARGINAL FUNCTION(NO) */

```

```

DECLARE

```

```

  1 MARGIN          BASED,
    2 NAME          CHAR(4),
    2 PARM(3)       REAL FLOAT BINARY(21),
  1 GENERAL         BASED,
    2 NAME          CHAR(4),
    2 #@INCR        REAL FIXED BINARY(15),
    2 UP@LIM(NO@INCR REFER(GENERAL.#@INCR))
                    REAL FLOAT BINARY(21),
    2 VALUE(NO@INCR REFER(GENERAL.#@INCR))
                    REAL FLOAT BINARY(21),
  #@ATTRI          REAL FIXED BINARY(15) INIT(1) EXTERNAL,
  P@MARG(#@ATTRI)  POINTER CTL EXTERNAL,
  I                REAL FIXED BINARY(15),
  DISPLAY          STREAM OUTPUT PRINT EXTERNAL,
  NO               REAL FIXED BINARY(15);

```

```

PUT FILE(DISPLAY) EDIT

```

```

  ('THIS IS A LISTING OF THE MARGINAL FUNCTION ',
   'FOR ATTRIBUTE NUMBER ',NO,
   'THE NAME OF THE FUNCTION IS: ',
   P@MARG(NO)->MARGIN.NAME)
  (SKIP,A,COL(20),A,F(5,0),
   COL(20),A,A(4));

```

```

IF P@MARG(NO)->MARGIN.NAME = 'GENE'

```

```

  THEN DO I = 1 TO P@MARG(NO)->GENERAL.#@INCR;

```

```

    PUT FILE(DISPLAY) EDIT
      ('UPPER LIMIT: ',P@MARG(NO)->GENERAL.UP@LIM(I),
       'VALUE: ',P@MARG(NO)->GENERAL.VALUE(I))
      (2 COL(20),A,E(12,5));

```

```

  END;

```

```

  ELSE DO I = 1 TO 3;

```

```

    PUT FILE(DISPLAY) EDIT
      ('PARAMETER ',I,': ',
       P@MARG(NO)->MARGIN.PARM(I))
      (COL(20),A,F(5,0),A,E(12,5));

```

```

  END;

```

```

END LISTM;

```

5.2.6 LISTU

The LISTU procedure is similar to LISTM, except that data defining a user's MUF are printed. The user is allowed to define any number of MUFs in IDAP, although only one may be used at a time when processing the decision tree.

Fig 5.17 represents the internal modular flow of LISTU; the source code for this procedure is also provided.

Fig. 5.17 The LISTU Procedure

```

/*****
LISTU: PROCEDURE(NO);
*****/

```

```

/* TO LIST THE MAUF CONSTANTS. */
/* LAST UPDATE: 3-26-'79 */

```

```

DECLARE

```

```

#@ATTRI      REAL FIXED BINARY(15) INIT(1) EXTERNAL,
#UFS          REAL FIXED BINARY(15) EXTERNAL,
DISPLAY       STREAM OUTPUT PRINT EXTERNAL,
P@UTIL(#UFS)  POINTER CTL EXTERNAL,
1 UTILITY     BASED,
  2 NAME      CHAR(3),
  2 #@I       REAL FIXED BINARY(15),
  2 PARM(#@ATTRI REFER(UTILITY.#@I))
              REAL FLOAT BINARY(21),
  2 K         REAL FLOAT BINARY(21),
NO            REAL FIXED BINARY(15),
I            REAL FIXED BINARY(15);

```

```

PUT FILE(DISPLAY) EDIT

```

```

('UTILITY FUNCTION ',NO,' HAS THE ')
(SKIP(2),A,F(5,0),A);

```

```

IF P@UTIL(NO)->UTILITY.NAME = 'ADD'

```

```

THEN PUT FILE(DISPLAY) EDIT
('ADDITIVE FORM.')(A);

```

```

ELSE DO;

```

```

  PUT FILE(DISPLAY) EDIT
  ('MULTIPLICATIVE FORM.')(A);

```

```

  PUT FILE(DISPLAY) EDIT
  ('THE CONSTANT OF ',P@UTIL(NO)->UTILITY.K,
   'WAS CALCULATED FROM THE FOLLOWING:')
  (COL(20),A,E(12,5),COL(20),A);

```

```

  END;

```

```

DO I = 1 TO P@UTIL(NO)->UTILITY.#@I;

```

```

  PUT FILE(DISPLAY) EDIT
  ('CONSTANT ',I,': ',
   P@UTIL(NO)->UTILITY.PARM(I))
  (COL(20),A,F(5,0),A,E(12,5));

```

```

END;

```

```

END LISTU;

```

5.2.7 LOCATE

LOCATE is a function that returns the address of a node whose ID or label is passed to it. This unique label is user-supplied; an explanation of node labeling appears in Sec. 3.

DOWN pointers in the PL/1 tree structure are followed until a node is reached whose label is the one desired. The address of that node is then returned. If, after the entire tree is searched for the desired node and that node is not found, LOCATE returns the NULL pointer, which is interpreted as an error later in the program.

The source code and an illustration of the internal modular flow of this procedure (Fig. 5.18) are provided.

Fig. 5.18 The LOCATE Procedure


```

/*****
LOCATE: PROCEDURE(A) RETURNS(POINTER);
*****/

/* RETURNS A POINTER TO THE NODE WITH AN ID EQUAL TO 'A' */

DECLARE  A      CHAR(80),
        P1     POINTER,
        ID1    CHAR(80),
        ID2    CHAR(80),
        LEN    REAL FIXED BINARY(15);

DECLARE
  DEBUG      BIT(2) INIT('11'B) EXTERNAL,
  DISPLAY    STREAM OUTPUT PRINT EXTERNAL,
  HEADER     POINTER EXTERNAL,
  ID@LENG    FIXED BINARY(15) EXTERNAL,
  INDEX      BUILTIN,
  NULLO      POINTER EXTERNAL,
  SUBSTR     BUILTIN,
  1 NODE     BASED,
    2 #@I     REAL FIXED BINARY(15),
    2 FATHER  POINTER,
    2 #1@SON  POINTER,
    2 R@SIBLING POINTER, /* SAME LEV ONLY, NOT BRO.*/
    2 #@SONS  REAL FIXED BINARY(15),
    2 OPT@FLAG BIT(1),
    2 ID      CHAR(65),
    2 COST(#@ATTRI REFER(NODE.#@I))
          REAL FLOAT BINARY(21),
    2 PROB    REAL FLOAT BINARY(21),
    2 UTILITY  REAL FLOAT BINARY(21),
    2 DOWN    POINTER;

DECLARE
  PROCESS    EXTERNAL ENTRY,
  BUILD      EXTERNAL ENTRY,
  SUBTOL     EXTERNAL ENTRY,
  GEORGE     EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOPOINT    EXTERNAL ENTRY,
  LONODE     EXTERNAL ENTRY,
  PEEL       EXTERNAL ENTRY RETURNS(CHAR(80)),
  PRINT      EXTERNAL ENTRY,
  LODOWN     EXTERNAL ENTRY,
  ERROR      EXTERNAL ENTRY,
  READRAW    EXTERNAL ENTRY,
  UTILOF     EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT      EXTERNAL ENTRY;

/* TAKE OFF LAST '.', IF ANY. */
IF DEBUG > '00'B
  THEN PUT FILE(DISPLAY) EDIT
    ('INITIATE LOCATION OF:',A)
    (SKIP(2),A,A(80));
ELSE;
ID1 = A;

```

```

LEN = INDEX(ID1,' ') - 1;
IF SUBSTR(ID1,LEN,1) = ' '
    THEN LEN = LEN - 1;
    ELSE;
ID1 = SUBSTR(ID1,1,LEN);
P1 = HEADER;
TRAV: DO UNTIL(P1 = NULLO);
        /* TAKE OFF LAST ' ', IF ANY. */
ID2 = P1->NODE.ID;
LEN = INDEX(ID2,' ') - 1;
IF SUBSTR(ID2,LEN,1) = ' '
    THEN LEN = LEN - 1;
    ELSE;
ID2 = SUBSTR(ID2,1,LEN);
        /* COMPARE THE TWO ID'S. */
IF ID1 = ID2
    THEN RETURN(P1);
    ELSE P1 = P1->NODE.DOWN;
END TRAV;

RETURN(NULLO); /* NODE CAN'T BE FOUND. */

END LOCATE;

```

5.2.8 LODOWN

This is a procedure that utilizes the DOWN pointers of nodes. All data defining all the nodes of the user's decision tree are printed when this procedure is invoked. For every node encountered, this procedure calls the PRINT procedure that actually prints the data.

The flow of this procedure is presented in Fig. 5.19; the source code is also provided.

Fig. 5.19 The LODOWN Procedure

```

/*****
LODOWN: PROCEDURE;
*****/

/* TRAVERSES THE ENTIRE TREE, PRINTING ALL NODE DATA. */

DCL P1      POINTER;
DECLARE
  DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
  HEADER       POINTER EXTERNAL,
  ID@LENG      FIXED BINARY(15) EXTERNAL,
  NULLO       POINTER EXTERNAL,
  1 NODE       BASED,
    2 #@I       REAL FIXED BINARY(15),
    2 FATHER    POINTER,
    2 #1@SON     POINTER,
    2 R@SIBLING  POINTER, /* SAME LEV ONLY, NOT BRO.*/
    2 #@SONS     REAL FIXED BINARY(15),
    2 OPT@FLAG   BIT(1),
    2 ID        CHAR(65),
    2 COST(#@ATTRI REFER(NODE.#@I))
              REAL FLOAT BINARY(21),
    2 PROB      REAL FLOAT BINARY(21),
    2 UTILITY   REAL FLOAT BINARY(21),
    2 DOWN     POINTER;
DECLARE
  PROCESS      EXTERNAL ENTRY,
  BUILD        EXTERNAL ENTRY,
  SUBTOL       EXTERNAL ENTRY,
  GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOPOINT      EXTERNAL ENTRY,
  LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
  LONODE       EXTERNAL ENTRY,
  PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
  PRINT        EXTERNAL ENTRY,
  ERROR        EXTERNAL ENTRY,
  READRAW      EXTERNAL ENTRY,
  UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT        EXTERNAL ENTRY;

  PUT FILE(DISPLAY) EDIT
    ('**** NODE LISTING ****')
    (SKIP(5),COL(20),A);
  P1 = HEADER;
LOCA: DO UNTIL(P1 = NULLO);
  CALL PRINT(P1);
  P1 = P1->NODE.DOWN;
END LOCA;

  PUT FILE(DISPLAY) EDIT
    ('**** NODE LISTING COMPLETE ****')(SKIP(5),COL(16),A);

END LODOWN;

```

5.2.9 LONODE

When invoked, this procedure prints out the data (by calling PRINT) defining specific nodes in the tree. These nodes are determined by the format of the user's input command.

The flow of this procedure is presented in Fig. 5.20, and its source code is also provided.

Fig. 5.20 The LONODE Procedure

```

/*****/
LONODE: PROCEDURE;
/*****/

/* LISTS THE NODES DESIRED IN THE COMMAND. */

DECLARE   Q1      REAL FIXED BINARY(15),
          LEN      REAL FIXED BINARY(15),
          COM      CHAR(80),
          P1       POINTER,
          PASS     CHAR(80);

DECLARE
  BATCH      BIT(1) INIT('1'B) EXTERNAL,
  COMMAND    CHAR(80) EXTERNAL,
  DEBUG      BIT(2) INIT('11'B) EXTERNAL,
  DISPLAY    STREAM OUTPUT PRINT EXTERNAL,
  INDEX      BUILTIN,
  NULLO      POINTER EXTERNAL,
  PLACE      REAL FIXED DECIMAL(7,3) EXTERNAL,
  SUBSTR     BUILTIN;

DECLARE
  PROCESS    EXTERNAL ENTRY,
  BUILD      EXTERNAL ENTRY,
  SUBTOL     EXTERNAL ENTRY,
  GEORGE     EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOPOINT    EXTERNAL ENTRY,
  LOCATE     EXTERNAL ENTRY RETURNS(POINTER),
  PEEL       EXTERNAL ENTRY RETURNS(CHAR(80)),
  PRINT      EXTERNAL ENTRY,
  LODOWN     EXTERNAL ENTRY,
  ERROR      EXTERNAL ENTRY,
  READRAW    EXTERNAL ENTRY,
  UTILOF     EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT      EXTERNAL ENTRY;

  PLACE = 13;
  COM = COMMAND;

/* PULL OFF NODES ONE AT A TIME. */
LOOP: DO WHILE('1'B);
  Q1 = INDEX(COM,'');
  LEN = 81 - Q1;
  COM = SUBSTR(COM,Q1,LEN);
  PASS = PEEL(COM);
  IF (DEBUG = '11'B & BATCH = '1'B)
    THEN PUT FILE(DISPLAY) EDIT
      ('AAA PASS =',PASS)
      (SKIP,A,A(80));
  ELSE;
  IF PASS = ' '
    THEN LEAVE LOOP;
  ELSE DO;
    /* LOCATE THEM AND PRINT THEM. */
    P1 = LOCATE(PASS);
    PLACE = 16.;

```

```

        IF P1 = NULLO
            THEN CALL ERROR;
            ELSE CALL PRINT(P1);
        END;
        /* START WITH THE NEXT NODE DESIRED. */
        Q1 = INDEX(COM, ' ') + 1;
        IF Q1 = 1
            THEN LEAVE LOOP;
            ELSE DO;
                LEN = 81 - Q1;
                COM = SUBSTR(COM, Q1, LEN);
            END;
        END LOOP;

        END LONODE;

```

5.2.10 LOPOINT

When the user wishes to list the data available to the various pointer variables of a node, this procedure may be invoked. When the variable DEBUG is greater than '00'B and LOPOINT is called, the labels of the nodes whose addresses are contained in the pointer variables will be printed.

Fig. 5.21 presents the internal modular flow of LOPOINT and the source code is also provided.

Fig. 5.21 The LOPOINT Procedure


```

/*****
LOPOINT: PROCEDURE(A);
*****/

/* PRINTS THE POINTERS OF A->NODE */

DECLARE
  A      POINTER,
  P1     POINTER;
DECLARE
  DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
  ID@LENG      REAL FIXED BINARY(15) EXTERNAL,
  NULLO        POINTER EXTERNAL,
  PLACE        REAL FIXED DECIMAL(7,3) EXTERNAL,
  1 NODE
    2 #@I      REAL FIXED BINARY(15),
    2 FATHER    POINTER,
    2 #1@SON    POINTER,
    2 R@SIBLING POINTER, /* SAME LEV ONLY, NOT BRO.*/
    2 #@SONS    REAL FIXED BINARY(15),
    2 OPT@FLAG  BIT(1),
    2 ID        CHAR(65),
    2 COST( #@ATTRI REFER(NODE. #@I))
      REAL FLOAT BINARY(21),
    2 PROB      REAL FLOAT BINARY(21),
    2 UTILITY   REAL FLOAT BINARY(21),
    2 DOWN      POINTER;
DECLARE
  PROCESS      EXTERNAL ENTRY,
  BUILD        EXTERNAL ENTRY,
  SUBTOL       EXTERNAL ENTRY,
  GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
  LONODE       EXTERNAL ENTRY,
  PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
  PRINT        EXTERNAL ENTRY,
  LODOWN       EXTERNAL ENTRY,
  ERROR        EXTERNAL ENTRY,
  READRAW      EXTERNAL ENTRY,
  UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT        EXTERNAL ENTRY;

PLACE = 20;
PUT FILE(DISPLAY) EDIT
  ('* POINTER LISTING OF ', A->NODE.ID)
  (SKIP, COL(10), A, A(ID@LENG));
PUT FILE(DISPLAY) EDIT
  ('FATHER: ')(COL(13), A);
IF A->NODE.FATHER = NULLO
  THEN PUT FILE(DISPLAY) EDIT
    ('NULL')(A);
  ELSE DO;
    P1 = A->NODE.FATHER;
    PUT FILE(DISPLAY) EDIT

```

```

        (P1->NODE.ID)(A(ID@LENG));
    END;
    PUT FILE(DISPLAY) EDIT
        ('#1@SON: ')(COL(13),A);
    IF A->NODE.#1@SON = NULLO
    THEN PUT FILE(DISPLAY) EDIT
        ('NULL')(A);
    ELSE DO;
        P1 = A->NODE.#1@SON;
        PUT FILE(DISPLAY) EDIT
            (P1->NODE.ID)(A(ID@LENG));
    END;
    PUT FILE(DISPLAY) EDIT
        ('R@SIBLING: ')(COL(13),A);
    IF A->NODE.R@SIBLING = NULLO
    THEN PUT FILE(DISPLAY) EDIT
        ('NULL')(A);
    ELSE DO;
        P1 = A->NODE.R@SIBLING;
        PUT FILE(DISPLAY) EDIT
            (P1->NODE.ID)(A(ID@LENG));
    END;
    PUT FILE(DISPLAY) EDIT
        ('DOWN: ')(COL(13),A);
    IF A->NODE.DOWN = NULLO
    THEN PUT FILE(DISPLAY) EDIT
        ('NULL')(A);
    ELSE DO;
        P1 = A->NODE.DOWN;
        PUT FILE(DISPLAY) EDIT
            (P1->NODE.ID)(A(ID@LENG));
    END;
END LOPOINT;

```

5.2.11 PEEL

This function is passed an 80-character string, and returns the substring between the first pair of single quotation marks within that string. The flow is illustrated in Fig. 5.22, and the source code is also provided.

Fig. 5.22 The PEEL Procedure

```

/*****
PEEL: PROCEDURE(A) RETURNS(CHAR(80));
*****/

/* TAKES STRING 'A' AND PEELS OFF THE SUBSTRING
   BETWEEN THE FIRST SET OF QUOTATION MARKS, AND
   RETURNS THAT SUBSTRING. */

