# DecisionMaker Software and Extracting Fuzzy Rules Under Uncertainty

## RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Kevin B. Walker of the University of Houston-Downtown. Dr. A. Glen Houston served as the RICIS research coordinator.

Funding was provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Robert T. Savely of the Information Technology Division, NASA/JSC.

# DecisionMaker Software
# and
# Extracting Fuzzy Rules Under Uncertainty

prepared by
Kevin B. Walker

in
Partial Fulfillment
of the Requirements

for
Computer Science 4395

Department of Applied Mathematical Sciences
University of Houston-Downtown
May 13, 1992

Committee Members/Approval

| | | |
|---|---|---|
| Dr. A. deKorvin | Faculty Advisor | Andre de Korvin |
| Dr. O. Sirisaengtaksin | Member | O. Sirisaengtaksin |
| Dr. M. Murphy | Member | Michael Murphy |
| Dr. R. Alo' | Chairman | 5/13/92 |

# ABSTRACT

In this paper, I examine knowledge acquisition under uncertainty. Theories proposed in Dr. A. deKorvin's paper *Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets* are discussed as they relate to rule calculation algorithms. A data structure for holding an arbitrary number of data fields is described. Limitations of Pascal for loops in the generation of combinations are also discussed. Finally, recursive algorithms for generating all possible combinations of attributes and for calculating the intersection of an arbitrary number of fuzzy sets are presented.

# ACKNOWLEDGEMENTS

First, I would like to thank my wife, Becky Walker for her incredible support over the last four and one-half years. I would also like to thank my good friend T.Kevin Johnston for his support and assistance (particularly in things mathematical). Several of my professors at the University of Houston - Downtown stand out. I would like to thank Dr. Andre' deKorvin, Dr. Mike Murphy, Dr. Ongard Sirasaegntaksin, Dr. Bob Lea, and Dr. Ken Oberhoff. I would also like to thank Dr. Richard Alo' for the Senior Seminar/Senior Project experience.

Last, I would like to thank my parents, Pat and Leo Walker, and my Grandmother, Gladys Hope Stevens Beals Eddy.

*To: Amanda and Grant*

# TABLE OF CONTENTS

# INTRODUCTION

My project was to write portions of a program to acquire knowledge under uncertainty. Program development was divided into two parts, user interface and rule calculation. Another student in Computer Science 4395 wrote the user interface. My portion was rule calculation for an arbitrary number of attributes. The resulting program is called DecisionMaker and uses theories of rough sets to learn from fuzzy examples. DecisionMaker looks at fuzzy diagnoses and builds rules from those diagnoses. Every possible combination of symptoms and decisions is tried. A fuzzy value is calculated for each combination telling us how much the decision depends on the symptoms.

Machine learning has been accomplished in several ways. One of the most popular ways is with a perceptron or neural network. These have been difficult to program and can require much computing power to run efficiently. Uncertainty is also a problem. Our program runs on an IBM or compatible personal computer and runs quickly when the number of attributes is kept reasonably small. Like the neural network, our program learns from examples. In our case, several examples are provided from a known expert. Each example includes one or more symptoms and one or more associated diagnoses or decisions.

The program looks for relationships between the symptoms and the diagnoses and produces rules and corresponding values of believability about each rule. One of the distinguishing features of this program is its ability to deal with uncertainty. The examples provided may have fuzzy attributes and the rules produced have fuzzy believability factors as well. An example of fuzzy attributes and rules might be: if the car engine cranks slowly (0.8 slow) and the lights were left on a long (.7 long) time then the battery is dead (.8 dead). This also shows that fuzzy examples are more typical than crisp ones.

A fuzzy set is similar to a crisp set. Like a crisp set, some things may be members of the set and some may not. The difference with fuzzy sets is that membership in a fuzzy set is not required to be all or nothing. For instance, let us say we have the set of blue things. Do we give membership to something that is green? It is not blue per se, but clearly green does contain some blue. This problem can be overcome with membership grade or degree of membership. In our example we could say that a green item is blue .5 or .6 depending on the amount of blue in the green. Each item in the set has a value associated with it indicating the degree to which the item is a member of the set. Values range from 0 to 1, with 1 indicating complete membership and 0 indicating no membership (items with membership grades of 0 are usually not written down.)

A rough set is a family of crisp sets all having the same lower and upper approximations. A more detailed explanation of the theory on which our program is based is contained in *Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets* by Andre' deKorvin.

Other programs exist that utilize this theory. University of Houston - Downtown students Donald Culas and Jeff Worm have written a program that demonstrates the theory nicely. There are limitations, however. The number of symptom attributes is limited to two, as is the number of decision attributes. Also, there is no provision for storing data between sessions so that data has to be reentered each time. In addition, the user interface is a bit difficult. Our program solves these problems. The program now accepts an arbitrary number of symptom and decision attributes. The user interface has been improved and provisions for storing and retrieving data in disk files have been made.

The primary reason for my use of rough and fuzzy sets is to deal with uncertainty. As stated above, knowledge acquisition can be programmed in several ways. Using rough sets allows us to use fuzzy examples to build our rules. A detailed explanation of the theories on which my portions of the program are based is contained in the next section. Following that is a description of how the program works. Finally I include my conclusions and a summary. The appendix contains a listing of the Pascal source code for my portions of the DecisionMaker program.

# THEORY

This chapter is a brief explanation of the mathematical theories used in the DecisionMaker software. A full explanation of the theories on which this program is based can be found in *Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets* by Dr. Andre' deKorvin of the University of Houston - Downtown.

DecisionMaker software uses the concepts of fuzzy sets and rough sets to produce its rules. Fuzzy sets are similar to crisp sets with the addition of membership grade or degree of membership. Each element of a fuzzy set has a value associated with it to indicate how much that element belongs to the set. A value of 1.0 would indicate total or complete membership while a membership grade of 0.0 would indicate no membership. Imagine the set of all blue things. Would grass belong to the set of blue things? Classical logic says no. Fuzzy logic says that since green consists of a mixture of blue and yellow, then grass belongs to the set of blue things (say .5 or .6 depending on the shade of green.) A fuzzy set, A, of "round things" is shown below.

A=.9/baseball + .3/bat + 1.0/pingpong ball + .5/bowl + .7/balloon

Clearly, some things are more round than others and that is indicated in the membership grade value. Also, those elements with membership grade of 0 are not usually listed.