DECLARE  A      CHAR(80),
        RET     CHAR(80),
        POS     REAL FIXED BINARY(15),
        LEN     REAL FIXED BINARY(15);

DECLARE
  INDEX      BUILTIN,
  PLACE      REAL FIXED DECIMAL(7,3) EXTERNAL,
  SUBSTR     BUILTIN;
DECLARE
  PROCESS    EXTERNAL ENTRY,
  BUILD      EXTERNAL ENTRY,
  SUBTOL     EXTERNAL ENTRY,
  GEORGE     EXTERNAL ENTRY RETURNS(CHAR(80)),
  LOPOINT    EXTERNAL ENTRY,
  LOCATE     EXTERNAL ENTRY RETURNS(POINTER),
  LONODE     EXTERNAL ENTRY,
  PRINT      EXTERNAL ENTRY,
  LODOWN     EXTERNAL ENTRY,
  ERROR      EXTERNAL ENTRY,
  READRAW    EXTERNAL ENTRY,
  UTILOF     EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
  POINT      EXTERNAL ENTRY;

PLACE = 15;
POS = INDEX(A, ' ') + 1;
IF POS = 1
  THEN RETURN(' ');
  ELSE;
LEN = 81 - POS;
RET = SUBSTR(A, POS, LEN);
LEN = INDEX(RET, ' ') - 1;
IF LEN < 0
  THEN DO;
    CALL ERROR;
    RETURN(' ');
  END;
  ELSE;
RET = SUBSTR(RET, 1, LEN);
RETURN(RET);

END PEEL;

```

5.2.12 POINT

This procedure is invoked only after the DOWN pointer variables have been calculated for all nodes within the user's decision tree.

The pointer variables FATHER and #1SON are calculated for each node in this procedure (as well as the variable #SONS). Only once during the construction of a tree is this procedure called.

Fig. 5.23 illustrates the internal modular flow of this procedure. The source code is also provided.

Fig. 5.23 The POINT Procedure

Fig: 5.24 The DAD@LEN Module - Within POINT

Fig. 5.25 The R@SIBS Module - Within POINT


```

/*****
POINT: PROCEDURE;
*****/

```

```

/* HAVE NODE.DOWN COMPLETED AT THIS POINT.
   NOW MUST ESTABLISH THE THREE OTHER POINTERS.
   LAST UPDATE: 9-13-80.
*/

```

```

DECLARE P1          POINTER,
        PTR          POINTER,
        LENG@1       REAL FIXED BINARY(15),
        LEN          REAL FIXED BINARY(15);

DECLARE
#@LEVEL      REAL FIXED BINARY(15) INIT(12) EXTERNAL,
#@NODES      REAL FIXED BINARY(15) EXTERNAL,
DEBUG        BIT(2) INIT('11'B) EXTERNAL,
DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
HEADER        POINTER EXTERNAL,
ID@LENG      REAL FIXED BINARY(15) EXTERNAL,
INDEX        BUILTIN,
NULLO        POINTER EXTERNAL,
PLACE        REAL FIXED DECIMAL(7,3) EXTERNAL,
SUBSTR        BUILTIN,
LEVEL(@LEVEL) POINTER CTL EXTERNAL,
1 NODE
  2 #@I        REAL FIXED BINARY(15),
  2 FATHER     POINTER,
  2 #1@SON     POINTER,
  2 R@SIBLING  POINTER, /* SAME LEV, NOT ONLY BROTHER */
  2 #@SONS     REAL FIXED BINARY(15),
  2 OPT@FLAG   BIT(1),
  2 ID         CHAR(65),
  2 COST(@ATTRI REFER(NODE.#@I))
              REAL FLOAT BINARY(21),
  2 PROB       REAL FLOAT BINARY(21),
  2 UTILITY    REAL FLOAT BINARY(21),
  2 DOWN       POINTER;

DECLARE
PROCESS      EXTERNAL ENTRY,
BUILD        EXTERNAL ENTRY,
SUBTOL       EXTERNAL ENTRY,
GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
LOPOINT      EXTERNAL ENTRY,
LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
LONODE       EXTERNAL ENTRY,
PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
PRINT        EXTERNAL ENTRY,
LODOWN       EXTERNAL ENTRY,
ERROR        EXTERNAL ENTRY,
READRAW      EXTERNAL ENTRY,
UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21));

```

```

/* FIND FATHERS */
PLACE = 10;
P1 = HEADER;
TRAV1: DO UNTIL(P1 = NULLO);
  IF DEBUG = '11'B
    THEN PUT FILE(DISPLAY) EDIT
      ('LOOKING FOR FATHER OF ',P1->NODE.ID)
      (SKIP,A,A(ID@LENG));
  ELSE;

      /* ASSIGN #@SONS AND FATHER,#1@SON. */
      LEN = DAD@LEN(P1);
      LENG@1 = LEN + 1;
      PTR = HEADER;
      TRAV2: DO WHILE(PTR

          /* LOCATE FATHER OF P1->NODE */
          IF (SUBSTR(P1->NODE.ID,1,LEN)=SUBSTR(PTR->NODE.ID,1,LEN))
            & (SUBSTR(PTR->NODE.ID,LENG@1,1) = '.' |
              SUBSTR(PTR->NODE.ID,LENG@1,1) = ' ')
            THEN FOUND: DO;
              IF DEBUG = '11'B
                THEN PUT FILE(DISPLAY) EDIT
                  ('FOUND FATHER')(SKIP,A);
              ELSE;
              PTR->NODE.#@SONS = PTR->NODE.#@SONS + 1;
              IF PTR->NODE.#@SONS = 1
                THEN PTR->NODE.#1@SON = P1;
              ELSE;
              P1->NODE.FATHER = PTR;
              LEAVE TRAV2;
            END FOUND;
          ELSE IF DEBUG = '11'B
            THEN PUT FILE(DISPLAY) EDIT
              ('DIDN'T FIND FATHER')(SKIP,A);
          ELSE;
          PTR = PTR->NODE.DOWN;
        END TRAV2;
        P1 = P1->NODE.DOWN;
      END TRAV1;
    CALL R@SIBS;

```

```

/*****
DAD@LEN: PROCEDURE(A) RETURNS(REAL FIXED BINARY(15));
*****/

```

```

/* GIVEN AN ID, THIS PROC CALCULATES THE LENGTH
   THAT THE ID OF THE FATHER SHOULD HAVE, LESS '.' */

```

```

DECLARE A          POINTER,
        LEN        REAL FIXED BINARY(15),

```

```

        LENGTH      REAL FIXED BINARY(15),
        I            REAL FIXED BINARY(15);

    LENGTH = INDEX(A->NODE.ID, ' ') - 1;
    DO I = LENGTH-1 TO 1 BY -1 WHILE(I > 0);
        IF SUBSTR(A->NODE.ID,I,1) = '.'
            THEN DO;
                LEN = I - 1;
                RETURN(LEN);
            END;
        ELSE;
            END;
    RETURN(0.); /* IF NODE HAS NO FATHER */

END DAD@LEN;

/*****
R@SIBS: PROCEDURE;
*****/

/* THIS IS TO ASSIGN THE R@SIBLING POINTERS */

DECLARE P1      POINTER, /* NODE IN QUESTION */
        PF      POINTER, /* FATHER OF P1 */
        PS      POINTER, /* R@SIBLING OF FATHER */
        PC      POINTER, /* #1@SON OF PS(UNCLE) */
        I      REAL FIXED BINARY(15) INIT(0),
        LEV     REAL FIXED BINARY(15) INIT(1);

DECLARE
    1 CHAIN      BASED(P1), /* A LIST OF NODES
                           ORDERED BY LEVEL */
    2 NEXT       POINTER,
    2 ID         POINTER, /* POINTING TO A NODE */
    LINK         POINTER, /* POINTING TO A LINK
                           IN "CHAIN" */
    (C_HEAD,C_LAST) POINTER;

/* ASSIGN IMMEDIATE BROTHERS = THOSE
WITH THE SAME FATHER-POINTERS.
IT IS NECESSARY THAT BROTHERS BE
ADJACENT TO EACH OTHER IN THE
R@SIBLING CHAINS.
*/

PLACE = 11;
P1 = HEADER;
TRAVERSA: DO UNTIL(P1 = NULLO);
PS = P1->NODE.DOWN;

```

```

TRAVB: DO WHILE(PS
  IF PS->NODE.FATHER = P1->NODE.FATHER
    THEN DO;
      IF DEBUG = '11'B
        THEN PUT FILE(DISPLAY) EDIT
          ('*** R@SIBLING OF ',P1->NODE.ID,
           ' IS ',PS->NODE.ID)
          (2 (SKIP,A,A(ID@LENG)));
      ELSE;
        P1->NODE.R@SIBLING = PS;
        LEAVE TRAVB;
      END;
    ELSE PS = PS->NODE.DOWN;
  END TRAVB;
  P1 = P1->NODE.DOWN;
END TRAVA;

IF DEBUG = '11'B
  THEN PUT FILE(DISPLAY) EDIT
    ('END OF LOOP TRAVA WITHIN POINTS R@SIBS.')
    (SKIP(2),A);
ELSE;

  /* NOW THAT IMMEDIATE BROTHERS ARE
    ASSIGNED, MUST CONTINUE R@SIBLING
    CHAINS ACROSS TO COUSINS->NODE. */

  PLACE = 14;

  /* ALLOCATE FIRST CHAIN LINK */

  ALLOCATE CHAIN SET(C_HEAD);
  C_HEAD->CHAIN.NEXT = NULLO;
  C_HEAD->CHAIN.ID = HEADER;
  C_LAST = C_HEAD;
  LINK = C_HEAD;

  /* ITERATE THROUGH ALL LINKS IN
    CHAIN, BUILDING ONTO THE END
    OF THE CHAIN AT THE SAME TIME.
  */

  AA: DO WHILE(LINK
    PLACE = 14.1;

    /* ADD ALL OF LINK'S R_SIBLINGS TO CHAIN */

    P1 = LINK->CHAIN.ID->NODE.R@SIBLING;
    AA1: DO WHILE(P1
      PLACE = 14.11;

      /* CHECK IF R_SIBS ALREADY THERE */

      IF INSIDE(P1)
        THEN LEAVE AA1;
      ELSE;

```

```

ALLOCATE CHAIN SET(C_LAST->CHAIN.NEXT);
IF DEBUG = '11'B
    THEN PUT FILE(DISPLAY) EDIT
        ('NEW CHAIN LINK: ',P1->NODE.ID)
        (SKIP,A,A(ID@LENG));
    ELSE;
    C_LAST = C_LAST->CHAIN.NEXT;
    C_LAST->CHAIN.NEXT = NULLO;
    C_LAST->CHAIN.ID = P1;
    P1 = P1->NODE.R@SIBLING;
END AA1;

```

/* ADD LINK'S #1@SON TO CHAIN */

```

P1 = LINK->CHAIN.ID->NODE.#1@SON;
AA2: DO WHILE(P1
    PLACE = 14.12;

```

/* CHECK IF #1@SON ALREADY THERE */

```

IF INSIDE(P1)
    THEN CALL ERROR;
    ELSE;
    ALLOCATE CHAIN SET(C_LAST->CHAIN.NEXT);
    IF DEBUG = '11'B
        THEN PUT FILE(DISPLAY) EDIT
            ('NEW SON AS CHAIN LINK: ',P1->NODE.ID)
            (SKIP,A,A(ID@LENG));
        ELSE;
        C_LAST = C_LAST->CHAIN.NEXT;
        C_LAST->CHAIN.NEXT = NULLO;
        C_LAST->CHAIN.ID = P1;
        P1 = NULLO;
END AA2;

```

/* ADD ALL OF LINK'S #1@SON'S R_SIBLINGS */

```

P1 = C_LAST->CHAIN.ID->NODE.R@SIBLING;
AA3: DO WHILE(P1
    PLACE = 14.13;

```

/* CHECK IF #1@SON'S R_SIBS ALREADY THERE */

```

IF INSIDE(P1)
    THEN LEAVE AA3;
    ELSE;
    ALLOCATE CHAIN SET(C_LAST->CHAIN.NEXT);
    IF DEBUG = '11'B
        THEN PUT FILE(DISPLAY) EDIT
            ('NEW CHAIN LINK: ',P1->NODE.ID)
            (SKIP,A,A(ID@LENG));
        ELSE;
        C_LAST = C_LAST->CHAIN.NEXT;
        C_LAST->CHAIN.NEXT = NULLO;

```

```

      C LAST->CHAIN.ID = P1;
      P1 = P1->NODE.R@SIBLING;
END AA3;

```

```

      /* CONTINUE TO NEXT LINK */

```

```

LINK = LINK->CHAIN.NEXT;
END AA;

```

```

/* NOW THAT THERE'S A LINKED
LIST OF NODE ID'S ORDERED BY
LEVEL, THE R@SIBLING POINTERS
CAN BE CONNECTED SO THAT IM-
MEDIATE BROTHERS IN THE DECISION
TREE ARE NOT THE ONLY NODES
JOINED WITH R@SIBLING POINTERS.
*/

```

```

IF DEBUG = '11'B
  THEN CALL PR_CHAIN;
  ELSE;
LINK = C HEAD;
BB: DO WHILE(LINK
  IF LINK->CHAIN.ID->NODE.R@SIBLING = NULLO
    & LINK->CHAIN.NEXT
    & IS@LEVEL(LINK->CHAIN.NEXT->CHAIN.ID)
      = IS@LEVEL(LINK->CHAIN.ID)
    THEN /* CONNECT ACROSS! */
      LINK->CHAIN.ID->NODE.R@SIBLING =
        LINK->CHAIN.NEXT->CHAIN.ID;
    ELSE IF DEBUG > '00'B
      THEN PUT FILE(DISPLAY) EDIT
        (' NEXT LINK EXAMINED ...')
        (SKIP,A);
      ELSE;
      LINK = LINK->CHAIN.NEXT;
END BB;

```

```

IF DEBUG > '00'B
  THEN PUT FILE(DISPLAY) EDIT
    ('ALL RIGHT SIBLINGS CONNECTED ...')
    (SKIP(2),A);
  ELSE;

```

```

      /* FREE CHAINS STORAGE */

```

```

P1 = C HEAD;
DO WHILE (P1
  PTR=P1->CHAIN.NEXT;
  FREE P1->CHAIN;
  P1 = PTR;
END;
IF DEBUG = '11'B

```

```

THEN PUT FILE(DISPLAY) EDIT
      ('CHAIN FREED WITHIN POINTS R@SIBS')
      (SKIP,A);
ELSE;

      /* SET ALL LEVEL(I) HEADERS. */

      PLACE = 14.5;
      ALLOCATE LEVEL;
      P1 = HEADER;
      LEVEL(1) = P1;
      FINDD: DO UNTIL((P1->NODE.#1@SON = NULLO & P1->NODE.R@SIBLING = NULLO)
        | I > #@NODES + 5);
        I = I + 1;
        IF DEBUG >= '10'B
          THEN CALL LOPOINT(P1);
        ELSE;
        IF P1->NODE.#1@SON = NULLO
          THEN P1 = P1->NODE.R@SIBLING;
        ELSE DO;
          LEV = LEV + 1;
          P1 = P1->NODE.#1@SON;
          LEVEL(LEV) = P1;
        END;
      END FINDD;

      /*****
      INSIDE: PROCEDURE(NODE) RETURNS(BIT(1));
      *****/

      /* IF THE NODE IS ALREADY INSIDE THE CHAIN LINKED-LIST,
      THIS FUNCTION RETURNS A '1'B.
      */

      DECLARE (NODE, LINK) POINTER;

      LINK = C HEAD;
      DO WHILE LINK
        IF NODE = LINK->CHAIN.ID
          THEN RETURN('1'B);
        ELSE LINK = LINK->CHAIN.NEXT;
      END;

      RETURN('0'B);

      END INSIDE;

      /*****
      PR_CHAIN: PROCEDURE;
      *****/

      DECLARE P1 POINTER;

```

```

PUT FILE(DISPLAY) EDIT
  ('BEGIN DUMP OF CHAIN WITHIN POINT PROCEDURE ...')
  (SKIP(2),A);
P1 = C_HEAD;
DO WHILE(P1
  PUT FILE(DISPLAY) EDIT
    (P1->CHAIN.ID->NODE.ID)
    (SKIP,A,A(ID@LENG));
  P1 = P1->CHAIN.NEXT;
END;
PUT FILE(DISPLAY) EDIT
  ('END OF CHAIN DUMP')
  (SKIP,A);

END PR_CHAIN;

END R@SIBS;

/*****
IS@LEVEL: PROCEDURE(P) RETURNS(FIXED BINARY(15));
*****/

/* RETURNS THE LEVEL OF P->NODE BASED ON THE NUMBER
   OF .S IN THE NODE.ID.
   IF P IS NULL, A 0 VALUE IS RETURNED, RATHER THAN ERROR.
*/

DECLARE
  P                POINTER,
  (LEV,I)          FIXED BINARY(15),
  PLACE           FIXED DECIMAL(7,3) EXTERNAL,
  NULLO           POINTER EXTERNAL,
  POS_BLANK       FIXED BINARY(15),
  (INDEX,SUBSTR)  BUILTIN;

PLACE = 10.1;
IF P = NULLO
  THEN RETURN(0);
  ELSE;

                                  /* LOCATE FIRST BLANK CHAR */

POS_BLANK = INDEX(P->NODE.ID, ' ');
IF POS_BLANK <= 1 | INDEX(SUBSTR(P->NODE.ID,1,POS_BLANK), '..') > 0
  THEN CALL ERROR;
  ELSE IF SUBSTR(P->NODE.ID,POS_BLANK-1,1) = '.'
    THEN POS_BLANK = POS_BLANK - 1;
    ELSE;

                                  /* COUNT # OF DOTS AND ADD 1
                                  FOR LEVEL OF NODE.

```


*/

```

LEV = 1;
DO I = 1 TO POS BLANK - 1;
  IF SUBSTR(P->NODE.ID,I,1) = '.'
    THEN LEV = LEV + 1;
  ELSE;
END;

RETURN(LEV);

END IS@LEVEL;

END POINT;
```

5.2.13 PRINT

Node listings are generated when this procedure is invoked. All data defining a node whose address is passed to this procedure are printed for the user. Depending on the value of DEBUG and SHOCST, the user may modify the amount of data printed. If SHOCST is '1'B, the node tells of the various attributes will be listed.

Fig. 5.26 illustrates the internal modular flow of this procedure. The source code is also provided.

Fig. 5.26 The PRINT Procedure

```

/*****/
PRINT: PROCEDURE(P1);
/*****/

/* PRINTS OUT A NODE POINTED TO BY P1 */
/* LAST UPDATE: 3-26-79 */

DECLARE P1      POINTER,
        START   REAL FIXED BINARY(15) INIT(65),
        I       REAL FIXED BINARY(15);

DECLARE
#@ATTRI      REAL FIXED BINARY(15) INIT(1) EXTERNAL,
DEBUG        BIT(2) EXTERNAL,
PROFICI      BIT(3) EXTERNAL,
DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
ID@LENG      REAL FIXED BINARY(15) EXTERNAL,
NULLO        POINTER EXTERNAL,
SHO@CST      BIT(1) INIT('1'B) EXTERNAL,
1 NODE       BASED,
  2 #@I       REAL FIXED BINARY(15),
  2 FATHER    POINTER,
  2 #1@SON     POINTER,
  2 R@SIBLING POINTER, /* SAME LEV ONLY, NOT BRO.*/
  2 #@SONS    REAL FIXED BINARY(15),
  2 OPT@FLAG  BIT(1),
  2 ID        CHAR(65),
  2 COST(#@ATTRI REFER(NODE.#@I))
          REAL FLOAT BINARY(21),
  2 PROB      REAL FLOAT BINARY(21),
  2 UTILITY   REAL FLOAT BINARY(21),
  2 DOWN      POINTER;

DECLARE
PROCESS      EXTERNAL ENTRY,
BUILD        EXTERNAL ENTRY,
SUBTOL       EXTERNAL ENTRY,
GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
LOPOINT      EXTERNAL ENTRY,
LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
LONODE       EXTERNAL ENTRY,
PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
LODOWN       EXTERNAL ENTRY,
ERROR        EXTERNAL ENTRY,
READRAW      EXTERNAL ENTRY,
UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
POINT        EXTERNAL ENTRY;

/* IF DEBUG = '11'B
   THEN PUT FILE(DISPLAY) EDIT
       ('      ** PRINT PROCEDURE INITIATED.')
       (SKIP,A);
   ELSE; */
PUT FILE(DISPLAY) EDIT
(P1->NODE.ID,
 'UTILITY: ',P1->NODE.UTILITY,

```

```

'NUMBER OF SONS: ',P1->NODE.#@SONS)
(SKIP(2),A(ID@LENG),
COL(10),A,E(12,5),
COL(10),A,F(5,0));
IF SHO@CST = '1'B
THEN DO;
    PUT FILE(DISPLAY) EDIT
        ('ATTRIBUTE LEVEL TOLLS: ')(COL(10),A);
    DO I = 1 TO #@ATTRI;
        IF START = 65
            THEN START = 13;
        ELSE START = START + 13;
        PUT FILE(DISPLAY) EDIT
            (P1->NODE.COST(I))(COL(START),E(12,5));
    END;
END;
ELSE;
IF DEBUG > '00'B | PROFICI < '100'B
THEN CALL LOPOINT(P1);
ELSE;
IF P1->NODE.FATHER = NULLO
THEN PUT FILE(DISPLAY) EDIT
    ('THIS IS AN INITIAL OPTION.')(COL(10),A);
ELSE IF P1->NODE.FATHER->NODE.OPT@FLAG = '1'B
THEN PUT FILE(DISPLAY) EDIT
    ('THIS IS AN OPTION NODE OF ITS FATHER.')(
    (SKIP,COL(10),A);
    ELSE PUT FILE(DISPLAY) EDIT
        ('THE PROBABILITY OF THIS NODES ',
        'OCCURRENCE IS ',P1->NODE.PROB)
        (SKIP,COL(10),A,A,E(12,5));
IF P1->NODE.#@SONS > 0
THEN IF P1->NODE.OPT@FLAG = '0'B
    THEN PUT FILE(DISPLAY) EDIT
        ('SONS ARE PROBABILISTIC.')(
        (COL(10),A);
    ELSE PUT FILE(DISPLAY) EDIT
        ('SONS ARE OPTIONS.')(COL(10),A);
ELSE PUT FILE(DISPLAY) EDIT
    ('THIS IS TERMINAL NODE (LEAF) OF TREE.')(
    (SKIP,COL(10),A);

END PRINT;

```

5.2.14 PROCESS

Probably the most logically complex procedure in IDAP is PROCESS, as seen in Fig. 5.27 and the source code. It is the heart of IDAP, and assigns the expected utility values to all nodes in the user's decision tree.

The general procedure followed in PROCESS is explained in the previous section.

Fig. 5.27 The PROCESS Procedure

Fig. 5.28 Part A of the COUNT Module - With PROCESS

Fig. 5.29 Part B of the COUNT Module - Within PROCESS

Fig. 5.30 Part A of the SUMCOST Module - Within Process

Fig. 5.31 Part B of the SUMCOST Module - Within PROCESS

Fig. 5.32 The UPSTART Module - Within PROCESS

Fig. 5.33 The P@ONLY Module - Within PROCESS

Fig. 5.34 Part A of the UPMORE Module - Within PROCESS

Fig. 5.35 Part B of the UPMORE Module - Within PROCESS

Fig. 5.36 The PRESENT Module - Within PROCESS

```

/*****
PROCESS: PROCEDURE(PASSED);
*****/

```

```

/* ASSIGNS EXPECTED UTILITIES TO ALL NODES. */
/* LAST UPDATE: 9-13-80 */

```

DECLARE

```

HEADER          POINTER EXTERNAL,
SENSE           BIT(1) EXTERNAL,
P1             POINTER,
NULLO          POINTER EXTERNAL,
PASSED         CHAR(1),
#@LEAVE        REAL FIXED BINARY(15) INIT(0) EXTERNAL,
#@OPT@NODES    REAL FIXED BINARY(15) INIT(0),
LEAF(@LEAVE)   POINTER CTL,
MAX(@ATTRI)    REAL FLOAT BINARY(21) CTL EXTERNAL,
MIN(@ATTRI)    REAL FLOAT BINARY(21) CTL EXTERNAL,
OPT@NODE(@OPT@NODES) POINTER CTL;

```

DECLARE

```

#@ATTRI        REAL FIXED BINARY(15) EXTERNAL,
COSTS(@LEAVE,@ATTRI) REAL FLOAT BINARY(21) EXTERNAL CTL,
#@LEVEL        REAL FIXED BINARY(15) EXTERNAL,
DEBUG          BIT(2) INIT('11'B) EXTERNAL,
DISPLAY        STREAM OUTPUT PRINT EXTERNAL,
ID@LENG        REAL FIXED BINARY(15) EXTERNAL,
PLACE          REAL FIXED DECIMAL(7,3) EXTERNAL,
PROFICI        BIT(3) INIT('111'B) EXTERNAL,
LEVEL(@LEVEL)  POINTER CTL EXTERNAL,
1 NODE         BASED,
  2 #@I        REAL FIXED BINARY(15),
  2 FATHER     POINTER,
  2 #1@SON     POINTER,
  2 R@SIBLING  POINTER, /* SAME LEV ONLY, NOT BRO.*/
  2 #@SONS     REAL FIXED BINARY(15),
  2 OPT@FLAG   BIT(1),
  2 ID         CHAR(65),
  2 COST(@ATTRI REFER(NODE.#@I))
              REAL FLOAT BINARY(21),
  2 PROB       REAL FLOAT BINARY(21),
  2 UTILITY    REAL FLOAT BINARY(21),
  2 DOWN      POINTER;

```

DECLARE

```

BUILD          EXTERNAL ENTRY,
SUBTOL         EXTERNAL ENTRY,
GEORGE         EXTERNAL ENTRY RETURNS(CHAR(80)),
LOPOINT        EXTERNAL ENTRY,
LOCATE         EXTERNAL ENTRY RETURNS(POINTER),
LONODE         EXTERNAL ENTRY,
PEEL           EXTERNAL ENTRY RETURNS(CHAR(80)),
PRINT          EXTERNAL ENTRY,
LODOWN         EXTERNAL ENTRY,
ERROR          EXTERNAL ENTRY,
READRAW        EXTERNAL ENTRY,

```

```

      UTILOF          EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
      POINT          EXTERNAL ENTRY;

      /* SET UTILITY VALUES BACK TO 0 */
      P1 = HEADER;
      DO WHILE(P1
        P1->NODE.UTILITY = 0;
        P1 = P1->NODE.DOWN;
      END;

      #@LEAVE = 0;
      CALL COUNT;
        PLACE = 20.1;
      CALL SUMCOST;
        PLACE = 20.2;
      IF PASSED = 'P'
        THEN DO;
          CALL UPSTART;
          PLACE = 20.3;
          IF SENSE /* SENSITIVITY ANALYSIS ONLY */
            THEN;
            ELSE CALL PRESENT;
          END;
        ELSE;
      FREE MAX,MIN,LEAF,OPT@NODE,COSTS;

      /*****
      COUNT: PROCEDURE;
      *****/

      /* COUNTS #@LEAVE AND #OPT@NODES, THEN POINTS TO THEM. */

      DECLARE
        I          BIT(1),
        LEV        REAL FIXED BINARY(15),
        P1         POINTER,
        CL         REAL FIXED BINARY(15) INIT(0),
        CN         REAL FIXED BINARY(15) INIT(0);

      TWICE: DO I = '0'B, '1'B;
        IF I = '1'B
          THEN DO;
            ALLOCATE LEAF,OPT@NODE,COSTS;
            IF DEBUG > '00'B
              THEN PUT FILE(DISPLAY) EDIT
                ('NUMBER OF LEAVES IN TREE: ',#@LEAVE,
                 ' NUMBER OF OPT. NODES: ',#OPT@NODES)
                (SKIP,COL(5),2 (A,F(5,0)));
            ELSE;
          END;
        END;

```

```

ELSE;

BY@LEV: DO LEV = #@LEVEL TO 1 BY -1;
IF DEBUG > '00'B
    THEN PUT FILE(DISPLAY) EDIT
        ('COUNTING IN LEVEL ',LEV)
        (SKIP,A,F(5,0));
    ELSE;
P1 = LEVEL(LEV);
IF P1 = NULLO
    THEN GO TO END@LEV;
    ELSE;
BY@SIB: DO UNTIL(P1 = NULLO);
    IF DEBUG > '01'B
        THEN PUT FILE(DISPLAY) EDIT
            ('EXAMINING NEXT NODE.')
            (SKIP,COL(3),A);
        ELSE;

            /* CHECK FOR LEAF */
IF P1->NODE.#1@SON = NULLO
    THEN DO;
        IF I = '0'B
            THEN #@LEAVE = #@LEAVE + 1;
            ELSE DO;
                PLACE = 18;
                CL = CL + 1;
                IF CL > #@LEAVE
                    THEN CALL ERROR;
                    ELSE LEAF(CL) = P1;
            END;
        END;
    ELSE;

        /* CHECK FOR OPTION NODE */
IF P1->NODE.OPT@FLAG = '1'B
    & P1->NODE.#1@SON
    THEN DO;
        IF I = '0'B
            THEN #OPT@NODES = #OPT@NODES + 1;
            ELSE DO;
                PLACE = 19;
                CN = CN + 1;
                IF CN > #OPT@NODES
                    THEN CALL ERROR;
                    ELSE OPT@NODE(CN) = P1;
            END;
        END;
    ELSE;
P1 = P1->NODE.R@SIBLING;
END BY@SIB;
END@LEV: END BY@LEV;

```

END TWICE;

END COUNT;

```

/*****
SUMCOST: PROCEDURE;
*****/

```

/* TRAVERSES TREE FROM LEAF UP, SUMMING COST(I). */

DECLARE

```

P1    POINTER,
(I,J) REAL FIXED BINARY(15);

```

```

    ALLOCATE MAX,MIN;
    MAX(*) = -1E50;
    MIN(*) = 1E50;

```

/* CALCULATE SUM OF COSTS */

```

LEAV: DO I = 1 TO #@LEAVE;
    COSTS(I,*) = 0;
    P1 = LEAF(I);
    UP: DO UNTIL(P1 = NULLO);
        SUM: DO J = 1 TO #@ATTRI;
            COSTS(I,J) = COSTS(I,J) + P1->NODE.COST(J);
        END SUM;
        P1 = P1->NODE.FATHER;
    END UP;
END LEAV;

```

```

DO J = 1 TO #@LEAVE;

```

/* FIND MAX & MIN FOR EACH ATTRIBUTE */

```

FIND@EX: DO I = 1 TO #@ATTRI;
    IF COSTS(J,I) > MAX(I)
        THEN MAX(I) = COSTS(J,I);
    ELSE;
    IF COSTS(J,I) < MIN(I)
        THEN MIN(I) = COSTS(J,I);
    ELSE;

```

```

    END FIND@EX;

```

```

END;

```

```

IF PASSED = 'A'

```

```

    THEN DO;

```

```

        CALL LISTLVS;
        RETURN;

```

```

    END;

```

```

ELSE IF DEBUG > '00'B

```

```

    THEN CALL LISTLVS;
    ELSE;

```

/* CALCULATE UTILITY OF SUM OF COSTS */

```

DO I = 1 TO #@LEAVE;
  LEAF(I)->NODE.UTILITY = UTILOF(I);
  IF DEBUG = '11'B
    THEN CALL PRINT(LEAF(I));
  ELSE;
END;

END SUMCOST;

```

```

/*****
UPSTART: PROCEDURE;
*****/

```

```

/* STARTS AT LEAVES, WORKING UPWARD, MULTIPLYING
UTILITY(I) * PROB(I) AND STORING IT IN
UTILITY(FATHER@OF@I) */

```

```

DECLARE
  P1    POINTER,
  I     REAL FIXED BINARY(15),
  PF    POINTER;

```

```

  PLACE = 21.0;
CALL P@ONLY;
  PLACE = 21.1;
CALL UPMORE;

```

```

/*****
P@ONLY: PROCEDURE;
*****/

```

```

/* STEP ALPHA FROM NOTES OF 9-15-'78 */
/* EVALUATING FROM LEAVES UPWARD */

```

```

DECLARE HOLDER    REAL FLOAT BINARY(21);

```

```

FROM@L: DO I = 1 TO #@LEAVE;
  P1 = LEAF(I);
  HOLDER = P1->NODE.UTILITY;
  PF = P1->NODE.FATHER;
                                /* CALC UTILITIES TO TOP OF TREE
                                OR TO 1ST OPT@NODE ENCOUNTERED. */
  UP: DO WHILE(PF
    IF PF->NODE.OPT@FLAG = '0'B
      THEN DO;
        HOLDER = P1->NODE.PROB * HOLDER;
        PF->NODE.UTILITY = PF->NODE.UTILITY

```

```

        + HOLDER;
    END;
    ELSE LEAVE UP;
    P1 = PF;
    PF = P1->NODE.FATHER;
    END UP;

END FROM@L;

END P@ONLY;

```

```

/*****
UPMORE: PROCEDURE;
*****/

```

```

/* STEP BETA FROM NOTES OF 9-15-'78 */

```

```

DECLARE
    BEST          POINTER,
    HOLDER        REAL FLOAT BINARY(21),
    P2            POINTER;

```

```

FROM@N: DO I = 1 TO #OPT@NODES;
    P1 = OPT@NODE(I);
    IF P1->NODE.#1@SON = NULLO
        THEN GO TO END@N;
    ELSE GET@MAX: DO;

```

```

        /* ASSIGN THE MAX UTIL. OF ALL SONS */
        PF = P1->NODE.#1@SON;
        BEST = PF;
        MAX: DO UNTIL(PF->NODE.FATHER
            | PF = NULLO);
            IF PF->NODE.UTILITY >
                BEST->NODE.UTILITY
            THEN BEST = PF;
            ELSE;
            PF = PF->NODE.R@SIBLING;
            IF PF = NULLO
            THEN LEAVE MAX;
            ELSE;

```

```

        END MAX;
        P1->NODE.UTILITY = BEST->NODE.UTILITY;
        PUT FILE(DISPLAY) EDIT
        ('THE UTILITY VALUE FOR',
        P1->NODE.ID,
        'HAS TAKEN THE VALUE ',
        P1->NODE.UTILITY,
        'FROM THE FOLLOWING ',
        ''BEST'' OPTION: ',
        BEST->NODE.ID)
        (SKIP(2),A,

```



```

                                COL(5),A(ID@LENG),
                                COL(5),A,
                                E(12,5),
                                COL(5),A,A,
                                COL(5),A(ID@LENG));
                                END GET@MAX;