Many mathematical operators have been defined for fuzzy sets. Intersection of two fuzzy sets A and B at every x has been defined as follows:

$$(A \cap B)(x) = \text{Min}\,\{A(x), B(x)\} \qquad (1)$$

Each corresponding element of the two sets is compared and the set of intersection is that set which results from taking the minimum of the two. Consider the two fuzzy sets, A and B. Let the intersection be AnB.

$$A = .2/X_1 + .5/X_2 + .9/X_3 + .7/X_4 + .6/X_5$$
$$B = .5/X_1 + .2/X_2 + .3/X_3 + .6/X_4 + .1/X_5$$
$$A \cap B = .2/X_1 + .2/X_2 + .3/X_3 + .6/X_4 + .1/X_5$$

The two other functions used in DecisionMaker are:

I(AcB) the degree to which A is contained in B

and

J(A#B) the degree to which A intersects B

I is calculated using the formula

$$I(AcB) = \min_x \max\{ 1\text{-}A(x), B(x)\}. \quad (2)$$

While J is calculated using the formula

$$J(A\#B) = \max_x \min \{ A(x), B(x) \}. \quad (3)$$

In the case of crisp sets, it is clear that

I(AcB) = 1 if and only if AcB. Otherwise it is 0.

Also, in the case of crisp sets it can be shown that

J(A#B) = 1 if and only if $A \cap B \neq \emptyset$. Otherwise it is 0.

What we are doing with the I(AcB) function is measuring the degree to which A implies B. This is correct because if A is contained in B, and if we have A, then we must have B as well. I values are associated with certain rules.

We call rules calculated with the J function "possible" rules. The following example shows how we use I(AcB) and J(A#B) to extract certain and possible rules from fuzzy diagnoses.

| | C | | D |
|---|---|---|---|
| $x_1$ | .3/R + .8/S | .2/H + .9/C | .3/Cl + .6/Op |
| $x_2$ | .1/R + .9/S | .2/H + .8/C | .2/Cl + .7/Op |
| $x_3$ | .8/R + .3/S | .7/H + .1/C | .8/Cl + .1/Op |
| $x_4$ | .7/R + .2/S | .6/H + .3/C | .7/Cl + .2/Op |

We can interpret this data in the following way: $x_1 \ldots x_4$ denote 4 automobiles in a repair shop. The symbols R and S stand for the automobile engine running roughly and smoothly; H and C stand for hot and cold engine temperature. Auto $x_1$ is running .3 rough and .8 smooth (this might be the combined opinions of several mechanics in the shop, but relative frequencies are not required). The column D represents fuzzy diagnoses. In our automotive model, $x_1$ could be said to have the thermostat stuck open, and we believe this diagnosis is .6. On the other hand, $x_3$ has the thermostat stuck closed and the corresponding belief is .7 strong.

DecisionMaker software takes this data and unravels it into fuzzy rules that suggest when the thermostat is likely to be open and when it is likely to be closed. The method that DecisionMaker software uses involves considering each extreme and each decision to be fuzzy sets. For example:

$$Cl = .3/x_1 + .2/x_2 + .8/x_3 + .7/x_4$$

We also have condition fuzzy sets such as:

$$R = .3/x_1 + .1/x_2 + .8/x_3 + .7/x_4$$
$$S = .8/x_1 + .9/x_2 + .3/x_3 + .2/x_4$$
$$H = .2/x_1 + .2/x_2 + .7/x_3 + .6/x_4$$
$$C = .9/x_1 + .8/x_2 + .1/x_3 + .3/x_4$$

DecisionMaker computes I(RcCl), I(ScCL), I(HcCl), I(CcCl), I(R∩HcCl), I(S∩HcCl), I(R∩CcCL), and I(S∩CcCl). The calculations produce the following values:

| | | | |
|---|---|---|---|
| I(RcCl) = .7 | I(ScCL) = .2 | I(RcOp) = .7 | I(ScOp) = .2 |
| I(HcCl) = .7 | I(CcCl) = .2 | I(HcOp) = .7 | I(CcOp) = .2 |
| I(R∩HcCl) = .7 | I(S∩HcCl) = .8 | I(R∩HcOp) = .7 | I(S∩HcOp) = .8 |
| I(R∩CcCL) = .7 | I(S∩CcCl) = .2 | I(R∩CcOp) = .7 | I(S∩CcOp) = .2 |

These values indicate the degree to which the decision is based on the symptoms. Similar calculations using J are produced. DecisionMaker computes J(RcCl), J(ScCL), J(HcCl), J(CcCl), J(R∩HcCl), J(S∩HcCl), J(R∩CcCL), and J(S∩CcCl). The calculations produce the following values:

| | | | |
|---|---|---|---|
| J(RcCl) = .8 | J(ScCL) = .3 | J(RcOp) = .3 | J(ScOp) = .7 |
| J(HcCl) = .7 | J(CcCl) = .3 | J(HcOp) = .2 | J(CcOp) = .7 |
| J(R∩HcCl) = .7 | J(S∩HcCl) = .3 | J(R∩HcOp) = .2 | J(S∩HcOp) = .2 |
| J(R∩CcCL) = .3 | J(S∩CcCl) = .3 | J(R∩CcOp) = .3 | J(S∩CcOp) = .7 |

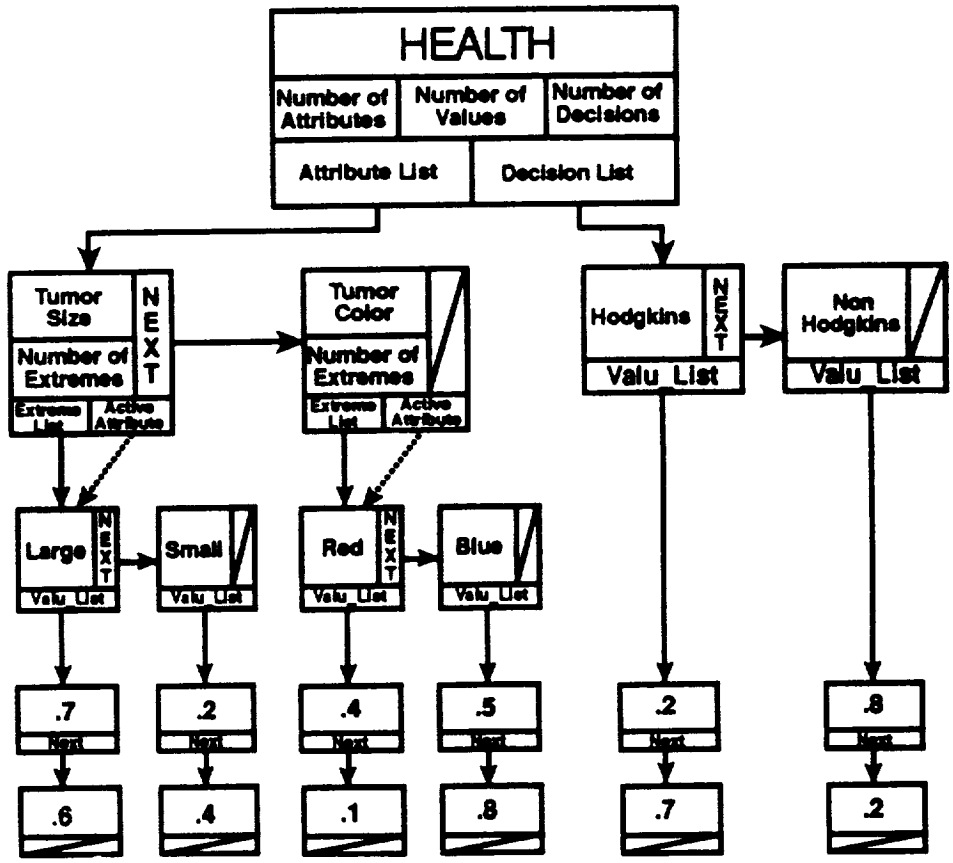
Now the rules can be extracted. Rules with I or J values below some arbitrary level can be ignored. In this way we can limit the number of rules and only look at the most significant ones.