                                /* CALC UTILITIES UP TO ANOTHER */
                                /* OPTION NODE */

P2 = P1;
HOLDER = P2->NODE.UTILITY;
PF = P2->NODE.FATHER;
UPWARD: DO WHILE(PF
                IF PF->NODE.OPT@FLAG = '0'B
                THEN DO;
                    HOLDER = HOLDER * P2->NODE.PROB;
                    PF->NODE.UTILITY = PF->NODE.UTILITY
                    + HOLDER;
                END;
                ELSE LEAVE UPWARD;
                P2 = PF;
                PF = P2->NODE.FATHER;
            END UPWARD;
        END@N: END FROM@N;

    END UPMORE;

    END UPSTART;

    /*****
    PRESENT: PROCEDURE;
    *****/

    /* LISTS LEVEL 1 NODES, AND PICKS OUT THE BEST CHOICE */

    DECLARE
        P1    POINTER,
        BEST  POINTER;

    PUT FILE(DISPLAY) EDIT
        ('***** THE FOLLOWING IS A LIST OF INITITAL OPTIONS *****)
        (SKIP(5),A);
    P1,BEST = LEVEL(1);
    DO UNTIL(P1 = NULLO);
        CALL PRINT(P1);
        IF P1->NODE.UTILITY >= BEST->NODE.UTILITY
            THEN BEST = P1;
        ELSE;
            P1 = P1->NODE.R@SIBLING;
        END;
    END;

    PUT FILE(DISPLAY) EDIT
        ('THE ABOVE LISTING INDICATES THAT THE BEST OPTION IS:',

```

```

        BEST->NODE.ID,
        'IT HAS AN EXPECTED UTILITY OF ',BEST->NODE.UTILITY)
    (SKIP(3),A,
    SKIP,A(ID@LENG),
    SKIP,A,E(12,5));

END PRESENT;

/*****/
LISTLVS: PROCEDURE;
/*****/

/* TO LIST THE COSTS ACCUMULATED AT THE LEAVES AFTER THE
   "AVERAGE OUT" PROCEDURE HAS BEEN EXECUTED. */

DECLARE (I,J)    FIXED BINARY(15);

PUT FILE(DISPLAY) EDIT
    ('THIS IS A LIST OF THE ATTRIBUTE LEVELS ACCUMULATED',
    ' AT THE TERMINAL NODES OF YOUR DECISION TREE.')
    (PAGE,A,SKIP,A);
DO I = 1 TO #@LEAVE;
    PUT FILE(DISPLAY) EDIT
        ('TERMINAL NODE: ',LEAF(I)->NODE.ID)
        (SKIP(3),A,A(65));
    DO J = 1 TO #@ATTRI;
        PUT FILE(DISPLAY) EDIT
            ('LEVEL FOR ATTRIBUTE ',J,': ',
            COSTS(I,J))
            (SKIP,COL(16),A,F(10),A,E(12,5));
    END;
END;

PUT FILE(DISPLAY) EDIT
    ('THE FOLLOWING EXTREMES FOR ATTRIBUTE',
    ' LEVELS ARE FROM OBSERVED DATA.')
    (SKIP(2),A,SKIP,A);
DO I = 1 TO #@ATTRI;
    PUT FILE(DISPLAY) EDIT
        ('MAXIMUM OF ',MAX(I),', AND MINIMUM OF ',MIN(I),
        ' FOR ATTRIBUTE ',I)
        (SKIP,A,E(12,5),A,E(12,5),A,F(10,0));
END;

END LISTLVS;

END PROCESS;

```

5.2.15 READRAW

When the command to build the user's decision tree is input, this procedure is invoked. It serves as the interpreter of the formatted input cards that contain the user's data defining a decision tree. These data may be present in one of the three RAWDAT input files: either RAWDAT1, RAWDAT2, or RAWDAT3. These three files are available so that more than one decision tree may be built and processed during a single execution of IDAP.

When the command to build the decision tree is interpreted, the user must supply the name of the RAWDAT file in which the formatted data for that tree resides. That name will be assigned to the file variable FILENAM.

Fig. 5.37 illustrates the internal modular flow of this procedure; the source code is also provided.

Fig. 5.37 The READRAW Procedure

Fig. 5.38 The STRT@ND Module
- Within READRAW

Fig. 5.39 The MARGINAL Module
- Within READRAW

Fig. 5.40 The GETGEN Module
- Within READRAW

Figure 5.41. The GETUTIL Module - Within READRAW

```

/*****
READRAW: PROCEDURE;
*****/

```

```

DECLARE NODE@COU      REAL FIXED BINARY(15) INIT(0),
P1                    POINTER EXTERNAL,
#UFS                  REAL FIXED BINARY(15) EXTERNAL,
P@UTIL(#UFS)          POINTER CTL EXTERNAL,
START                REAL FIXED BINARY(15),
I                    REAL FIXED BINARY(15),
CARD@IM              CHAR(80) EXTERNAL,
CHAR@1               CHAR(1) DEFINED CARD@IM POS(1),
DISPLAY              STREAM OUTPUT PRINT EXTERNAL,
DEBUG               BIT(2) EXTERNAL,
FILENAM              FILE VARIABLE EXTERNAL,
MAR@COU              REAL FIXED BINARY(15) INIT(0),
#@ATTRI              REAL FIXED BINARY(15) EXTERNAL,
NO@INCR              REAL FIXED BINARY(15) EXTERNAL,
P@MARG(#@ATTRI)      POINTER CTL EXTERNAL,
PLACE                REAL FIXED DECIMAL(7,3) EXTERNAL,
PFIRST               BIT(1),
ERROR                EXTERNAL ENTRY,
CALCK                EXTERNAL ENTRY

                        RETURNS (REAL FLOAT BINARY(21)),
#@LEVEL              REAL FIXED BINARY(15) INIT(12) EXTERNAL,
#@NODES              REAL FIXED BINARY(15) INIT(10) EXTERNAL,
HEADER              POINTER EXTERNAL,
ID@LENG              FIXED BINARY(15) EXTERNAL,
NULLO               POINTER EXTERNAL,
LISTM               EXTERNAL ENTRY,
LEVEL(#@LEVEL)      POINTER CTL EXTERNAL,
1 NODE              BASED,
  2 #@I              REAL FIXED BINARY(15),
  2 FATHER           POINTER,
  2 #1@SON           POINTER,
  2 R@SIBLING        POINTER, /* SAME LEV ONLY, NOT BRO.*/
  2 #@SONS           REAL FIXED BINARY(15),
  2 OPT@FLAG         BIT(1),
  2 ID               CHAR(65),
  2 COST(#@ATTRI REFER(NODE.#@I))
                        REAL FLOAT BINARY(21),
  2 PROB             REAL FLOAT BINARY(21),
  2 UTILITY          REAL FLOAT BINARY(21),
  2 DOWN            POINTER,
1 MARGIN            BASED,
  2 NAME             CHAR(4),
  2 PARM(3)          REAL FLOAT BINARY(21),
1 GENERAL           BASED,
  2 NAME             CHAR(4),
  2 #@INCR           REAL FIXED BINARY(15),
  2 UP@LIM(NO@INCR REFER(GENERAL.#@INCR))
                        REAL FLOAT BINARY(21),

```

```

      2 VALUE(NO@INCR REFER(GENERAL.#@INCR))
      REAL FLOAT BINARY(21),
SUBSTR      BUILTIN,
1 UTILITY   BASED,
      2 NAME      CHAR(3),
      2 #@I       REAL FIXED BINARY(15),
      2 PARM(#@ATTRI REFER(UTILITY.#@I))
      REAL FLOAT BINARY(21),
      2 K         REAL FLOAT BINARY(21),
CST@CRD     CHAR(80) CTL EXTERNAL,
C@1         CHAR(10) DEFINED CST@CRD POS(10),
C@2         CHAR(10) DEFINED CST@CRD POS(20),
C@3         CHAR(10) DEFINED CST@CRD POS(30),
C@4         CHAR(10) DEFINED CST@CRD POS(40),
C@5         CHAR(10) DEFINED CST@CRD POS(50),
P@LAST      POINTER;

ON ENDFILE(FILENAM) BEGIN;
      PUT FILE(DISPLAY) EDIT
      ('END OF RAW DATA CARDS ... ',
      NODE@COU, ' CARDS PROCESSED.')
      (SKIP(3),A,F(6,0),A);
      GO TO ENDD;
      END;
ON CONVERSION BEGIN;
      PUT FILE(DISPLAY) EDIT
      ('CONVERSION ERROR')(SKIP(3),COL(6),A);
      CALL ERROR;
      END;

P@LAST = NULLO;
PUT FILE(DISPLAY) EDIT
('RAW DATA CARDS BEING INPUT ... ',
'TREE CONSTRUCTION INITIATED ... ')
(SKIP(2),A,SKIP,A);

      /* READ IN RAWDATA CARD, SELECT BRANCH */
PFIRST = '0'B;
READ@C: DO WHILE('1'B);
  GET FILE(FILENAM) EDIT
    (CARD@IM)(COL(1),A(80));
  IF CHAR@1
    THEN DO;
      PLACE = 0.02;
      CALL ERROR;
    END;
  ELSE;
    PFIRST = '1'B;
    PLACE = 0.01;
    SELECT(CHAR@1);
      WHEN('N') GO TO E@NODE;
      WHEN('P') GO TO E@PARM;

```



```

        WHEN('M') GO TO E@MARG;
        WHEN('U') GO TO E@UTIL;
        OTHERWISE CALL ERROR;
    END;
    GO TO END@C;
E@NODE:      /* PROCESS NODE-ID CARD */
    IF DEBUG >= '10'B
        THEN PUT FILE(DISPLAY) EDIT
            ('INITIATE PROCESSING OF NODE-ID CARD ...')
            (SKIP(2),A);
        ELSE;
        NODE@COU = NODE@COU + 1;
        PLACE = 1.0;
        IF NODE@COU > #@NODES
            THEN CALL ERROR;
            ELSE CALL STRT@ND;
        GO TO END@C;
E@PARM:      /* PROCESS PARAMETER-SETTING CARD */
    IF DEBUG >= '10'B
        THEN PUT FILE(DISPLAY) EDIT
            ('INITIATE PROCESSING OF PARMETER CARD ...')
            (SKIP(2),A);
        ELSE;
        PLACE = 1.1;
        #@LEVEL = SUBSTR(CARD@IM,2,5);
        #@NODES = SUBSTR(CARD@IM,7,5);
        #@ATTRI = SUBSTR(CARD@IM,12,5);
        #UFS = SUBSTR(CARD@IM,17,5);
        IF DEBUG > '00'B
            THEN PUT FILE(DISPLAY) EDIT
                ('# LEVELS IN TREE = ',#@LEVEL,
                 '# NODES IN TREE <= ',#@NODES,
                 '# ATTRIBUTES = ',#@ATTRI,
                 '# UTILITY FUNCTIONS = ',#UFS)
                (SKIP,4 (COL(10),A,F(5,0)));
            ELSE;
            #@NODES = #@NODES + 1;
            ALLOCATE LEVEL,P@MARG,P@UTIL;
            GO TO END@C;
E@MARG:      /* INPUT PARMS FOR MARGINAL FUNCTION */
    CALL MARGINAL;
    GO TO END@C;
E@UTIL:      /* INPUT PARMS FOR UTILITY FUNCTION */
    CALL GETUTIL;
END@C: END READ@C;
ENDDD: P@LAST->NODE.DOWN = NULLO;

/*****/
STRT@ND: PROCEDURE;
/*****/

/* ASSIGN RAW DATA TO A NODE */

```

```

DECLARE
  #@COST      REAL FIXED BINARY(15) INIT(1),
  I           REAL FIXED BINARY(15),
  P           CHAR(11),
  CST(5)      REAL FLOAT BINARY(21);

PLACE = 2.1;
ALLOCATE NODE SET(P1);
                        /* ASSIGN DOWN POINTER */
IF P@LAST = NULLO
  THEN HEADER = P1;
  ELSE P@LAST->NODE.DOWN = P1;
P@LAST = P1;
  P1->NODE.FATHER,
  P1->NODE.#1@SON,
  P1->NODE.R@SIBLING = NULLO;
P1->NODE.#@SONS,
P1->NODE.UTILITY = 0.;
P1->NODE.ID = SUBSTR(CARD@IM,2,65);
IF SUBSTR(CARD@IM,68,1) = ' '
  THEN P1->NODE.OPT@FLAG = '0'B;
  ELSE P1->NODE.OPT@FLAG = SUBSTR(CARD@IM,68,1);
P = SUBSTR(CARD@IM,70,11); /* PROBABILITY */
IF P = ' '
  THEN;
  ELSE P1->NODE.PROB = P;
START = 5;

                        /* READ IN COST CARDS */
GET: DO WHILE(#@COST <= #@ATTRI);
  GET FILE(FILENAM) EDIT
    (CST@CRD)(COL(1),A(80));

                        /* CHANGE ' 'S TO '0.0'S */
ZERO: DO WHILE('1'B);
  SELECT(' ');
    WHEN (C@1) SUBSTR(CST@CRD,10,3) = '0.0';
    WHEN (C@2) SUBSTR(CST@CRD,20,3) = '0.0';
    WHEN (C@3) SUBSTR(CST@CRD,30,3) = '0.0';
    WHEN (C@4) SUBSTR(CST@CRD,40,3) = '0.0';
    WHEN (C@5) SUBSTR(CST@CRD,50,3) = '0.0';
    OTHERWISE LEAVE ZERO;
  END;
END ZERO;
CST(1) = C@1;
CST(2) = C@2;
CST(3) = C@3;
CST(4) = C@4;
CST(5) = C@5;

                        /* ASSIGN COSTS */
DO I = 1 TO 5 WHILE(#@COST <= #@ATTRI);
  IF START = 5
    THEN START = 1;
    ELSE START = START + 1;

```

```

        P1->NODE.COST(@COST) = CST(START);
        @COST = @COST + 1;
    END;
END GET;

END STRT@ND;

/*****
MARGINAL: PROCEDURE;
*****/

/* READ DATA FOR THE MARGINAL FUNCTION AS SPECIFIED */
/* IN THIS CARD@IM FIELD. */

DECLARE
    ATT@#    REAL FIXED BINARY(15),
    NAME     CHAR(4);

    IF DEBUG >= '10'B
        THEN PUT FILE(DISPLAY) EDIT
            ('INITIATE PROCESSING OF MARGINAL FUNCTION CARD ...')
            (SKIP(2),A);
        ELSE;
        PLACE = 2.19;
        ATT@# = SUBSTR(CARD@IM,2,3);
        NAME = SUBSTR(CARD@IM,5,4);
        PLACE = 2.2;
        MAR@COU = MAR@COU + 1;
        IF MAR@COU > @ATTRI
            THEN CALL ERROR;
        ELSE;
        IF NAME = 'GENE'
            THEN GEN: DO;
                NO@INCR = SUBSTR(CARD@IM,10,10);
                ALLOCATE GENERAL SET(P@MARG(ATT@#));
                P@MARG(ATT@#)->GENERAL.NAME = NAME;
                IF DEBUG = '11'B
                    THEN PUT FILE(DISPLAY) EDIT
                        ('ALLOCATED MARGINAL ',ATT@#)
                        (SKIP,COL(5),A,F(5,0));
                ELSE;
                CALL GETGEN;
            END GEN;
        ELSE N@GEN: DO;
            ALLOCATE MARGIN SET(P@MARG(ATT@#));
            P@MARG(ATT@#)->MARGIN.NAME = NAME;
            CST@CRD = CARD@IM;
            IF C@1 = ' ' THEN C@1 = '0.0';
            IF C@2 = ' ' THEN C@2 = '0.0';
            IF C@3 = ' ' THEN C@3 = '0.0';
            P@MARG(ATT@#)->MARGIN.PARM(1) = C@1;
            P@MARG(ATT@#)->MARGIN.PARM(2) = C@2;
            P@MARG(ATT@#)->MARGIN.PARM(3) = C@3;

```

END N@GEN;

```

/*****/
GETGEN: PROCEDURE;
/*****/

```

/* READ DATA CARDS FOR MARGINAL FUNCTION */

```

DCL INCR@#      REAL FIXED BINARY(15) INIT(1),
  UP(2)         REAL FLOAT BINARY(21),
  VAL(2)        REAL FLOAT BINARY(21),
  I             REAL FIXED BINARY(15),
  START         REAL FIXED BINARY(15) INIT(2);

```

```

GGO: DO WHILE(INCR@# <= NO@INCR);
  GET FILE(FILENAM) EDIT
    (CST@CRD)(COL(1),A(80));
  IF C@1 = ' ' THEN C@1 = '0.0';
  IF C@2 = ' ' THEN C@2 = '0.0';
  IF C@3 = ' ' THEN C@3 = '0.0';
  IF C@4 = ' ' THEN C@4 = '0.0';
  UP(1) = C@1;  VAL(1) = C@2;
  UP(2) = C@3;  VAL(2) = C@4;
  /* ASSIGN UPPER LIMITS & VALUES TO */
  /* THIS GENERAL MARGINAL FUNCTION */
  DO I = 1 TO 2 WHILE(INCR@# <= NO@INCR);
    IF START = 2
      THEN START = 1;
      ELSE START = 2;
    P@MARG(ATT@#)->GENERAL.UP@LIM(INCR@#) = UP(START);
    P@MARG(ATT@#)->GENERAL.VALUE(INCR@#) = VAL(START);
    INCR@# = INCR@# + 1;
  END;
END GGO;

```

END GETGEN;

```

IF DEBUG > '01'B
  THEN CALL LISTM(ATT@#);
ELSE;

```

END MARGINAL;

```

/*****/
GETUTIL: PROCEDURE;
/*****/

```

/* READ DATA CARDS FOR THE UTILITY FUNCTION PARAMETERS */

```

DECLARE
  #@A      REAL FIXED BINARY(15) INIT(1),
  ONE      REAL FLOAT BINARY(21) INIT(1.0000000),
  UTILNO   REAL FIXED BINARY(15) EXTERNAL,
  PARAM(5) REAL FLOAT BINARY(21);