We have discussed the concepts of fuzzy sets and some of their operators such as intersection, I(AcB) and J(A#B). These concepts are both simple and powerful and allow us to program a type of machine learning. In the next section I discuss how the DecisionMaker program was written and how it works.

# RESULTS

As previously stated, the goal of this project was to write a program to calculate rules from fuzzy diagnoses. A program written for the special case of two condition attributes, with two extremes each and two decision attributes, had already been written. I had access to this program but did not use any of the code as a basis for my program. My goal was a more general program, one that allowed for an arbitrary number of condition attributes, each with an arbitrary number of extremes, and allowing for an arbitrary number of decision attributes.

Designing a program to handle a data structure of undetermined size consists of two separate parts. The first part is the data structure. The second part is algorithms to manipulate that data structure. The data structure is a fairly simple linked list, illustrated below.



## Data Structure

The main data structure is a record containing several fields. Two fields are pointers, one to the condition branch and the other to the decision branch. There are three integer fields to contain information about the number of conditions, the number of decisions, and the number of data sets. Values for the integer fields are supplied by the user at run time. There is also a field of type "string" to contain a label supplied by the user.

The condition branch of the data structure consists of an arbitrary number of records, one for each of the condition attributes. The attributes records are arranged in a linked list so that their only limit is the amount of memory available in the computer. In addition to a label, each attribute record has an integer field containing the number of extremes for that attribute. There are pointers to the head of the linked list of extremes, the active extreme, and the next attribute.

Continuing on to the extreme record, we have a label, a pointer to the linked list of values associated with that extreme, and a pointer to the next extreme under that attribute. Each record in the value list consists of two fields, one for the real data and a pointer to the next record.

The decision branch is similar except for the lack of extreme records. Each decision record has a label, two pointers, one to the value list and another to the next decision record. The value records are identical to the ones used in the condition branch.

While simple in design, the DecisionMaker data structure allows the user to determine its size. We can see that the size of the data structure is limited only by available memory. The algorithms for entering the data, calculating the rules and printing them out, are much more complex.

## Algorithms

After the data structure is designed, the second part of writing a program is the algorithm. Algorithm design for this program can be divided into three parts. The first part is the user interface. The user interface for DecisionMaker was primarily written by Mr. Danny Picazo, a student at the University of Houston - Downtown. The user interface allows the user to enter data into the different fields of the data structure in any order, edit the entries, and select the active entry fields with "hot keys." The user interface is robust, handling all manner of user errors without causing an abend. This is by far the largest section of the program, being in excess of 2000 lines of Pascal source code.

The second part of the algorithm design is rule calculation. Rule calculation consists of generating all possible combinations of condition attributes and decision attributes. This can be most easily explained by example. Suppose that the user chooses two condition attributes and two decision attributes. Also suppose that each condition attribute has two possible extremes. The number of possible combinations is

(3 X 3 X 2) - 2 or 16 combinations.

A more general formula is:

$(E_1 \text{ X } E_2 \text{ X } \ldots \text{ X } E_n \text{ X } D) - D,$

where En is the number of extremes in the nth attribute + 1, and D is the number of decisions. D is subtracted because we do not consider the case where decisions are combined with no condition attributes. For example, if one condition attribute is size and the other color, the extremes could be large/small and red/blue. Decisions could be A and B. This would generate the following combinations:

| | |
|---|---|
| Red and A | Red and B |
| Blue and A | Blue and B |
| Large and A | Large and B |
| Small and A | Small and B |
| Large and Red and A | Large and Red and B |
| Small and Red and A | Small and Red and B |
| Large and Blue and A | Large and Blue and B |
| Small and Red and A | Small and Red and B |

We see at once that the number of combinations grows exponentially with respect to the number of attributes and extremes.

When the number of attributes etc. is known at compile time, the algorithm for calculating combinations is simple, as the following Pascal program illustrates.

```
program comb(input,output);

const
  on = true;
  off = false;

var
 i,j,k,l : integer;
 A,B,C,D : boolean;

begin
 A:= Off;
 for i := 1 to 2 do
 begin
    B:= Off;
    for j := 1 to 2 do
    begin
       C:= Off;
       for k := 1 to 2 do
       begin
          D:= Off;
          for l := 1 to 2 do
```

```
            begin
               if A then Write('A ');
               if B then Write('B ');
               if C then Write('C ');
               if D then Write('D ');
               Writeln;
               D:= on;
            end;{for l}
            C:= on;
         end;{for k}
         B:= on;
      end;{for j}
      A:= on;
   end;{for i}
end.
```

This program writes out each possible combination of ABCD. To add the letter E to this list would require rewriting the program and recompiling. Thus a for loop structure is not the answer to our problem. One way to solve this problem of combination involves recursion. Consider the following pseudo-code which visits every combination of extremes in the DecisionMaker data structure at least once:

```
procedure visit

  begin

  if (More Attributes) then
     visit (Next Attribute)

  if (More Extremes) then
     visit (Same Attribute, Next Extreme)

  if (Last Attribute) then
     process;

  end visit
```

Since Pascal is strongly typed and requires the number of parameters in a procedure call to match the number of formal parameters, more work is required for this pseudo-code to run. Below is the actual Pascal source code for the visit procedure.

```
procedure Visit(Attr :Attr_Ptr);

  var
     Last_Extr : Extr_Ptr;

  begin

    if (Attr^.Next_Attr <> NIL) then
       Visit(Attr^.Next_Attr);

    if (Attr^.Actv_Extr^.Next_Extr <> NIL) then
    begin
       Last_Extr := Attr^.Actv_Extr;
       Attr^.Actv_Extr := Attr^.Actv_Extr^.Next_Extr;
       Visit(Attr);
       Attr^.Actv_Extr := Last_Extr;
    end;

    if (Attr^.Next_Attr = NIL ) then
      process;

  end; { Visit }
```

Visit sets the Actv_Extr pointer to each of the extremes in turn so that when the last attribute is reached, process is called. As program flow pops back up the call stack Actv_Extr is varied and all possible combinations are "processed". Clearly, process can be defined as required. The version of process included below simply writes out each combination of condition attributes. The logic is much easier to follow here than in the actual program, with all it's special cases.

```
procedure process;

var
   One_or_More : boolean;
   i : integer;
   Attr : Attr_Ptr;

begin
   One_or_More := False;
   Attr := Main.Attr_List;
   for i := 1 to Main.Num_Attr do
   begin
      if Attr^.Curr_Extr^.Extreme <> 'none' then
        begin
           Write(Attr^.Curr_Extr^.Extreme,' ');
           One_or_More := True;
        end;
      Attr := Attr^.Next_Attr;
   end;{for i}
   if One_or_More then
      writeln;
end;{ Process }
```

In the DecisionMaker program, process is required to manage the calculation of an arbitrary number of fuzzy sets and to print the rules.