IF DEBUG >= '10'B
  THEN PUT FILE(DISPLAY) EDIT
        ('INITIATE PROCESSING OF UTILITY FUNCTION CARD ...')
        (SKIP(2),A);
  ELSE;
PLACE = 77.1;
UTILNO = SUBSTR(CARD@IM,2,3);
IF UTILNO < 1 | UTILNO > #UFS
  THEN DO;
        PLACE = 77.2;
        CALL ERROR;
  END;
  ELSE;
  ALLOCATE UTILITY SET(P@UTIL(UTILNO));
  P@UTIL(UTILNO)->UTILITY.NAME = SUBSTR(CARD@IM,5,3);
  IF DEBUG > '01'B
    THEN PUT FILE(DISPLAY) EDIT
          ('UTILITY FUNCTION HAS THE FORM: ',
           P@UTIL(UTILNO)->UTILITY.NAME)
          (SKIP,COL(10),A,A(3));
  ELSE;
PLACE = 77.;
SELECT(P@UTIL(UTILNO)->UTILITY.NAME);
  WHEN('ADD');
  WHEN('MUL');
  OTHERWISE CALL ERROR;
END;

/* ASSIGN PARAMETERS TO UTILITY FUNC. */

START = 5;
GOO: DO WHILE(#@A <= #@ATTRI);
  GET FILE(FILENAM) EDIT
    (CST@CRD)(COL(1),A(80));
  IF C@1 = ' ' THEN C@1 = '0.0';
  IF C@2 = ' ' THEN C@2 = '0.0';
  IF C@3 = ' ' THEN C@3 = '0.0';
  IF C@4 = ' ' THEN C@4 = '0.0';
  IF C@5 = ' ' THEN C@5 = '0.0';
  PARAM(1) = C@1;
  PARAM(2) = C@2;
  PARAM(3) = C@3;
  PARAM(4) = C@4;
  PARAM(5) = C@5;
  DO I = 1 TO 5 WHILE(#@A <= #@ATTRI);
    IF START = 5
      THEN START = 1;
    ELSE START = START + 1;
    P@UTIL(UTILNO)->UTILITY.PARM(#@A) = PARAM(START);
    #@A = #@A + 1;

```

```

        END;
    END GOO;
    PLACE = 77.3;
    IF P@UTIL(UTILNO)->UTILITY.NAME = 'MUL'
        THEN DO;

            /* CHECK IF K=1, THEN MUL DEGENERATES TO ADD */
            P@UTIL(UTILNO)->UTILITY.K = CALCK;
            IF P@UTIL(UTILNO)->UTILITY.K < -90.
                THEN MERR: DO;
                    PLACE = 77.4;
                    CALL ERROR;
                    P@UTIL(UTILNO)->UTILITY.NAME = 'ADD';
                END MERR;
            ELSE;
        END;
    ELSE;
    IF (DEBUG='11'B & P@UTIL(UTILNO)->UTILITY.NAME='MUL')
        THEN PUT FILE(DISPLAY) EDIT
            ('UTILITY CONSTANT K IS ',
             P@UTIL(UTILNO)->UTILITY.K)
            (SKIP(2),A,E(12,5));
    ELSE;
    IF DEBUG > '01'B
        THEN PRIN: DO;
            PUT FILE(DISPLAY) EDIT
                ('UTILITY FUNCTION ',P@UTIL(UTILNO)->UTILITY.NAME)
                (SKIP,COL(10),A,A(3));
            DO I = 1 TO P@UTIL(UTILNO)->UTILITY.#@I;
                PUT FILE(DISPLAY) EDIT
                    ('PARAMETER ',I,': ',
                     P@UTIL(UTILNO)->UTILITY.PARM(I))
                    (COL(20),A,F(5,0),A,E(12,5));
            END;
            IF P@UTIL(UTILNO)->UTILITY.NAME = 'MUL'
                THEN PUT FILE(DISPLAY) EDIT
                    ('UTILITY CONSTANT K: ',
                     P@UTIL(UTILNO)->UTILITY.K)
                    (COL(20),A,E(12,5));
            ELSE;
        END PRIN;
    ELSE;
END GETUTIL;

END READRAW;

```

5.2.16 STRIP

When passed a string of 80 characters, this function returns a value that represents the position of the first character before the first blank space after the first non-blank character in that string.

Fig. 5.42 illustrates the internal modular flow of this function; the source code is also provided.

Fig. 5.42 The STRIP Procedure

```

/*****
STRIP: PROCEDURE(CARD) RETURNS(REAL FIXED BINARY(15));
*****/

```

```

/* RETURNS THE POSITION BEFORE THE FIRST BLANK AFTER */
/* THE FIRST NON-BLANK CHARACTER. */

```

```

DECLARE
  CARD      CHAR(80),
  SUBSTR    BUILTIN,
  I         REAL FIXED BINARY(15),
  TRIP      BIT(1) INIT('0'B);

```

```

DO I = 1 TO 80;
  IF SUBSTR(CARD,I,1) = ' '
    THEN IF TRIP
      THEN DO;
        I = I - 1;
        RETURN(I);
      END;
    ELSE;
    ELSE TRIP = '1'B;
  END;
RETURN(-1);

END STRIP;

```

5.2.17 SUBTOL

The user has the option of listing nodes that belong to a specific subtree. Such a subtree is totally defined by a single node; all descendants of that node belong to the same subtree.

When SUBTOL is invoked, the node specified in the user's command serves to define a subtree, and all data of the nodes of that subtree are listed when PRINT is called.

Fig. 5.43 illustrates the flow of SUBTOL, and source code is provided.

Fig. 5.43 The SUBTOL Procedure

Fig. 5.44 The L@SUBT Module within SUBTOL

```

/*****
SUBTOL: PROCEDURE;
*****/

```

```

DECLARE  A    CHAR(80),
        P1    POINTER;

```

```

DECLARE
#@LEVEL      REAL FIXED BINARY(15) INIT(12) EXTERNAL,
#@NODES      REAL FIXED BINARY(15) INIT(10) EXTERNAL,
COMMAND      CHAR(80) EXTERNAL,
DEBUG        BIT(2) INIT('11'B) EXTERNAL,
DISPLAY      STREAM OUTPUT PRINT EXTERNAL,
ID@LENG      REAL FIXED BINARY(15) EXTERNAL,
INDEX        BUILTIN,
NULL0       POINTER EXTERNAL,
PLACE       REAL FIXED DECIMAL(7,3) EXTERNAL,
SUBSTR       BUILTIN,
LEVEL(@LEVEL) POINTER CTL EXTERNAL,
1 NODE      BASED,
  2 #@I      REAL FIXED BINARY(15),
  2 FATHER   POINTER,
  2 #1@SON    POINTER,
  2 R@SIBLING POINTER, /* SAME LEV ONLY, NOT BRO.*/
  2 #@SONS    REAL FIXED BINARY(15),
  2 OPT@FLAG  BIT(1),
  2 ID        CHAR(65),
  2 COST(@ATTRI REFER(NODE.#@I))
              REAL FLOAT BINARY(21),
  2 PROB      REAL FLOAT BINARY(21),
  2 UTILITY    REAL FLOAT BINARY(21),
  2 DOWN      POINTER;

```

```

DECLARE
PROCESS      EXTERNAL ENTRY,
BUILD        EXTERNAL ENTRY,
GEORGE       EXTERNAL ENTRY RETURNS(CHAR(80)),
LOPOINT      EXTERNAL ENTRY,
LOCATE       EXTERNAL ENTRY RETURNS(POINTER),
LONODE       EXTERNAL ENTRY,
PEEL         EXTERNAL ENTRY RETURNS(CHAR(80)),
PRINT        EXTERNAL ENTRY,
LODOWN       EXTERNAL ENTRY,
ERROR        EXTERNAL ENTRY,
READRAW      EXTERNAL ENTRY,
UTILOF       EXTERNAL ENTRY RETURNS(FLOAT BINARY(21)),
POINT        EXTERNAL ENTRY;

```

```

IF DEBUG > '00'B
  THEN PUT FILE(DISPLAY) EDIT
        ('INITIALIZE SUBTREE LISTING.')
        (SKIP(2),A);

```

```

  ELSE;
PLACE = 17;
A = PEEL(COMMAND);
IF A = ' '

```

```

      THEN CALL ERROR;
      ELSE P1 = LOCATE(A);
IF P1 = NULLO
      THEN CALL ERROR;
      ELSE CALL L@SUBT(P1);

```

```

/*****
L@SUBT: PROCEDURE(A);
*****/

```

```

/* LIST THE NODES THAT ARE THE PROGENY OF A->NODE,
   LIST THEM LEVEL BY LEVEL. */

```

```

DECLARE  A      POINTER,
        IDE     CHAR(80),
        LEN     REAL FIXED BINARY(15),
        CTR     REAL FIXED BINARY(15) INIT(0),
        P1      POINTER,
        I       REAL FIXED BINARY(15);

```

```

PUT FILE(DISPLAY) EDIT
  ('**** SUBTREE LISTING OF: ',A->NODE.ID)
  (SKIP(5),COL(5),A,A(ID@LENG));

```

```

LEN = INDEX(A->NODE.ID,' ') - 1;
IF SUBSTR(A->NODE.ID,LEN,1) = ' '
  THEN LEN = LEN - 1;
  ELSE;

```

```

IDE = SUBSTR(A->NODE.ID,1,LEN);

```

```

/* TRAVERSE TREE LEVEL BY LEVEL */

```

```

LEV: DO I = 1 TO #@LEVEL;

```

```

  P1 = LEVEL(I);

```

```

  IF DEBUG > '00'B

```

```

    THEN PUT FILE(DISPLAY) EDIT
      ('INCREMENTING TO LEVEL ',I)
      (SKIP,A,F(5,0));

```

```

  ELSE;

```

```

  IF P1 = NULLO

```

```

    THEN LEAVE LEV;

```

```

  ELSE;

```

```

/* EXAMINE ALONG THAT LEVEL */

```

```

PLACE = 12;

```

```

SIB: DO UNTIL(P1 = NULLO);

```

```

  IF SUBSTR(P1->NODE.ID,1,LEN) = IDE
    THEN DO;

```

```

    IF DEBUG > '01'B

```

```

      THEN PUT FILE(DISPLAY) EDIT
        ('DESCENDANT FOUND..')
        (SKIP,A);

```

```

    ELSE;

```

```

        CALL PRINT(P1);
      END;
    ELSE DO;
      IF DEBUG > '01'B
        THEN PUT FILE(DISPLAY) EDIT
              ('DESCENDANT NOT FOUND.')
              (SKIP,A);
        ELSE;
          CTR = CTR + 1;
          IF CTR > #@NODES
            THEN CALL ERROR;
          ELSE;
            END;
          P1 = P1->NODE.R@SIBLING;
        END SIB;
      END LEV;

      PUT FILE(DISPLAY) EDIT
        ('**** LISTING COMPLETE ****')
        (SKIP(5),COL(5),A);

    END L@SUBT;

  END SUBTOL;

```

5.2.18 UTILOF

Fig. 5.45 illustrates the internal modular flow of the UTILOF procedure, and its source code is provided in the following pages.

This procedure is invoked while the user's decision tree is being processed, after attribute levels have been accumulated at terminal nodes. UTILOF calculates the utility values of each path in the user's decision tree. As explained in Sec. 3, there are seven types of marginal functions, and two types of MUF available in IDAP.

Fig. 5.45 The UTILOF Procedure

Fig. 5.46 The CALC@M Module - Within UTIL@F

```

/*****
UTILOF: PROCEDURE(N) RETURNS(REAL FLOAT BINARY(21));
*****/

```

```

/* RETURNS THE UTILITY VALUE OF THE ARRAY 'COSTS' */
/* OF LEAF NUMBER N. */
/* LAST UPDATE: 4-6-'79 */

```

DECLARE

```

N REAL FIXED BINARY(15),
#@ATTRI REAL FIXED BINARY(15) EXTERNAL,
#@LEAVE REAL FIXED BINARY(15) EXTERNAL,
#UFS REAL FIXED BINARY(15) EXTERNAL,
DEBUG BIT(2) EXTERNAL,
DISPLAY STREAM OUTPUT PRINT EXTERNAL,
EXP BUILTIN,
PLACE REAL FIXED DECIMAL(7,3) EXTERNAL,
ERROR EXTERNAL ENTRY,
I REAL FIXED BINARY(15),
COSTS(#@LEAVE, #@ATTRI)
REAL FLOAT BINARY(21) CTL EXTERNAL,
RET REAL FLOAT BINARY(21);
P@UTIL(#UFS) POINTER CTL EXTERNAL,
P@MARG(#@ATTRI) POINTER CTL EXTERNAL,
1 UTILITY BASED,
2 NAME CHAR(3),
2 #@I REAL FIXED BINARY(15),
2 PARM(#@ATTRI REFER(UTILITY.#@I))
REAL FLOAT BINARY(21),
2 K REAL FLOAT BINARY(21),
1 MARGIN BASED,
2 NAME CHAR(4),
2 PARM(3) REAL FLOAT BINARY(21),
1 GENERAL BASED,
2 NAME CHAR(4),
2 #@INCR REAL FIXED BINARY(15),
2 UP@LIM(NO@INCR REFER(GENERAL.#@INCR))
REAL FLOAT BINARY(21),
2 VALUE(NO@INCR REFER(GENERAL.#@INCR))
REAL FLOAT BINARY(21),
UTILNO REAL FIXED BINARY(15) EXTERNAL,
MAX(#@ATTRI) REAL FLOAT BINARY(21) CTL EXTERNAL,
MIN(#@ATTRI) REAL FLOAT BINARY(21) CTL EXTERNAL,
TWO REAL FLOAT BINARY(21) INIT(2.00000000),
ONE REAL FLOAT BINARY(21) INIT(1.00000000);

```

```

IF P@UTIL(UTILNO)->UTILITY.NAME = 'ADD'
THEN GG: DO;
    IF DEBUG > '01'B
    THEN PUT FILE(DISPLAY) EDIT
        ('UTILITY CALCULATIONS PROCEEDING ',
        'ACCORDING TO ADDITIVE FORM ...')

```

```

                (SKIP,COL(15),A,A);
ELSE;

                /* ADDITIVE FORM OF UTILITY FUNCTION. */
RET = 0.0000000;
DO I = 1 TO #@ATTRI;
    PLACE = 88.;
    RET = RET + P@UTIL(UTILNO)->UTILITY.PARM(I)*CALC@M(I);
END;
END GG;
ELSE MM: DO;
    IF DEBUG > '01'B
        THEN PUT FILE(DISPLAY) EDIT
            ('UTILITY CALCULATIONS PROCEEDING ',
             'ACCORDING TO MULTIPLICATIVE FORM ...')
            (SKIP,COL(15),A,A);
    ELSE;

        /* MULTIPLICATIVE FORM OF UTILITY FUNC. */
RET = ONE;
DO I = 1 TO #@ATTRI;
    PLACE = 88.5;
    RET = RET * (ONE + P@UTIL(UTILNO)->UTILITY.K
                 * P@UTIL(UTILNO)->UTILITY.PARM(I)
                 * CALC@M(I));
END;
    PLACE = 88.6;
    RET = (RET - ONE)/P@UTIL(UTILNO)->UTILITY.K;
END MM;
IF DEBUG = '11'B
    THEN PUT FILE(DISPLAY) EDIT
        ('VALUE FOR UTILITY IS ',RET)
        (SKIP,COL(50),A,E(12,5));
ELSE;
RETURN(RET);

```

```

/*****
CALC@M: PROCEDURE(I) RETURNS(REAL FLOAT BINARY(21));
*****/

```

```

/* TAKES THE VALUE FOR ATTRIBUTE(I) AND PLUGS IT INTO */
/* THE APPROPRIATE MARGINAL FUNCTION.                */

```

```

DECLARE
I      REAL FIXED BINARY(15),
J      REAL FIXED BINARY(15),
RET    REAL FLOAT BINARY(21),
DELTA  REAL FLOAT BINARY(21) INIT(0.00000001),
HOLDER REAL FLOAT BINARY(21);

```



```
IF P@MARG(I)->MARGIN.NAME = 'GENE'
  THEN DO;
```

```
/* GENERAL FORM OF MARGINAL FUNCTION */
```

```
GEN: DO J = 1 TO P@MARG(I)->GENERAL.#@INCR;
      IF COSTS(N,I) <= P@MARG(I)->GENERAL.UP@LIM(J)
        THEN DO;
          RET = P@MARG(I)->GENERAL.VALUE(J);
          GO TO END@CAL;
        END;
      ELSE;
    END GEN;
    PLACE = 99.1;
    CALL ERROR;
  END;
ELSE NON@G: DO;
```

```
/* NON-GENERAL FORM OF MARGINAL FUNCTION */
```

```
IF MAX(I) - MIN(I) < DELTA
  THEN COSTS(N,I) = ONE;
  ELSE COSTS(N,I) =
    (COSTS(N,I)-MIN(I))/(MAX(I)-MIN(I));
PLACE = 99.2;
IF (COSTS(N,I) < 0 | COSTS(N,I) > 1)
  THEN CALL ERROR;
  ELSE;
PLACE = 99.3;
SELECT(P@MARG(I)->MARGIN.NAME);
  WHEN('LIND') GO TO LIND;
  WHEN('LINU') GO TO LINU;
  WHEN('EXPD') GO TO EXPD;
  WHEN('EXPU') GO TO EXPU;
  WHEN('GAMM') GO TO GAMM;
  WHEN('QUAD') GO TO QUAD;
  OTHERWISE CALL ERROR;
END;
END NON@G;
```