A recursive function called Recalc is called from within procedure process that returns a pointer to an attribute containing the intersection of any extremes pointed to by the Actv_Extr field of an attribute. The Pascal source code for function Recalc is shown below.

```
function Recalc (Attr:Attr_Ptr) : Attr_Ptr;

begin
   if Attr^.Next_Attr <> NIL then
     Recalc := Calc_Intersect(Attr, Attr^.Next_Attr)
   else
     Recalc := Attr;
end;
```

The final part of the algorithm design is rule output. Rule output is accomplished in two ways: one is by writing the rules to screen, the other by writing to a disk file. Rules are composed in English at the same time the values are calculated. Rules are output in the following format.

Red tumor color and large tumor size suggest disease A with belief level .6
Blue tumor color suggests disease B with belief level .7

DecisionMaker software allows the user to save the rules to disk. In addition, the user may extract rules by selecting a threshold value for I and or J, below which the rule will not be printed. Extraction becomes more important as the number of rules increases.

# CONCLUSIONS

Much has been done but there is much left to do. Time constraints imposed by the school term did not allow me to implement every feature I wanted in DecisionMaker.

Although DecisionMaker is easy to use due to the menus, there needs to be a better help facility. A better help facility would be accessible from any screen and be context sensitive. This would not be hard to code given enough time.

Alternate methods for computing I(AcB) have been discussed. It could be useful for the user to choose between several methods for calculating belief level of the rules. One such method is described below. Consider an expert who is "pretty good". He does statisticlly well, but is not the best there is. We might want to compute I(AcB) in the following way:

I(AcB) = $M(A \cap B)/M(A)$ or % of A in B

where $V = X_1/X_1 + X_2/X_2 + X_3/X_3 + .... + X_n/X_n$

and $M(V) = X_1 + X_2 + X_3 + ... + X_n$.

For example if we let A = {A,B,C} and B = {b,c,d,e}

it follows that A^B = {1/b + 1/c} = {b,c}

and I(AcB) = 2/3.

J(A#B) can be calculated using $J = 1\text{-}I(A \cap \neg B)$.

Another method of calulation involves using weights for each condition attribute. Each of these methods should be added to the program and the user could choose one of them from a menu.

The most important improvement to this program would be some improved way of calculating the rules. Because the number of rules grows exponentially with the number of condition attributes and extremes, the data structure quickly exceeds available memory. Alternate methods of calculating rules need to be found. One possible method would be to calculate rules with one extreme at a time and then eliminate those extremes which don't produce I values above a certain threshold. The visit procedure which produces the combinations would probably have to be changed substantially to do this, however it might be possible to mark each extreme as "eliminated." This would allow visit to ignore such extremes much the way process ignores extremes marked "none."

There are many possible improvements that can be made to the DecisionMaker program. Perhaps other students will try in the future.

# SUMMARY

In this work I have examined and demonstrated the theories expressed in Dr. Andre' deKorvin's paper *Extracting Fuzzy Rules Under Uncertainty and Measuring Definability Using Rough Sets*. The program described herein handles an arbitrary number of condition attributes, extremes of attributes, decisions and data sets. The data structure that allows the program to store an arbitrary amount of information is described. The limitations of Pascal for loops in creating combination of an arbitrary number of attributes was discussed. Recursive algorithms were presented that create all possible combinations of an arbitrary number of attributes. With these algorithms and data structures the DecisionMaker software is able to derive fuzzy rules from fuzzy diagnoses. Finally, source code for the DecisionMaker software is included.

```
Program DM(input,output);

  {************************************************************************}
  {************************************************************************}
  {******                                                          ******}
  {******  PROGRAM FOR PARTIAL FULFILLMENT OF THE REQUIREMENTS     ******}
  {******                    FOR CS 4395                           ******}
  {******                                                          ******}
  {******           Programmers :  Danny Picazo                    ******}
  {******                          Kevin B. Walker                 ******}
  {******                                                          ******}
  {******       This program simulate the ideas set forth in       ******}
  {******  Dr. Andre de Korvin's paper, "Extracting Fuzzy Rules    ******}
  {******  under uncertainty and Measuring Definibility using      ******}
  {******  Rough Sets". The program is designed to combine the     ******}
  {******  methods of rough sets and fuzzy sets to measure         ******}
  {******  uncertainty. Fuzzy rules are extracted to provide       ******}
  {******  the user a foundation from which they may formulate     ******}
  {******  a decision.                                             ******}
  {******       The main unit, DM, was written jointly. Units      ******}
  {******  DM_Calc and DM_View were written by Kevin Walker        ******}
  {******  Unit IO_PROC was written by Danny Picazo. In the DM     ******}
  {******  unit the main menu was written by Kevin Walker, the     ******}
  {******  data structure was designed jointly and the rest        ******}
  {******  was written by Danny Picazo.                            ******}
  {******                                                          ******}
  {************************************************************************}
  {************************************************************************}


Uses
  CRT, IO_PROC, DM_Calc, DM_View;

{Compiler Directives}
{====================}
{U$+}   { allow user to halt program execution by pressing CTRL-Break }

Const
  {Main Data Structure Constants}
  {==============================}
  MAX_STRING = 60;
  MAX_QUESTION_LENGTH = 60;
  MAX_ATTRIBUTE_LENGTH = 30;
  MAX_DECISION_LENGTH = 30;
  MAX_EXTREME_LENGTH = 15;

  DEF_DRIVE = 'C:';
  DEF_FILENAME = '(none)';
  DEF_EXT  = '.FZY';

  {Keyboard Return Code Constants}
  {===============================}
```

```
  NULL = #0;
  ESC_KEY = #27;     CR = #13;
  INS_KEY = #82;     DEL_KEY = #83;     BACKSPC = #8;
  TAB_KEY = #9;      SHFT_TAB = #15;
  UP_ARROW = #72;    DN_ARROW = #80;    LF_ARROW = #75;    RT_ARROW = #77
  HOME_KEY = #71;    END_KEY = #79;     PGUP_KEY = #73;    PGDN_KEY = #81
  F1_KEY = #59;      F2_KEY = #60;      F3_KEY = #61;      F4_KEY = #62;
  ALT_Q = #16;       ALT_D = #32;       ALT_A = #30;       ALT_S = #31;

  {Border ASCII Char Codes}
  {=======================}
  LT = 201;    RT = 187;    HORIZ = 205;
  LB = 200;    RB = 188;    VERT = 186;
  SHADOW_CHAR = 176;

Type
  MESSAGES = array[0..6] of string;

  SHADOW_CHOICE = ( Shadow, NoShadow );

  KEY_TYPE = ( AlphaNum, BackSpace, TabKey, ShftTab, HomeKey, EndKey
UpKey,
               DnKey, LfKey, RtKey, Escape, F1Key, F2Key, F3Key, F4Key
               EnterKey, ErrorKey, AltS, AltD, AltA, AltQ, PgUpKey
PgDnKey,
               InsKey, DelKey, EraseKey );

Var
   CHOICE: char;
    Row,     X,Y,I,J,     DCSN_ROW,TOP_DCSN,     ATTR_ROW,TOP_ATTR
EXTR_ROW,TOP_EXTR,
   Norm_Text,Norm_Background,High_Light : integer;
   QUIT, DATA_SAVED, QUIT_FLAG, NO_DATA_FLAG, EXIT_KEY_PRESSED: boolean
   WHICH_KEY: integer;
   MAIN_DATA: QUESTION_REC;
   PATH,FILENAME,EXT: string;
   ch : char;


{FORWARD CALL DECLARATIONS}
{=========================================================}
Procedure Help;                          Forward;
Procedure CLEAR_SYSTEM_PARAMETERS;       Forward;


Procedure INPUT_SYSTEM_DATA;
   const
      QSTN_XPOS = 9;    QSTN_YPOS = 4;
      ATTR_XPOS = 9;    ATTR_YPOS = 9;
      EXTR_XPOS = 60;   EXTR_YPOS = 9;
      DCSN_XPOS = 9;    DCSN_YPOS = 18;
   var
```

```
      A, LAST_ATTR_NUM: integer;
      DO_NOTHING,
      NO_MORE_ATTR_INPUT: boolean;
      INPUT_WNDW: array [1..4] of record
                     X,Y: integer;
                  end;
      CURR_EXTR,PREV_EXTR: EXTR_PTR;
      CURR_ATTR,PREV_ATTR: ATTR_PTR;
      CURR_DCSN,PREV_DCSN: DCSN_PTR;

   procedure WRITE_EXTR( var TOP_ROW,CURR_ROW: integer;   CURR_ATTR:
ATTR_PTR;
                         var CURR_EXTR,PREV_EXTR: EXTR_PTR);
      var
         NUM: string;
      begin

CLEAR_WINDOW(EXTR_XPOS,EXTR_YPOS,EXTR_XPOS+MAX_EXTREME_LENGTH,EXTR_YPO
S+2);
         textcolor(High_Light);
         gotoxy(EXTR_XPOS+MAX_EXTREME_LENGTH,EXTR_YPOS);
         if (TOP_ROW > 1 ) then
            write(chr(30));
         gotoxy(EXTR_XPOS+MAX_EXTREME_LENGTH,EXTR_YPOS+2);
         if ((TOP_ROW+2) < CURR_ATTR^.NUM_EXTR+1) and (CURR_ATTR <> nil)
then
            write(chr(31));
         textcolor(Norm_Text);
         if ( CURR_ATTR = nil ) then
            CURR_EXTR := nil
         else
            CURR_EXTR := CURR_ATTR^.EXTR_LIST;
         I := 1;
         while ( I < TOP_ROW ) do
            begin  CURR_EXTR := CURR_EXTR^.NEXT_EXTR;  I := I + 1;  end;
         I := 0;
         while ( I <= 2 ) and ( CURR_EXTR <> nil ) do
            begin
               str(TOP_ROW+I,NUM);
               gotoxy(EXTR_XPOS-4,EXTR_YPOS+I);
               write(NUM:2, ') ', CURR_EXTR^.EXTREME);
               CURR_EXTR := CURR_EXTR^.NEXT_EXTR;
               I := I + 1;
            end;
         if ( I <= 2 ) then
            begin
               str(TOP_ROW+I,NUM);
               gotoxy(EXTR_XPOS-4,EXTR_YPOS+I);
               write(NUM:2, ') ');
            end;
         PREV_EXTR := nil;
         if ( CURR_ATTR = nil ) then
```

```
            CURR_EXTR := nil
         else
            CURR_EXTR := CURR_ATTR^.EXTR_LIST;
         I := 1;
         while ( I < CURR_ROW ) do
            begin
               PREV_EXTR := CURR_EXTR;
               CURR_EXTR := CURR_EXTR^.NEXT_EXTR;
               I := I + 1;
            end;
      end;   {procedure WRITE_EXTR}

   procedure   WRITE_ATTR(   var   TOP_ROW,CURR_ROW:   integer;      va
CURR_ATTR,PREV_ATTR: ATTR_PTR );
      var
         NUM: string;
      begin