```
EXPD: /* EXPONENTIAL-DESIREABLE MARGINAL FUNC. */
```

```
RET = ONE - EXP(-P@MARG(I)->MARGIN.PARM(1) * COSTS(N,I));
RET = RET/(ONE-EXP(-P@MARG(I)->MARGIN.PARM(1)));
GO TO END@CAL;
```

```
EXPU: /* EXPONENTIAL-NONDESIREABLE MARGINAL */
```

```
RET = ONE-EXP(-P@MARG(I)->MARGIN.PARM(1)*(ONE-COSTS(N,I)));
RET = RET/(ONE-EXP(-P@MARG(I)->MARGIN.PARM(1)));
GO TO END@CAL;
```

```
LIND: /* LINEAR-DESIRABLE MARGINAL FUNCTION */
```

```
RET = COSTS(N,I);
GO TO END@CAL;
```

```
LINU:          /* LINEAR-UNDESIRABLE MARGINAL FUNCTION */
```

```
RET = 1. - COSTS(N,I);
GO TO END@CAL;
```

```
GAMM:          /* GAMMA FORM OF MARGINAL FUNCTION */
```

```
RET = COSTS(N,I)/P@MARG(I)->MARGIN.PARM(1);
RET = RET/P@MARG(I)->MARGIN.PARM(2);
RET = RET**P@MARG(I)->MARGIN.PARM(1);
HOLDER = P@MARG(I)->MARGIN.PARM(1)*P@MARG(I)->MARGIN.PARM(2);
HOLDER = (COSTS(N,I) - HOLDER)/P@MARG(I)->MARGIN.PARM(2);
HOLDER = EXP(-HOLDER);
RET = RET * HOLDER;
GO TO END@CAL;
```

```
QUAD:          /* QUADRATIC FORM OF MARGINAL FUNCTION */
```

```
RET = P@MARG(I)->MARGIN.PARM(1);
RET = RET + TWO * P@MARG(I)->MARGIN.PARM(2) * COSTS(N,I);
HOLDER = COSTS(N,I)*COSTS(N,I)*P@MARG(I)->MARGIN.PARM(3);
RET = RET - HOLDER;
```

```
END@CAL: IF DEBUG = '11'B
      THEN PUT FILE(DISPLAY) EDIT
            ('VALUE OF ',RET,
             'WAS RETURNED FOR MARGINAL NUMBER ',I)
            (SKIP(2),A,E(12,5),
             COL(5),A,F(5,0));
      ELSE;
      RETURN(RET);
```

```
END CALC@M;
```

```
END UTILOF;
```

5.3 SYSTEM REQUIREMENTS

The following table summarizes typical statistics of IDAP's execution. The data are the averages of the three examples shown in Sec. 4.

Required Memory	250 K
SYSMMSG File	95 lines
DISPLAY File	653 lines (DEBUG='11'B)
SYSPRINT File	0 lines
CPU time	0 seconds
WAIT Time	3.7 seconds
CPU + WAIT Time	3.7 seconds
CPU/WAIT Time	0
WAIT*CORE	9 100KBSEC
CPU*CORE	1 100KBSEC
CHARGE AT CURRENT ANL RATES	\$0.35

The number of lines in the DISPLAY file and the required memory vary, because they depend on the user's preferences of output volume and on the number of nodes in the user's decision tree, respectively.

The user may adjust the amount of memory allocated for IDAP's execution by changing the appropriate parameter on the JOB card when the job is submitted for execution. This amount of memory may be estimated from the number of nodes present in the tree and the following table.

Variable in NODE Structure	Required Storage (Bytes)
#I	2
FATHER	4
#1SON	4
RSIBLING	4
#SONS	2
OPTFLAG	1
ID	65
COST	4*(no. of attributes)
PROB	4
UTILITY	4
DOWN	4

Total:	94 + 4*(no. of attributes)

If, for example, the user's decision tree contains 1000 nodes each with three attributes, then, at 1024 bytes/K, the additional storage required is about 104 K. This additional requirement must be added onto the base of 250 K. Note that the above estimates should be used as rough guides only.

5.4 JCL AT ARGONNE

At the present time, load modules for each of the external procedures are stored in the load library C190.B99832.IDAWORK, and the complete executable load module for the current version of IDAP is in C190.B99832.LOADERS(IDA).

The JCL required to execute IDAP is shown below; it invokes the member IDAP from the appropriate PDS.

```
//IDA JOB (L99832,5,,2),REGION=250K,CLASS=A,PRTY=N
... CUA card here ...
//*MAIN ORG=RADS14
//*
//*
//*
//STEP1 EXEC PGM=IDA
//STEPLIB DD DSN=C190.B99832.LOADERS,DISP=SHR
//*
//*
//***** DATA FOR NODES OF TREE
//
//RAWDAT1 DD *
```

user's formatted RAWDAT input data here

```
//RAWDAT2 DD DUMMY
//RAWDAT3 DD DUMMY
//*
//*
//***** USERS COMMANDS
//
//USER DD *
```

user's "commands" here

```
//DISPLAY DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
/*
```

Note the presence of the DISPLAY, RAWDAT and USER DD statements in the above listing. BLKSIZE on the DISPLAY output file may be increased, if IDAP is not being debugged, because of the program's error-handling routines that preclude most ABENDS. If the user wishes to generate more than one decision tree during a single execution of IDAP, the formatted input data for these trees may be placed in any of the RAWDAT files. Otherwise, unused RAWDAT files should be "dummied" out as shown above.

5.5 THE IDAIN PROCEDURE

IDAIN is a procedure written in WYLBUR EXEC language. Such a procedure is well suited for the long wait-time and the simple I/O requirements encountered when the user is interactively creating the data for a decision tree.

This procedure prompts the user for all the data of a decision tree, formats the data, and allows the user to edit them. Finally, it saves the formatted data as a file suitable for input to the IDAP PL/1 program.

The techniques of the program are straightforward, except for the use of the "EXE ACT LINES a/b NODEL CLE" statements. Such statements are necessary because the entire IDAIN program is too large to be placed in the user's exec file; the program must be placed in the user's active file, and only one part at a time of that file executed.

To initiate execution of IDAIN, the user, after logging onto WYLBUR, must use the dataset B99832.IDA.EXEC and then type in "exe act lines 1/1 node l cle." Note that UPPER CASE must be in effect. A complete example is provided later in this document. If any difficulties arise, please contact the authors.

The following listing was used as input to a WYLBUR pre-editor* to generate IDAIN, and is not executable by WYLBUR. It contains symbolic labels that were resolved and removed by the pre-editor. One reason that this code is listed here is because the internal modular flow is more easily seen in the pre-edited form; comments and labels are present here, but not in the edited executable file.

```

; WYLBUR EXEC FILE FOR CREATING AN INPUT FILE FOR
; THE IDA PROGRAM. WRITTEN BY MARK J. JUSKO.
EXE ACT LINES INTRO/ENDINTRO NODEL CLE
BLNK::  BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK
        BLANKBLANK

```

*The pre-editor used to generate IDAIN was written by the authors at ANL, who may be contacted for more complete explanation of that procedure. This pre-editor is itself a WYLBUR EXEC procedure that operates on a file to remove comments and resolve symbolic label references.

ENDBLNK:: BLANKBLANK

INTRO:: COMM THIS IS A PROC FOR CREATING AN INPUT FILE FOR THE
COMM IDA PROGRAM. YOU WILL BE PROMPTED TO ENTER A
COMM NUMBER OF THINGS FOR THIS FILE. REMEMBER TO PRESS
COMM THE CARRIAGE-RETURN KEY SO YOUR ENTRY WILL BE
COMM PROCESSED.

; ;

; GET OLD FILE

SET ESC %

C 'BLANKBLANK' TO '' IN BLNK/ENDBLNK NOL

COPY BLNK TO 500

COMM

COMM DO YOU HAVE A FILE TO WHICH YOU WANT TO ADD DATA? ...

COMM (YOU MAY TYPE IN "DUMMY");

REA STR SO PRO ' IF SO, ENTER THAT FILE NAME, ELSE CR: '

IF (SO EQ '') EXE NODS

IF (SO EQ 'DUMMY') COPY BLNK/ENDBLNK TO END

IF (SO EQ 'DUMMY') EXEC POIN

; HAVE OLD DS

COPY FROM %SO TO 500

COPY BLNK/ENDBLNK TO END

POIN:: POINT

SET VAL W0=*

SET VAL W1=L

SET VAL N2=W1-499

COMM THERE ARE %N2 LINES IN YOUR ACTIVE FILE NOW.

COMM FROM WHAT POINT WOULD YOU LIKE TO CONTINUE ...

COMM ENTER '1' FOR MARGINAL FUNCTIONS,

COMM '2' FOR UTILITY FUNCTIONS, OR

COMM '3' FOR NODE DATA.

READ VAL NO PROMPT 'ENTER 1, 2 OR 3, OR CR: '

IF (NO EQ 1) EXE ACT LINES MARG/ENDMARG NODEL CLE

IF (NO EQ 2) EXE ACT LINES UTIL/ENDUTIL NODEL CLE

IF (NO EQ 3) EXE ACT LINES NODE/ENDNODE NODEL CLE

; WANT NEW DATASET

NODS:: COPY BLNK/ENDBLNK TO END

COMM EXACTLY HOW MANY LEVELS WILL THERE BE

READ STRING SO PROMPT ' IN THE TREE? '

SET VAL W0=L-10

C 1 TO 'P%SO' IN %W0 NOL

COMM WHAT IS THE UPPER LIMIT OF THE NUMBER OF NODES

READ STRING SO PROMPT ' IN THE TREE? '

C 7 TO '%SO' IN %W0 NOL

COMM EXACTLY HOW MANY ATTRIBUTES WILL THERE BE

READ STRING SO PROMPT ' ASSOCIATED WITH EACH NODE? '

C 12 TO '%SO' IN %W0 NOL

SET VAL N9=SO

COMM HOW MANY UTILITY FUNCTIONS (UPPER LIMIT) WILL

READ STRING SO PROMPT ' YOU WANT? '

C 17 TO '%SO' IN %W0 NOL

SET VAL W0=W0+1

; N9=# OF ATTRIBUTES

ENDINTRO:: EXE ACT LINES MARG/ENDMARG NODEL CLE

; ;

; GET MARGINAL FUNCTIONS

MARG:: SET VAL NO=0

LMARG:: SET VAL NO=NO+1

IF (NO NE 1) EXE NOEX

LIMARG:: READ STRING SO PROMPT 'EXPLANATION FOR MARGINAL FUNCTION? '

IF (INDEX('%SO','N') GE 1) EXE NOEX

COMM Marginal functions in this procedure can take

COMM any of the following forms:

COMM

COMM NAME	CODE	FORM	PARAMETER 1	2	3
COMM GENERAL	GENE	specify upper limits and values in the 'a' intervals.	a	0	0
COMM LINEAR	LINE	$M = X'$	0	0	0
COMM EXPONENTIAL (DESIREABLE)	EXPD	$M = \frac{1 - \exp(-rX')}{1 - \exp(-r)}$	r	0	0
COMM EXPONENTIAL (UNDESIREABLE)	EXPU	$M = \frac{1 - \exp(-r(1-X'))}{1 - \exp(-r)}$	r	0	0
COMM GAMMA	GAMM	$M = \frac{(X')^a}{(ab)} * \exp(-\frac{X' - ab}{b})$	a	b	0
COMM QUADRATIC	QUAD	$M = a + 2bX' - c(X')^2$	a	b	c

COMM

COMM

COMM GENERAL

GENE

specify upper limits and values in the 'a' intervals.

a 0 0

COMM

COMM LINEAR

LINE

$M = X'$

0 0 0

COMM

COMM EXPONENTIAL

(DESIREABLE)

EXPD

$M = \frac{1 - \exp(-rX')}{1 - \exp(-r)}$

r 0 0

COMM

COMM EXPONENTIAL

(UNDESIREABLE)

EXPU

$M = \frac{1 - \exp(-r(1-X'))}{1 - \exp(-r)}$

r 0 0

COMM

COMM

COMM

COMM

COMM

COMM GAMMA

GAMM

$M = \frac{(X')^a}{(ab)} * \exp(-\frac{X' - ab}{b})$

a b 0

COMM

COMM

COMM

COMM

COMM

COMM QUADRATIC

QUAD

$M = a + 2bX' - c(X')^2$

a b c

194

COMM

COMM

COMM

COMM In all cases, $0 \leq X' \leq 1$ because

COMM

COMM

$X = \min(X_i)$

COMM

$X' = \frac{X - \min(X_i)}{\max(X_i) - \min(X_i)}$

COMM

COMM

COMM

COMM The following illustrates a GENERAL marginal

COMM function where PARAMETER 1 IS 3.

COMM

COMM

COMM

$M(x)$

COMM

This function is defined on the interval
0,1 and takes on values between 0.25
and 1.0.

```

NOEX::  IF (N0 GT N9) EXE MQUIT
        ; S9=NAME OF MARGINAL FUNCTION
        C 1/5 TO 'M%N0' IN %WO NOL
        COMM
        COMM WHAT IS THE FOUR-CHARACTER NAME OF THE
        READ S'RING S9 PROMPT '      %NO TH MARGINAL FUNCTION? '
        IF (S9 EQ 'GENE') EXE MOK
        IF (S9 EQ 'LINE') EXE MOK
        IF (S9 EQ 'EXPD') EXE MOK
        IF (S9 EQ 'EXPU') EXE MOK
        IF (S9 EQ 'GAMM') EXE MOK
        IF (S9 EQ 'QUAD') EXE MOK
        COMM SORRY, TRY AGAIN ...
        EXE LMARG
MOK::   C 5/8 TO '%S9' IN %WO NOL
        IF (S9 EQ 'GENE') EXE GENE SAVE
        IF (S9 EQ 'GENE') EXE LMARG
        READ STRING S0 PROMPT 'WHAT IS PARAMETER 1? '
        C 10 TO '%S0' IN %WO NOL
        READ STRING S0 PROMPT 'WHAT IS PARAMETER 2? '
        C 20 TO '%S0' IN %WO NOL
        READ STRING S0 PROMPT 'WHAT IS PARAMETER 3? '
        C 30 TO '%S0' IN %WO NOL
LMARG:: SET VAL W0=W0+1
        IF (W0 GT L) COPY BLNK/ENDBLNK TO END
        EXE LMARG
MQUIT:: COMM NOW THE INPUT FOR THE %N9 MARGINAL FUNCTIONS
        COMM IS COMPLETE.
        EXE ACT LINES UTIL/ENDUTIL NODEL CLE
        ;
        ; GENERAL MARGINAL FUNCTION
        ; N8=# INCREMENTS IN THIS GENE FUNCTION
GENE::  COMM
        COMM HOW MANY INCREMENTS WILL THERE BE FOR
        READ VAL N8 PROMPT '      THIS MARGINAL FUNCTION? '
        C 10 TO '%N8' IN %WO NOL
        C 20 TO '0.0' IN %WO NOL
        C 30 TO '0.0' IN %WO NOL

```



```

; GET THE N8 UPPER LIMITS AND THEIR VALUES
SET VAL W0=W0+1
IF (W0 GT L) COPY BLNK/ENDBLNK TO END
SET VAL N5=0
SET VAL N1=-10
SET VAL N2=0
SLOOP:: SET VAL N5=N5+1
        IF (N5 GT N8) EXE ENDLOOP
        IF (N1 GE 30) EXE AT20
        ; START IN COL 20
        SET VAL N1=N1+20
        SET VAL N2=N2+20
        EXE GOK
AT20::  SET VAL N1=10
        SET VAL N2=20
        SET VAL W0=W0+1
        IF (W0 GT L) COPY BLNK/ENDBLNK TO END
GOK::   READ STRING S0 PROMPT 'WHAT IS THE UPPER LIMIT OF RANGE %N5? '
        READ STRING S1 PROMPT 'WHAT IS THE VALUE IN THAT RANGE? '
        COMM
        C %N1 TO '%S0' IN %W0 NOL
        C %N2 TO '%S1' IN %W0 NOL
        EXE SLOOP
ENDLOOP:: COPY BLNK/ENDBLNK TO END
POINT
SET VAL W0=*
ENDMARG:: EXE RETURN ; END OF GENE FUNCTION
; ;
; UTILITY FUNCTION
UTIL::  COMM
        READ STRING S0 PROMPT 'EXPLANATION FOR UTILITY FUNCTION? '
        IF (INDEX('%S0','N') GE 1) EXE UNOEX
        COMM
        COMM The utility function  $U = U(X_1, X_2, X_3, \dots, X_n)$  may be one of two forms:
        COMM
        COMM
        COMM The Additive form (code: ADD):
        COMM
        COMM
        COMM (1) 
$$U = \sum_{i=1}^n K_i * M_i(X_i) \quad \text{when } \sum_{i=1}^n K_i = 1$$

        COMM
        COMM
        COMM
        COMM The Multiplicative form (code: MUL):
        COMM
        COMM
        COMM (2) 
$$U = \frac{1}{K} \left( \prod_{i=1}^n (1 + K * K_i * M_i(X_i)) - 1 \right) \quad \text{when } \sum_{i=1}^n K_i = 1$$

        COMM
        COMM
        COMM
        COMM and  $1 + K = \prod_{i=1}^n (1 + K * K_i)$ 
        COMM
        COMM
        COMM In both cases,  $M_i(X_i)$  is the  $i$ th marginal function

```