CLEAR_WINDOW(ATTR_XPOS,ATTR_YPOS,ATTR_XPOS+MAX_ATTRIBUTE_LENGTH,ATTR_
POS+4);
         textcolor(High_Light);
         gotoxy(ATTR_XPOS+MAX_ATTRIBUTE_LENGTH,ATTR_YPOS);
         if (TOP_ROW > 1 ) then
            write(chr(30));
         gotoxy(ATTR_XPOS+MAX_ATTRIBUTE_LENGTH,ATTR_YPOS+4);
         if ((TOP_ROW+4) < MAIN_DATA.NUM_ATTR+1) then
            write(chr(31));
         textcolor(Norm_Text);
         CURR_ATTR := MAIN_DATA.ATTR_LIST;
         I := 1;
         while ( I < TOP_ROW ) do
            begin  CURR_ATTR := CURR_ATTR^.NEXT_ATTR;  I := I + 1;  end
         I := 0;
         while ( I <= 4 ) and ( CURR_ATTR <> nil ) do
            begin
               str(TOP_ROW+I,NUM);
               gotoxy(ATTR_XPOS-4,ATTR_YPOS+I);
               write(NUM:2, ') ', CURR_ATTR^.ATTRIBUTE);
               CURR_ATTR := CURR_ATTR^.NEXT_ATTR;
               I := I + 1;
            end;
         if ( I <= 4 ) then
            begin
               str(TOP_ROW+I,NUM);
               gotoxy(ATTR_XPOS-4,ATTR_YPOS+I);
               write(NUM:2, ') ');
            end;
         PREV_ATTR := nil;
         CURR_ATTR := MAIN_DATA.ATTR_LIST;
         I := 1;
         while ( I < CURR_ROW ) do
            begin
```

```
            PREV_ATTR := CURR_ATTR;
            CURR_ATTR := CURR_ATTR^.NEXT_ATTR;
            I := I + 1;
         end;
   end;   {procedure WRITE_ATTR}

  procedure  WRITE_DCSN(  var  TOP_ROW,CURR_ROW:  integer;     var
CURR_DCSN,PREV_DCSN: DCSN_PTR );
   var
      NUM: string;
   begin

CLEAR_WINDOW(DCSN_XPOS,DCSN_YPOS,DCSN_XPOS+MAX_DECISION_LENGTH,DCSN_YP
OS+2);
      textcolor(High_Light);
      gotoxy(DCSN_XPOS+MAX_DECISION_LENGTH,DCSN_YPOS);
      if (TOP_ROW > 1 ) then
         write(chr(30));
      gotoxy(DCSN_XPOS+MAX_DECISION_LENGTH,DCSN_YPOS+2);
      if ((TOP_ROW+2) < MAIN_DATA.NUM_DCSN+1) then
         write(chr(31));
      textcolor(Norm_Text);
      CURR_DCSN := MAIN_DATA.DCSN_LIST;
      I := 1;
      while ( I < TOP_ROW ) do
         begin  CURR_DCSN := CURR_DCSN^.NEXT_DCSN;  I := I + 1;  end;
      I := 0;
      while ( I <= 2 ) and ( CURR_DCSN <> nil ) do
         begin
            str(TOP_ROW+I,NUM);
            gotoxy(DCSN_XPOS-4,DCSN_YPOS+I);
            write(NUM:2, ') ', CURR_DCSN^.DECISION);
            CURR_DCSN := CURR_DCSN^.NEXT_DCSN;
            I := I + 1;
         end;
      if ( I <= 2 ) then
         begin
            str(TOP_ROW+I,NUM);
            gotoxy(DCSN_XPOS-4,DCSN_YPOS+I);
            write(NUM:2, ') ');
         end;
      PREV_DCSN := nil;
      CURR_DCSN := MAIN_DATA.DCSN_LIST;
      I := 1;
      while ( I < CURR_ROW ) do
         begin
            PREV_DCSN := CURR_DCSN;
            CURR_DCSN := CURR_DCSN^.NEXT_DCSN;
            I := I + 1;
         end;
   end;   {procedure WRITE_DCSN}
```

```
procedure INPUT_SCREEN;
   begin
      Menu_Screen(0);
      textbackground(Black);  textcolor(High_Light);
      gotoxy(1,25);     write( 'F1' );
      gotoxy(10,25);    write( 'F2' );
      gotoxy(24,25);    write( 'F3' );
      gotoxy(40,25);    write( 'ESC' );
      gotoxy(50,25);    write( 'Tab,PgUp,PgDn' );
      textcolor(Norm_Text);
      gotoxy(3,25);     write('-Help');
      gotoxy(12,25);    write('-ClearData');
      gotoxy(26,25);    write('-InputValues');
      gotoxy(43,25);    write('-Menu');
      gotoxy(63,25);    write('-NextInputWindow');
      DRAW_BORDER(4,3,77,6,ord(Shadow));
      textcolor(High_Light);
      gotoxy(3,3);        write('Q');
      textcolor(Norm_Text);   write('UESTION/PROBLEM TO STUDY' );
      DRAW_BORDER(4,8,41,15,ord(Shadow));
      gotoxy(6,4);   write('=> ', MAIN_DATA.QUESTION);
      textcolor(High_Light);
      gotoxy(3,8);       write('S');
      textcolor(Norm_Text);   write('YMPTOMS OF PROBLEM');
      for I := 1 to 5 do
         begin  gotoxy(6,8+I);  write( I:1, ')');  end;
      WRITE_ATTR(TOP_ATTR,ATTR_ROW,CURR_ATTR,PREV_ATTR);
      DRAW_BORDER(4,17,41,22,ord(Shadow));
      textcolor(High_Light);
      gotoxy(3,17);        write('D');
      textcolor(Norm_Text);  write('ECISIONS/OUTCOMES TO PROBLEM');
      for I := 1 to 3 do
         begin  gotoxy(6,17+I);  write( I:1, ')');  end;
      WRITE_DCSN(TOP_DCSN,DCSN_ROW,CURR_DCSN,PREV_DCSN);
      I := round(MemAvail/1000);
      gotoxy(60,21);   write( 'Memory Avail: ', I:1, 'K' );
   end;   {procedure INPUT_SCREEN}

procedure QUESTION_INPUT;
   begin
         GET_INPUT_STRING( QSTN_XPOS, QSTN_YPOS, MAX_QUESTION_LENGTH,
                     MAIN_DATA.QUESTION, WHICH_KEY, EXIT_KEY_PRESSED
);
         case (WHICH_KEY) of
            ord(UpKey):
                     begin
                        A := 3;  DCSN_ROW := TOP_DCSN + 2;
                     end;
            ord(EnterKey),ord(DnKey):
                     begin
                        A := 2;  ATTR_ROW := TOP_ATTR;
                     end;
```

```
                    else  ;
               end;   (case WHICH_KEY)
         end;   (procedure QUESTION_INPUT)

      procedure EXTREME_INPUT( var CURR_ATTR: ATTR_PTR );
         var
                           X,Y: integer;
                     TEMP_EXTR: string;
                    INPUT_EXTR: EXTR_PTR;
                    EXTR_ADDED,
            NO_MORE_EXTR_INPUT: boolean;

         begin
            if (MAIN_DATA.NUM_ATTR > 0) then
               begin
                  textbackground(Norm_Text);
                  textcolor(Norm_BackGround);
                  gotoxy(ATTR_XPOS,ATTR_YPOS+(ATTR_ROW-TOP_ATTR));
                  write(CURR_ATTR^.ATTRIBUTE);
                  for    I    :=    1    to    (MAX_ATTRIBUTE_LENGTH    -
length(CURR_ATTR^.ATTRIBUTE)) do
                     write(' ');
                  textbackground(Norm_BackGround);
                  textcolor(Norm_Text);
               end;
            TOP_EXTR := 1;
            EXTR_ROW := 1;
            NO_MORE_EXTR_INPUT := False;
            X := EXTR_XPOS;
            Y := EXTR_YPOS;
            WRITE_EXTR(TOP_EXTR,EXTR_ROW,CURR_ATTR,CURR_EXTR,PREV_EXTR);
            EXIT_KEY_PRESSED := False;
            repeat
               if (CURR_EXTR = nil ) then
                  TEMP_EXTR := ''
               else
                  TEMP_EXTR := CURR_EXTR^.EXTREME;
               EXTR_ADDED := False;
               gotoxy(X,Y);

GET_INPUT_STRING(X,Y,MAX_EXTREME_LENGTH,TEMP_EXTR,WHICH_KEY,EXIT_KEY_P
RESSED);
               if ( length(TEMP_EXTR) > 0 ) then
                  begin
                     if ( CURR_EXTR = nil ) then
                        begin
                           new(INPUT_EXTR);
                           INPUT_EXTR^.EXTREME:= TEMP_EXTR;
                           INPUT_EXTR^.NEXT_EXTR := nil;
                           INPUT_EXTR^.VALU_LIST := nil;
                           TEMP_EXTR := '';
                           if ( CURR_ATTR^.NUM_EXTR = 0 ) then
```

```
                                begin
                                   CURR_ATTR^.NUM_EXTR := 1;
                                   CURR_ATTR^.EXTR_LIST := INPUT_EXTR;
                                end
                             else
                                begin
                                   PREV_EXTR := nil;
                                   CURR_EXTR := CURR_ATTR^.EXTR_LIST;