```

      COMM      operating on the value of attribute i.
      COMM In both cases, 0 <= U <= 1.
      COMM
UNOEX::  COMM
      COMM WHAT IS THE NUMBER OF THE NEXT UTILITY FUNCTION ...
      READ STRING S0 PROMPT '      TYPE 'END' TO STOP: '
      IF (S0 EQ 'END') EXE UQUIT
      C 1/4 TO 'U%S0' IN %W0 NOL
      COMM WHAT IS THE THREE CHARACTER NAME OF THE
      READ STRING S0 PROMPT '      UTILITY FUNCTION? '
      IF (S0 EQ 'ADD') EXEC UUOK
      IF (S0 EQ 'MUL') EXE UUOK
      COMM ERROR, PLEASE TRY AGAIN ...
      EXE UTIL
UUOK::  C 5/7 TO '%S0' IN %W0 NOL
      SET VAL W0=W0+1
      IF (W0 GT L) COPY BLNK/ENDBLNK TO END
      ; N1= COL NUMBER
      ; N2= ATTRIBUTE NUMBER
      SET VAL N1=0
      SET VAL N2=0
ULOOP:: SET VAL N2=N2+1
      IF (N2 GT N9) EXE UNOEX
      SET VAL N1=N1+10
      IF (N1 LT 60) EXE UOK
      ; INCREMENT LINE AND COLUMN
      SET VAL W0=W0+1
      IF (W0 GT L) COPY BLNK/ENDBLNK TO END
      SET VAL N1=10
UOK::  COMM WHAT IS THE %N2 TH PARAMETER IN
      READ STRING S0 PROMPT '      THE UTILITY FUNCTION? '
      C %N1 TO '%S0' IN %W0 NOL
      EXE ULOOP
UQUIT:: COPY BLNK/ENDBLNK TO END
      POINT
      SET VAL W0=*
      COMM NOW THE UTILITY FUNCTION DATA INPUT IS COMPLETE.
ENDUTIL:: EXE ACT LINES NODE/ENDNODE NODEL CLE
      ;
      ; ***** NODE DATA CARDS
NODE::  SET VAL N0=W0
      READ STRING S0 PROMPT 'EXPLANATION FOR NODE INPUT? '
      IF (INDEX('%S0','Y') EQ 0) EXEC NNOEX
      COMM
      COMM IN THE REMAINDER OF THIS SESSION, THE TERM 'NODE
      COMM DESIGNATION' WILL BE USED FREQUENTLY. YOU WILL BE
      COMM PROMPTED TO ENTER A CHARACTER STRING TO BE USED IN
      COMM THE DESIGNATION OF EACH NODE. THIS STRING WILL BE
      COMM USED BY THE COMPUTER FOR THE IDENTIFICATION OF THE
      COMM NODE IN QUESTION, AS WELL AS BY YOU THE USER. IT
      COMM MUST CONTAIN THE FOLLOWING:
      COMM
      COMM      - THE HIERARCHICAL REPRESENTATION OF THE
      COMM      NODE (SUCH AS 1.3.2) STARTING IN COLUMN

```

```

COMM          ONE, WHICH MAY NOT CONTAIN BLANKS.
COMM
COMM          - AT LEAST ONE BLANK FOLLOWING THIS
COMM          HIERARCHICAL REPRESENTATION.
COMM
COMM          - ANY NOTATION DESIRED BY YOU THE USER,
COMM          WHICH WILL BE IGNORED BY THE COMPUTER.
COMM          IT MAY NOT CONTAIN QUOTATION MARKS.
COMM
COMM IN ALL CASES, THIS NODE DESIGNATION WILL BE
COMM TRUNCATED TO 65 CHARACTERS.
COMM
NNOEX::      C 1/1 TO 'N' IN %W0 NOL
COMM
READ STRING S0 PROMPT 'WHAT IS THE NEXT NODE DESIGNATION? '
SET VAL S0=S0||' '
IF (SUBSTR('%S0',1,3) EQ 'END') EXEC NQUIT
; VERIFY THE NODE DESIGNATION
SET VAL N3=INDEX(S0,' ')-1
SET VAL S1=SUBSTR(S0,1,N3)
POINT '%S1' (1) IN %N0/L NOL
IF (* EQ -1) EXE OK
; THERES A REPEAT!
COMM THERE MAY BE AN ERROR, CHECK TO SEE IF THESE ARE RIGHT:
POINT '%S1' IN %N0/L
OK::        C 2 TO '%S0' IN %W0 NOL
SET VAL S1='REGARDING THE ABOVE NODE,'
COMM
REPT::      COMM %G0
READ STRING S2 PROMPT '%S1 ARE ITS DESCENDANTS PROB. OR OPT.?'
IF (SUBSTR('%S2',1,1) EQ 'P') EXEC C680
IF (SUBSTR('%S2',1,1) EQ 'O') EXEC C681
COMM ERROR, PLEASE TRY AGAIN. ENTER EITHER 'P' OR 'O'.
EXE REPT
C680::      C 68 TO 'O' IN %W0 NOL
EXE PF
C681::      C 68 TO '1' IN %W0 NOL
PF::        COMM %S1 IF IT IS PROBABALISTIC FROM ITS FATHER, ENTER
READ STRING S0 PROMPT 'THAT PROBABILITY, OTHERWISE PRESS ''CR'': '
C 70 TO '%S0' IN %W0 NOL
SET VAL W0=W0+1
IF (L LE %W0) COPY BLNK/ENDBLNK TO END
; INPUT VALUES FOR ALL ATTRIBUTES
; N5=COLUMN NO., N1=ATT #, S9=VALUE OF ATT
SET VAL N5=0
SET VAL N1=0
NINCR::     SET VAL N5=N5+10
SET VAL N1=N1+1
IF (%N5 LE 50) EXEC N1
SET VAL W0=W0+1
IF (L LE %W0) COPY BLNK/ENDBLNK TO END
SET VAL N5=10
N1::        IF (N1 GT N9) EXEC N2 ; DONE WITH PARAM INPUT
READ VAL S9 PROMPT '%S1 WHAT IS VALUE OF ATTRIBUTE %N1? '

```

```

      C %N5 TO '%S9' IN %W0 NOL
      EXEC NINCR
N2::   IF (%N5 EQ 10) EXEC N3
      EXE N4
N3::   IF (%N1 GT %N9) EXE NNOEX
N4::   SET VAL W0=W0+1
      IF (L LE %W0) COPY BLNK/ENDBLNK TO END
      EXE NNOEX
NQUIT:: COMM COMPLETED CONSTRUCTION OF NODE DATA CARDS.
      POINT
      IF (* NE -1) DEL */L
      READ STRING S0 USING L COL 2/2
      IF (S0 EQ ' ') DEL L
      NUM 500/L START 500
      COMM
      COMM COMPLETED NODE DATA INPUT.
ENDNODE:: EXEC ACT LINES MOD/QUITS NODEL CLE
      ;
      ; MODIFICATIONS ALLOWED NOW.
MOD::  COMM
      COMM NOW YOU HAVE TWO OPTIONS.
      COMM ENTER '1' IF YOU WISH TO SAVE THIS DATASET & TERMINATE THIS SESSION.
      COMM      '2' IF YOU WISH TO MODIFY OR SEE THIS DATASET.
      READ VAL NO PROMPT 'ENTER 1 OR 2: '
      IF (NO EQ 1) EXE ENDMOD
      ; LIMITED EDITING CAPABILITIES
MCONT:: COMM
      COMM TYPE IN 'END' IF YOU WISH TO END THIS MODIFICATION
      COMM      PROCESS AND CONTINUE ON.
      READ STRING S0 PROMPT 'DO YOU WANT TO SEE THIS DATASET? '
      IF (S0 EQ 'END') EXE ENDMOD
      IF (INDEX('%S0','N') GE 1) EXE MOVE
      NUM 500/L START 500
      COMM
      COMM
      L 500/L
      EXE MCONT
MOVE:: COMM DO YOU WISH TO MOVE LINES OF THIS DATASET
      READ STRING S0 PROMPT '      TO OTHER POSITIONS? '
      IF (S0 EQ 'END') EXE ENDMOD
      IF (INDEX('%S0','N') GE 1) EXE DEL
      COMM OK, FIRST ENTER THE CONTIGUOUS LINE NUMBERS TO
      COMM      BE MOVED, THEN ENTER THE LINE AFTER WHICH
      COMM      THEY ARE TO BE INSERTED.
      READ STRING S0 PROMPT 'LINE NUMBER(S) TO BE MOVED: '
      READ VAL W0 PROMPT '%S0 WILL BE INSERTED AFTER: '
      SET VAL W0=W0+.001
      MOVE %S0 TO %W0
      EXE MCONT
DEL::  READ STRING S0 PROMPT 'DO YOU WISH TO DELETE LINES? '
      IF (S0 EQ 'END') EXE ENDMOD
      IF (INDEX('%S0','N') GE 1) EXE MCONT
      READ STRING S0 PROMPT 'WHICH LINES SHOULD BE DELETED? '
      DEL %S0

```

```
EXE MCONT
ENDMOD:: COMM DONE WITH MODIFICATIONS.
        READ STRING SO PROMPT 'DO YOU WANT TO SAVE THIS DATASET? '
        IF (INDEX(SO,'N') GE 1) EXE MOD
        ; SAVE DATASET
        READ STRING SO PROMPT 'TO BE SAVED AS WHAT, AND VOLUME: '
        S %SO CARD LINES 500/L
        COMM
        READ STRING SO PROMPT 'DO YOU WANT THIS DATASET PUNCHED? '
        IF (INDEX(SO,'Y') GE 1) PUNCH 500/L LIST
        COMM
QUITS:: COMM DONE!!
```

6 IDAP'S ERROR MESSAGES

Associated with each type of error is an error number, the approximate procedure in which this error may have been encountered, and the error message text itself. Note that in future versions of IDAP the program may be able to recover from errors, as anticipated by the "REC?" column in the following table. At the present time, however, no recovery is possible and the "REC?" column should be disregarded.

An error number is simply the value the variable PLACE had when the error occurred. The value of PLACE is changed throughout the program, and the only time it is printed is when an error occurs. The third column of the following table lists the name of the procedure in which this variable was changed.

The fourth column in the following table contains the error message text. It indicates a possible explanation of the source of the error. Some of the error messages include the statement "PLEASE NOTIFY"; when such an error occurs, the user should contact the author as soon as possible, supplying complete listings. It is hoped that through the interaction of programmers and users, IDAP will develop into a more useful tool. There is always the possibility that certain errors occur due to programming, not user, problems.

ONCODE is a PL/1 built-in variable. When an error occurs, its value is printed in the DISPLAY output file; the user may consult IBM's PL/1 documentation for a complete explanation of the different values associated with ONCODE. The reason this variable is included in IDAP is that this program invokes the ERROR procedure (which terminates the program) not only when an anticipated error is uncovered (such as a wrong command), but also when the PL/1 processor discovers an error (such as overflow or underflow). The variable ONCODE may be necessary when studying these later errors.

Table 6.1 IDAP'S Error Messages; (Last Update: 11-7-'78)

Error Number	REC ?	Description	Message
0.01	NO	IN READRAW	INVALID CHARACTER IN RAWDAT FILE.
0.02	NO	IN READRAW	FIRST RAWDAT INPUT CARD IS NOT A PARAMETER SETTING CARD.
0.5	NO	IN MAIN	INVALID COMMAND.
0.51	YES	IN MAIN	INVALID 'LIST' OPTION IN COMMAND.
0.52	NO	IN BUILD	LACK OF SUBSTRING IN "BUILD" COMMAND.
0.53	NO	IN BUILD	INVALID SUBSTRING IN "BUILD" COMMAND.
0.6	YES	IN MAIN	INVALID SUBSTRING FOR UTILITY REFERENCE NUMBER.
0.61	YES	IN MAIN	NO VALUE SPECIFIED IN 'SET' COMMAND.
0.62	YES	IN MAIN	INVALID 'SET' OPTION.
0.7	YES	IN MAIN	INVALID SUBSTRING IN "LIST MARG" COMMAND.

Table 6.1 (Cont'd)

Error Number	REC ?	Description	Message
0.8	YES	IN MAIN	INVALID SUBSTRING IN "LIST UTIL" COMMAND.
1.0	NO	IN READRAW	QUANTITY OF NODES INPUT EXCEEDS THE PRE-SPECIFIED UPPER LIMIT.
1.01	NO	IN ALTER	LACK OF SUBSTRING IN "ALTER" COMMAND.
1.1	NO	IN READRAW	ERROR IN OBTAINING TREE PARAMETER.
		IN ALTER	CONVERSION ERROR - CHECK SUBSTRING IN ALTER COMMAND.
1.2	NO	IN ALTER	INVALID NODE-HIERARCHY CODE.
2.1	NO	IN READRAW	ALLOCATION IF NODE FAILED; MORE SPACE REQUIRED.
2.19	NO	IN READRAW	ERROR IN OBTAINING MARGINAL FUNCTION PARAMETER.
2.2	NO	IN READRAW	VALUE OF MARGINAL NUMBER EXCEEDS THE PRE-SPECIFIED NUMBER OF ATTRIBUTES.
10	NO	IN POINT	ERROR IN ESTABLISHING POINTERS. PLEASE NOTIFY.
11	NO	IN POINT	ERROR IN RSIBS PROCEDURE, PLEASE NOTIFY. POSSIBLE ERROR IN NODE HIERARCHY CODES.
12	NO	IN SUBTOL	ERROR IN POINTER USE. PLEASE NOTIFY.
13	YES	IN LONODE	ERROR IN LISTING NODES. PLEASE NOTIFY.
14	NO	IN POINT	ERROR IN RSIBS PROCEDURE, PLEASE NOTIFY.
14.5	NO	IN POINT	ERROR IN ESTABLISHING LEVEL POINTERS. PLEASE NOTIFY.
15	NO	IN PEEL	ERROR IN USE OF QUOTATION MARKS IN COMMANDS.
16	YES	IN LONODE	ERROR IN LISTING NODES. PLEASE NOTIFY.
17	YES	IN SUBTOL	INVALID NODE HIERARCHY CODE IN "LIST SUBTREE" COMMAND.
18	NO	IN PROCESS	ERROR IN COUNT SUBROUTINE. PLEASE NOTIFY.
19	NO	IN PROCESS	ERROR IN COUNT SUBROUTINE. PLEASE NOTIFY.
20	YES	IN LOPOINT	ERROR IN PRINTING POINTERS. PLEASE NOTIFY.
20.1	NO	IN PROCESS	ERROR IN SUMCOST SUBROUTINE. PLEASE NOTIFY.
20.2	NO	IN PROCESS	ERROR IN UPSTART SUBROUTINE. PLEASE NOTIFY.
20.3	NO	IN PROCESS	ERROR IN PRESENT SUBROUTINE. PLEASE NOTIFY.
21	NO	IN PROCESS	ERROR IN PONLY SUBROUTINE. PLEASE NOTIFY.
21.1	NO	IN PROCESS	ERROR IN UPMORE SUBROUTINE. PLEASE NOTIFY.
22.2	NO	IN ALTER	ERROR IN CHANGING MARGINAL FUNCTION DATA.
22.3	NO	IN ALTER	ERROR IN CHANGING MARGINAL FUNCTION DATA.
66.1	NO	IN CALCK	ERROR IN CALCULATING UTILITY CONSTANT; PLEASE NOTIFY.
66.2	NO	IN CALCK	ERROR IN CALCULATING UTILITY CONSTANT; PLEASE NOTIFY.
77.	NO	IN READRAW	INVALID UTILITY FUNCTION NAME.
77.1	NO	IN READRAW	ERROR IN UTILITY FUNCTION INPUT SEQUENCE.

Table 6.1 (Cont'd)

Error Number	REC ?	Description	Message
77.2	NO	IN READRAW	NUMBER OF UTILITY FUNCTIONS EXCEEDED THE PRESPECIFIED MAXIMUM.
77.3	NO	IN READRAW	ERROR IN CALCULATING UTILITY FUNCTION CONSTANT.
77.4	YES	IN READRAW	UTILITY CONSTANT OF 0., THE DEGENERATE ADDITIVE FORM OF THE UTILITY FUNCTION WILL BE USED; OTHER CONSTANTS WILL REMAIN MAIN UNALTERED.
80.1	NO	IN ALTER	INVALID SUBSTRING IN "ALTER" COMMAND.
88.	NO	IN UTILOF	ERROR IN CALCULATIONS OF MARGINAL FUNCTION.
88.5	NO	IN UTILOF	ERROR IN CALCULATIONS OF THE UTILITY FUNCTION.
88.6	NO	IN UTILOF	ERROR IN CALCULATIONS OF THE UTILITY FUNCTION; PLEASE NOTIFY.
99.1	NO	IN UTILOF	UPPER LIMIT OF GENERAL MARGINAL FUNCTION EXCEEDED.
99.2	NO	IN UTILOF	ERROR IN CALCULATIONS OF MARGINAL FUNC- TION; PLEASE NOTIFY.
99.3	NO	IN UTILOF	INVALID MARGINAL FUNCTION NAME.

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

REFERENCES

1. Keeney, R., and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Tradeoffs*, Wiley, New York (1976).
2. Walsh, W., *A Bayesian Approach to Forecasting Technical Success*, in Proceedings of U. of Chicago Conference on the Economics of Research and Development Policy (April 2, 1979).
3. Keeney, R., *Energy Policy and Value Tradeoffs*, IIASA Research Memorandum RM-75-76, Schloss Laxenburg, Austria (Dec. 1975).
4. Raiffa, H., *Decision Analysis - Introductory Lectures on Choices Under Uncertainty*, Addison-Wesley Publishing Co., Massachusetts (1970).
5. Buehring, W., W. Foell, and R. Keeney, *Examining Energy/Environment Policy Using Decision Analysis*, Energy Systems and Policy Vol. 2, no. 3, 1978.
6. Chinnis, J. Jr., *Evaluation Modules for Energy Technologies: Part II - Satellite Power Stations*, Decisions and Designs, Inc. Final Report PR 78-8-78 (1978).
7. Keeney, R., and A. Sicherman, *An Interactive Computer Program for Assessing and Analyzing Preferences Concerning Multiple Objectives*, IIASA Research Memorandum (Schloss Laxenburg, 2361 Laxenburg, Austria) (1975).
8. Keeney, A., *Evaluation of Proposed Storage Sites*, Operations Research, Vol. 27, No. 1 (Jan./Feb. 1979).
9. Sicherman, A., *An Interactive Computer Program for Assessing and Using Multiattribute Utility Functions*, Technical Report 111, Operations Research Center, MIT (1975).

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

ACKNOWLEDGMENTS

Ken Mullen was involved with the initial analysis and design stages of IDAP. Gene Brunner and Donald Keefer of Gulf Science and Technology Company provided several suggestions that were implemented on the IDAP. Other individuals, not necessarily associated with ANL, have assisted with the creation and refining of IDAP and with the preparation of this document. We would like to thank Beatrice Grabowski for the editing of this document.

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

APPENDIX 1 - RECENT CHANGES IN THE IDAP

Since this report was written, changes have been made in the IDAP. These changes are briefly described in this section.

CMS (Conversational Monitor System) is a software system directed by a control program (CP) on an IBM 3033 computer at ANL under VM (virtual machine) 370. CMS allows a user to interactively write, edit and execute computer programs by means of the user's virtual machine. A virtual machine is one for which the user can establish a customized configuration, including virtual card readers, CPUs, printers and other I/O devices. The remainder of this section assumes the user is able to log into CMS at ANL and is familiar with the basic commands available under CMS and CP.

In order to execute the IDAP on CMS, the user's virtual machine must conform to several configuration specifications. These specifications, with a list of the commands that can be entered to generate them, are listed below.

REQUIREMENT	COMMAND(S)
1000 KByte memory.	CP DEFINE STOR AS 1M IPL CMS (if this is not already the storage size of machine.)
link to B99832's CMS minidisk for access to IDAP files	CP LINK TO B99832 191 AS addr RR ACCESS addr type (where addr is a virtual address such as 193, and type is a file type such as B.)

Once the user has logged into CMS and has the proper machine configuration, the command

EXEC IDA

can be typed to begin execution of the CMS exec file that will establish file definitions, load the IDAP programs, and initiate the IDAP execution. A listing of the IDAP CMS exec file is presented below.