                                   while ( CURR_EXTR^.NEXT_EXTR <> nil ) d
                                      begin
                                         PREV_EXTR := CURR_EXTR;
                                         CURR_EXTR := CURR_EXTR^.NEXT_EXTR
                                      end;
                                   CURR_EXTR^.NEXT_EXTR := INPUT_EXTR;
                                  CURR_ATTR^.NUM_EXTR := CURR_ATTR^.NUM_EXTI
+ 1;
                                end;
                             PREV_EXTR := CURR_EXTR;
                             CURR_EXTR := INPUT_EXTR;
                             EXTR_ADDED := True;
                          end   {if CURR_EXTR = nil}
                       else   {CURR_EXTR <> nil}
                          CURR_EXTR^.EXTREME := TEMP_EXTR;
                    end;   {if length(TEMP_EXTR) > 0}
                 case (WHICH_KEY) of
                       ord(UpKey):
                             begin
                                Y:= Y - 1;
                                EXTR_ROW := EXTR_ROW - 1;
                                if ( Y < EXTR_YPOS ) then
                                   if ( EXTR_ROW <= 0 ) then
                                      begin
                                         TOP_EXTR := 1;
                                         EXTR_ROW := 1;
                                         NO_MORE_EXTR_INPUT := True;
                                         A := 2;   {return to ATTR box}
                                      end
                                   else       {you can scroll up in window
                                      begin
                                         Y := Y + 1;
                                         TOP_EXTR := EXTR_ROW;

WRITE_EXTR(TOP_EXTR,EXTR_ROW,CURR_ATTR,CURR_EXTR,PREV_EXTR);
                                      end
                                else       {normal up}
                                   begin
                                      CURR_EXTR := PREV_EXTR;
                                      TEMP_EXTR := CURR_EXTR^.EXTREME;
                                      PREV_EXTR := CURR_ATTR^.EXTR_LIST;
                                      I := 2;
                                      while ( I < EXTR_ROW ) do
                                         begin
```

```
                                                    PREV_EXTR    :=
PREV_EXTR^.NEXT_EXTR;
                                        I := I + 1;
                                    end;
                                end;
                            end;    {UpKey}
                        ord(EnterKey),ord(DnKey):
                            begin
                                Y := Y + 1;
                                if ( Y > (EXTR_YPOS+2) ) then
                                    if ( EXTR_ROW > CURR_ATTR^.NUM_EXTR ) then
                                        begin
                                            A := 2;
                                            EXTR_ROW := CURR_ATTR^.NUM_EXTR+1;
                                            TOP_EXTR := EXTR_ROW - 2;
                                            NO_MORE_EXTR_INPUT := True;
                                        end
                                    else            {you can scroll down in
window}
                                        begin
                                            Y := Y - 1;
                                            TOP_EXTR := TOP_EXTR + 1;
                                            EXTR_ROW := TOP_EXTR + 2;

WRITE_EXTR(TOP_EXTR,EXTR_ROW,CURR_ATTR,CURR_EXTR,PREV_EXTR);
                                        end
                                else        {normal down}
                                    if ( EXTR_ROW > CURR_ATTR^.NUM_EXTR ) then
                                        begin
                                            A := 2;
                                            EXTR_ROW := CURR_ATTR^.NUM_EXTR;
                                            TOP_EXTR := EXTR_ROW - 2;
                                            NO_MORE_EXTR_INPUT := True;
                                            DCSN_ROW := TOP_DCSN;
                                        end
                                    else
                                        begin
                                            PREV_EXTR := CURR_EXTR;
                                            CURR_EXTR := CURR_EXTR^.NEXT_EXTR;
                                            if (EXTR_ADDED) then
                                                TEMP_EXTR := ''
                                            else
                                                TEMP_EXTR := CURR_EXTR^.EXTREME;
                                            EXTR_ROW := EXTR_ROW + 1;
                                        end;
                            end;    {DnKey}
                        ord(EraseKey):
                            begin
                                sound(100);
                                delay(50);
                                nosound;
                            end;
```

```
                else  begin
                         NO_MORE_EXTR_INPUT := True;
                         NO_MORE_ATTR_INPUT := True;
                      end;
             end;    (case WHICH_KEY)
             EXIT_KEY_PRESSED := False;
          until ( NO_MORE_EXTR_INPUT );
          EXIT_KEY_PRESSED := True;
          TOP_EXTR := 1;
          EXTR_ROW := 1;
          gotoxy(EXTR_XPOS+MAX_EXTREME_LENGTH,EXTR_YPOS);  write(' ');
       end;    {procedure EXTREME_INPUT}

    procedure ATTRIBUTE_INPUT;
       var
                          X,Y: integer;
                    TEMP_ATTR: string;
                   INPUT_ATTR: ATTR_PTR;
                   ATTR_ADDED: boolean;

       begin
          DRAW_BORDER(55,8,77,13,ord(Shadow));
          textcolor(High_Light);
          gotoxy(54,8);   write('A');
          textcolor(Norm_Text);   write('TTRIBUTES OF SYMPTOMS');
          for I := 1 to 3 do
             begin  gotoxy(57,8+I);  write( I:1, ')' );  end;
          WRITE_EXTR(TOP_EXTR,EXTR_ROW,CURR_ATTR,CURR_EXTR,PREV_EXTR);
          NO_MORE_ATTR_INPUT := False;
          if (ATTR_ROW > (MAIN_DATA.NUM_ATTR+1)) then
             ATTR_ROW := MAIN_DATA.NUM_ATTR+1;
          if (ATTR_ROW < 1) then
              ATTR_ROW := 1;
          if (TOP_ATTR < 1) then
             TOP_ATTR := 1;
          X := ATTR_XPOS;
          if (ATTR_ROW = TOP_ATTR) then
             Y := ATTR_YPOS
          else
             Y := ATTR_YPOS + ( ATTR_ROW - TOP_ATTR );
          WRITE_ATTR(TOP_ATTR,ATTR_ROW,CURR_ATTR,PREV_ATTR);
          repeat
             if (CURR_ATTR = nil ) then
                TEMP_ATTR := ''
             else
                TEMP_ATTR := CURR_ATTR^.ATTRIBUTE;
             ATTR_ADDED := False;
             gotoxy(X,Y);

GET_INPUT_STRING(X,Y,MAX_ATTRIBUTE_LENGTH,TEMP_ATTR,WHICH_KEY,EXIT_KE
_PRESSED);
             if ( length(TEMP_ATTR) > 0 ) then
```

```
                begin
                   if ( CURR_ATTR = nil ) then
                      begin
                         new(INPUT_ATTR);
                         INPUT_ATTR^.ATTRIBUTE:= TEMP_ATTR;