```

* CMS EXEC FILE TO RUN IDA
*LAST UPDATE: NOVEMBER 4, 1980
*
CP TERMINAL LINEDEL OFF
&CONTROL OFF
&IF .&3 = .ALL &CONTROL ALL
&IF .&3 = .CMS &CONTROL CMS
&PROG = &1
&IF .&PROG = .? &GOTO -ERR
&IF .&PROG = .NEWS T IDANEWS DATA *
&IF .&PROG = .NEWS &EXIT 0
&IF .&PROG = . &PROG = IDA
&EXTLIB = &2
*
* PROMPT FOR FILEDEFS
*
-PRO1 &TYPE ENTER FILE OF RAWDAT1 INPUT FILE:
&READ VARS &RAW1 &FT1 &FM1
&IF .&RAW1 = . &RAW1 = DUMMY
&IF .&FT1 = . &FT1 = DATA
&IF .&FM1 = . &FM1 = A
*
-PRO2 &TYPE ENTER FILE OF RAWDAT2 INPUT FILE:
&READ VARS &RAW2 &FT2 &FM2
&IF .&RAW2 = . &RAW2 = DUMMY
&IF .&FT2 = . &FT2 = DATA
&IF .&FM2 = . &FM2 = A
*
-PRO3 &TYPE ENTER FILE OF RAWDAT3 INPUT FILE:
&READ VARS &RAW3 &FT3 &FM3
&IF .&RAW3 = . &RAW3 = DUMMY
&IF .&FT3 = . &FT3 = DATA
&IF .&FM3 = . &FM3 = A
*
-PRO4 &TYPE ENTER FILE FOR GRAPHIC DATA POINT INPUT/ OUTPUT:
&READ VARS &GRAF &FT4 &FM4
&IF .&GRAF = . &GRAF = DUMMY
&IF .&FT4 = . &FT4 = DATA
&IF .&FM4 = . &FM4 = A
*
&IF &GRAF
&IF &RAW1
&IF &RAW2
&IF &RAW3
&GOTO -ERR
*
* LOAD
*
-OK GLOBAL TXTLIB CMSLIB PLILIB &EXTLIB IDALIB
&STACK HT
LOAD &PROG
&STACK RT
*
* FILEDEFS

```

```

*
FILEDEF DISPLAY T
FILEDEF USER T
&IF &RAW1 = DUMMY FILEDEF RAWDAT1 DUMMY
&IF &RAW1 = DUMMY &GOTO -TWO
FILEDEF RAWDAT1 DISK &RAW1 &FT1 &FM1 (RECFM FB LRECL 80 BLKSIZE 800
-TWO &IF &RAW2 = DUMMY FILEDEF RAWDAT2 DUMMY
    &IF &RAW2 = DUMMY &GOTO -THREE
FILEDEF RAWDAT2 DISK &RAW2 &FT2 &FM2 (RECFM FB LRECL 80 BLKSIZE 800
-THREE &IF &RAW3 = DUMMY FILEDEF RAWDAT3 DUMMY
    &IF &RAW3 = DUMMY &GOTO -FOUR
FILEDEF RAWDAT3 DISK &RAW3 &FT3 &FM3 (RECFM FB LRECL 80 BLKSIZE 800
-FOUR &IF &GRAF = DUMMY FILEDEF GFILE DUMMY
&IF &GRAF = DUMMY &GOTO -NEXT
FILEDEF GFILE DISK &GRAF &FT4 &FM4 (RECFM FB LRECL 80 BLKSIZE 800
-NEXT FILEDEF IDAERR DISK IDAERR DATA *(RECFM FB LRECL 80 BLKSIZE 800
FILEDEF IDAHEL DISK IDAHEL DATA *(RECFM FB LRECL 80 BLKSIZE 800
*
* GO
*
START
&EXIT 0
*
* ERROR PRINT OUT EXPLANATION ...
*
-ERR &BEGTYPE

```

TO EXECUTE THE "IDA" DECISION ANALYSIS PROGRAM,
THE USER MUST DEFINE SEVERAL DATA FILES:

FILE NAME	PURPOSE
RAWDAT1	THESE THREE FILES SHOULD CONTAIN THE RAW DATA THAT IS USED FOR INPUT TO IDA. AT LEAST ONE OF THESE FILES SHOULD NOT BE DUMMIED OUT WHEN THE USER GENERATES INPUT DATA OR BUILDS A TREE FROM WITHIN IDA.
RAWDAT2	
RAWDAT3	
GFILE	THIS FILE IS USED TO STORE DATA POINTS THAT CAN BE PLOTTED ON A LINE PRINTER BY IDA.

THE CORRECT FORMAT OF THE COMMAND TO EXECUTE "IDA" IS:

```
EXEC IDA newmain addlib cntl
```

WHERE "NEWMAIN" AND "ADDLIB" ARE OPTIONAL AND ARE TO
BE USED ONLY IN THE CASE OF PROGRAM DEVELOPMENT;
THEIR DEFAULT VALUES ARE SUFFICIENT TO EXECUTE THE CURRENT
VERSION OF THE "IDA" DECISION ANALYSIS PROGRAM.

THE VALUE OF "CNTL" CAN BE "CMS" OR "ALL"; IT IS
THE CONTROL PARAMETER FOR THE CMS EXEC FILE THAT INVOKES
IDA AND IS OPTIONAL. ITS DEFAULT VALUE IS "OFF".

TO OBTAIN A COPY OF THE IDA NEWSLETTER, ENTER:

EXEC IDA NEWS

&END

&EXIT 0

Several features have been added to the IDAP recently. These features are summarized below.

The user now has the ability to parameterize certain values within a decision tree when the RAWDAT input file is being generated. And when that tree is built by IDAP, the user can alter any of these global parameters by the use of a single command. A sensitivity analysis can also be performed on these global parameters. By parameterizing node tolls or probabilities, the user gains significant power when the decision tree consists of many similar subtrees. By issuing one command, parameters in the decision tree can be changed to a specific value.

The IDAIN procedure is now a PL/1 program within the IDAP on CMS. The user can construct a RAWDAT input data file and process the resulting tree in one terminal session.

Graphics capabilities are now available. IDAP is capable of generating a line printer plot from data that are either prompted from the user, stored within a data set, or generated during a sensitivity analysis procedure.

A sequence of HELP commands is also available to the user under CMS. As IDAP is developed, this sequence will be updated.

The following is an example IDAP terminal session. Note that when the user is being prompted for data, CMS types a "." character. Similarly, note the use of the HELP and GRAPH commands.

```
.EXEC IDA
CP TERMINAL LINEDEL OFF
ENTER FILE OF RAWDAT1 INPUT FILE:
.
ENTER FILE OF RAWDAT2 INPUT FILE:
.
ENTER FILE OF RAWDAT3 INPUT FILE:
.
ENTER FILE FOR GRAPHIC DATA POINT INPUT/ OUTPUT:
.JUNK
DMSLI0740I EXECUTION BEGINS...
```

WELCOME TO IDA!

AN INTERACTIVE PROGRAM
FOR DECISION ANALYSIS.

TODAY IS NOVEMBER 12, 1980. THE TIME IS 11:08:14 AM.

NO. OF ERROR MESSAGES AVAILABLE: 14

BEGIN ENTERING COMMANDS ...

```
COMMAND: .HELP
COMMAND ACCEPTED.
```

NO. OF HELP DOCUMENTS AVAILABLE: 18

IDA ACCEPTS COMMANDS THAT CONTAIN
SPECIFIC KEYWORDS. THE KEYWORDS THAT
ARE CURRENTLY DOCUMENTED IN THE "HELP"
COMMAND ARE INCLUDED IN THE FOLLOWING.

```
SESSION
*
SENSIT
COMMAND
ALTER
ALTER MARG
ALTER NODE
BUILD
END
QUIT
LIST INPUT
LIST MARG
LIST NODE
LIST SUBTREE
LIST UTIL
PROCESS
SET
GRAPH
```

TO OBTAIN A DETAILED DESCRIPTION OF
ANY OF THE ABOVE ASPECTS OF IDA, TYPE:

HELP 'KEYWORD'

WHERE "KEYWORD" IS ONE OF THOSE LISTED ABOVE.
NOTE: QUOTATION MARKS REQUIRED!

END OF DOCUMENTATION.

COMMAND: .HELP 'COMMAND'
COMMAND ACCEPTED.

HELP DOCUMENTATION ...

COMMAND

THE FOLLOWING LIST PROVIDES THE FORMATS FOR VARIOUS IDA COMMANDS.

NOTE: - THE "/" CHARACTER MUST BE ENTERED AS SHOWN.

- THE "#" CHARACTER MUST BE ENTERED AS SHOWN.
- PARAMETERS ENCLOSED IN "<>" MUST BE SUPPLIED AS APPROPRIATE.
- THE PARAMETER "" REPRESENTS A BIT STRING; ONLY 0'S AND 1'S CAN BE SUBSTITUTED.
- THE PARAMETER "<X>" REPRESENTS AN INTEGER VALUE; ONLY INTEGER VALUES MAY BE SUBSTITUTED.
- ALL OTHER STRINGS ARE KEYWORDS AND MUST BE ENTERED EXACTLY AS SHOWN IN THE COMMAND FORMAT.

* <COMMENT>
ALTER GLOB <NAME> <VALUE>
ALTER NODE <NODE-ID>
ALTER NODE <NODE-ID> PROB
ALTER NODE <NODE-ID> COST
ALTER NODE <NODE-ID> COST # <ATTNO>
ALTER MARG
ALTER MARG <ATTNO>
ALTER UTIL <MUF>
AVERAGE OUT
BUILD <RAWDAT<I>
END
GRAPH
GRAPH FILE
GRAPH SAVE
HELP
HELP <KEYWORD>

```

INPUT 'RAWDAT<X>'
LIST GLOB
LIST GLOB '<NAME>'
LIST INPUT
LIST SUBTREE '<NODEID>'
LIST NODE '<NODEID>'
LIST MARG
LIST MARG '<ATTNO>'
LIST PAR
LIST UTIL
LIST UTIL '<MUF>'
LIST ERR
LIST ERR '<ERRNO>'
LIST TIME
LIST DEBUG
LIST PAGE
LIST PROF
LIST SHOWC
LIST BATCH
LIST LISTE
LIST GRAPH

OUTPUT
OUTPUT '<COMMENT>'
OUTPUT PAGE
OUTPUT PAGE '<COMMENT>'
PROCESS '<MUF>'
QUIT
SENS GLOB '<NAME>' UTIL <X> FROM <F> TO <T> BY <B>
SENS MARG '<ATTNO>' UTIL <MUF> FROM <F> TO <T> BY <B>
SENS UTIL '<MUF>' ATT <ATTNO> FROM <F> TO <T> BY <B>
SENS NODE '<NODEID>' ATT <ATTNO> UTIL <MUF> FROM <F> TO <T> BY <B>
SET BATCH '<B>'
SET DEBUG '<BB>'
SET GRAPH '<B>'
SET LISTE '<B>'
SET PAGE '<XX>'
SET PROF '<BBB>'
SET SHOWC '<B>'
SET TERSE

END OF DOCUMENTATION.

COMMAND: .HELP 'GRAPH'
COMMAND ACCEPTED.

HELP DOCUMENTATION ...

```

 GRAPH

THERE ARE THREE COMMANDS TO EXECUTE GRAPHICS IN IDA.
 ALL THE GRAPHICS IS DONE ON A LINE PRINTER, AND MAY OR
 MAY NOT INVOLVE THE USE OF THE "GFILE" DATA FILE.
 THE THREE COMMANDS ARE:

COMMAND	FUNCTION
GRAPH	TO PLOT A SET OF POINTS INTERACTIVELY WITH NO READ OR WRITE TO "GFILE".
GRAPH FILE	TO PLOT POINTS THAT EXIST WITHIN "GFILE".
GRAPH SAVE	TO PLOT POINTS INTERACTIVELY, AND THEN SAVE THESE DATA POINTS IN "GFILE".

END OF DOCUMENTATION.

COMMAND: .GRAPH
 COMMAND ACCEPTED.
 INTERACTIVE, NO READ/WRITE TO GFILE.

ENTER XDIM, YDIM, FROM, AND TO: .3 4 0 0

ENTER 1 Y- AND 3 X-VALUES FOR Y(1)
 :
 .9 0 0 0

ENTER 1 Y- AND 3 X-VALUES FOR Y(2)
 :
 .6 2 3 4

ENTER 1 Y- AND 3 X-VALUES FOR Y(3)
 :
 .3 4 9 16

ENTER 1 Y- AND 3 X-VALUES FOR Y(4)
 :
 .0 8 18 64

X-AXIS WILL RANGE FROM 0 TO 0
 IF THESE VALUES ARE 0 OR IF
 THEY ARE CONFLICTING, THEY WILL
 BE SCALED AUTOMATICALLY.
 ENTER "FROM" AND "TO" VALUES IF YOU
 WANT TO OVERRIDE THEM:.

INITIATE LINE PLOT ...

SIZE OF EACH CHARACTER: 1.28000E+00

X-AXIS EXTREMES: 0.00000E+00 6.40000E+01

NUMBER OF Y POINTS: 4

NUMBER OF X POINTS: 3

***** NOTES *****

"\$" => HIDDEN POINT(S)

"<" => POINT(S) TOO FAR TO LEFT OF PLOT PAGE

">" => POINT(S) TOO FAR RIGHT OF PLOT PAGE.

COMMAND: .LIST PARM
COMMAND ACCEPTED.

TODAY IS NOVEMBER 12, 1980. THE TIME IS 11:17:13 AM.
VALUE OF "DEBUG" IS: 01
VALUE OF "PROFICI" IS: 000
VALUE OF "SHOWC" IS: 1
VALUE OF "BATCH" IS: 0
VALUE OF "PAGE" IS: 0
VALUE OF "LISTEXT" IS: 1
VALUE OF "GRAPH" IS: 1

COMMAND: .COMMAND WITH AN ERROR IN IT
COMMAND ACCEPTED.

**** ERROR NUMBER 0.500
ONCODE = 0
ORIGIN OF ERROR:
GETCOM PROCEDURE
MESSAGE:

INVALID COMMAND AT TOP LEVEL. TYPE "HELP" IF NECESSARY.

COMMAND: .HELP
COMMAND ACCEPTED.

IDA ACCEPTS COMMANDS THAT CONTAIN
SPECIFIC KEYWORDS. THE KEYWORDS THAT
ARE CURRENTLY DOCUMENTED IN THE "HELP"
COMMAND ARE INCLUDED IN THE FOLLOWING.

SESSION
*
SENSIT
COMMAND
ALTER
ALTER MARG
ALTER NODE
BUILD
END
QUIT
LIST INPUT
LIST MARG
LIST NODE
LIST SUBTREE
LIST UTIL
PROCESS
SET
GRAPH

TO OBTAIN A DETAILED DESCRIPTION OF
ANY OF THE ABOVE ASPECTS OF IDA, TYPE:

HELP 'KEYWORD'

WHERE "KEYWORD" IS ONE OF THOSE LISTED ABOVE.
NOTE: QUOTATION MARKS REQUIRED!

END OF DOCUMENTATION.

COMMAND: .END
COMMAND ACCEPTED.

TOTAL NUMBER OF COMMANDS PROCESSED: 11

TODAY IS NOVEMBER 12, 1980. THE TIME IS 11:19:52 AM.

THIS SESSION WITH IDA HAS BEEN SUCCESSFULLY COMPLETED.
R;