                         INPUT_ATTR^.NEXT_ATTR := nil;
                         INPUT_ATTR^.NUM_EXTR := 0;
                         INPUT_ATTR^.EXTR_LIST := nil;
                         TEMP_ATTR := '';
                         if ( MAIN_DATA.NUM_ATTR = 0 ) then
                            begin
                               MAIN_DATA.NUM_ATTR := 1;
                               MAIN_DATA.ATTR_LIST := INPUT_ATTR;
                            end
                         else
                            begin
                               PREV_ATTR := nil;
                               CURR_ATTR := MAIN_DATA.ATTR_LIST;
                               while ( CURR_ATTR^.NEXT_ATTR <> nil ) do
                                  begin
                                     PREV_ATTR := CURR_ATTR;
                                     CURR_ATTR := CURR_ATTR^.NEXT_ATTR;
                                  end;
                               CURR_ATTR^.NEXT_ATTR := INPUT_ATTR;
                               MAIN_DATA.NUM_ATTR := MAIN_DATA.NUM_ATTR
+ 1;
                            end;
                         PREV_ATTR := CURR_ATTR;
                         CURR_ATTR := INPUT_ATTR;
                         ATTR_ADDED := True;
                      end   {if CURR_ATTR = nil}
                   else   {CURR_ATTR <> nil}
                      CURR_ATTR^.ATTRIBUTE := TEMP_ATTR;
                end;   {if length(TEMP_ATTR) > 0}
             case (WHICH_KEY) of
                   ord(UpKey):
                         begin
                            Y:= Y - 1;
                            ATTR_ROW := ATTR_ROW - 1;
                            if ( Y < ATTR_YPOS ) then
                               if ( ATTR_ROW <= 0 ) then
                                  begin
                                     TOP_ATTR := 1;
                                     ATTR_ROW := 1;
                                     NO_MORE_ATTR_INPUT := True;
                                     A := 1;   {set up call to QUESTION}
                                  end
                               else        {you can scroll up in window}
                                  begin
                                     Y := Y + 1;
                                     TOP_ATTR := ATTR_ROW;
```

```
WRITE_ATTR(TOP_ATTR,ATTR_ROW,CURR_ATTR,PREV_ATTR);
                                end
                            else        (normal up)
                               begin
                                  CURR_ATTR := PREV_ATTR;
                                  TEMP_ATTR := CURR_ATTR^.ATTRIBUTE;
                                  PREV_ATTR := MAIN_DATA.ATTR_LIST;
                                  I := 2;
                                  while ( I < ATTR_ROW ) do
                                     begin
                                                        PREV_ATTR    :=
PREV_ATTR^.NEXT_ATTR;
                                        I := I + 1;
                                     end;
                               end;
                         end;   {UpKey}
                   ord(DnKey):
                         begin
                            Y := Y + 1;
                            if ( Y > (ATTR_YPOS+4) ) then
                               if ( ATTR_ROW > MAIN_DATA.NUM_ATTR ) then
                                  begin
                                     A := 3;
                                     ATTR_ROW := MAIN_DATA.NUM_ATTR+1;
                                     TOP_ATTR := ATTR_ROW - 4;
                                     NO_MORE_ATTR_INPUT := True;
                                  end
                               else            {you can scroll down in
window)
                                  begin
                                     Y := Y - 1;
                                     TOP_ATTR := TOP_ATTR + 1;
                                     ATTR_ROW := TOP_ATTR + 4;

WRITE_ATTR(TOP_ATTR,ATTR_ROW,CURR_ATTR,PREV_ATTR);
                                  end
                            else        {normal down}
                               if ( ATTR_ROW > MAIN_DATA.NUM_ATTR ) then
                                  begin
                                     A := 3;
                                     ATTR_ROW := MAIN_DATA.NUM_ATTR;
                                     TOP_ATTR := ATTR_ROW - 4;
                                     NO_MORE_ATTR_INPUT := True;
                                     DCSN_ROW := TOP_DCSN;
                                  end
                               else
                                  begin
                                     PREV_ATTR := CURR_ATTR;
                                     CURR_ATTR := CURR_ATTR^.NEXT_ATTR
                                     if (ATTR_ADDED) then
                                        TEMP_ATTR := ''
                                     else
```

```
                                                    TEMP_ATTR    :=
CURR_ATTR^.ATTRIBUTE;
                                          ATTR_ROW := ATTR_ROW + 1;
                                       end;
                              end;    {DnKey}
                     ord(EnterKey),ord(AltA):
                              if (length(TEMP_ATTR) > 0) or (ATTR_ADDED) then
                                 EXTREME_INPUT(CURR_ATTR);
                     ord(EraseKey):
                              begin
                                 sound(100);
                                 delay(50);
                                 nosound;
                              end;
                           else  NO_MORE_ATTR_INPUT := True;
                     end;    {case WHICH_KEY}
                     WRITE_EXTR(TOP_EXTR,EXTR_ROW,CURR_ATTR,CURR_EXTR,PREV_EXTR);
                     EXIT_KEY_PRESSED := False;
                  until ( NO_MORE_ATTR_INPUT );
                  CLEAR_WINDOW(54,8,77,13);
                  EXIT_KEY_PRESSED := True;
                  gotoxy(ATTR_XPOS+MAX_ATTRIBUTE_LENGTH,ATTR_YPOS); write(' ');
                  gotoxy(ATTR_XPOS+MAX_ATTRIBUTE_LENGTH,ATTR_YPOS+4);   write('
');
            end;    {procedure ATTRIBUTE_INPUT}

         procedure DECISION_INPUT;
            var
                              X,Y: integer;
                        TEMP_DCSN: string;
                       INPUT_DCSN: DCSN_PTR;
                       DCSN_ADDED,
               NO_MORE_DCSN_INPUT: boolean;
            begin
               NO_MORE_DCSN_INPUT := False;
               if  (DCSN_ROW  >  (MAIN_DATA.NUM_DCSN+1))  then    DCSN_ROW  :=
MAIN_DATA.NUM_DCSN+1;
               if (DCSN_ROW < 1) then  DCSN_ROW := 1;
               if (TOP_DCSN < 1) then  TOP_DCSN := 1;
               X := DCSN_XPOS;
               if (DCSN_ROW = TOP_DCSN) then
                  Y := DCSN_YPOS
               else
                  Y := DCSN_YPOS + ( DCSN_ROW - TOP_DCSN );
               WRITE_DCSN(TOP_DCSN,DCSN_ROW,CURR_DCSN,PREV_DCSN);
               repeat
                  if (CURR_DCSN = nil ) then
                     TEMP_DCSN := ''
                  else
                     TEMP_DCSN := CURR_DCSN^.DECISION;
                  DCSN_ADDED := False;
                  gotoxy(X,Y);
```

```
GET_INPUT_STRING(X,Y,MAX_DECISION_LENGTH,TEMP_DCSN,WHICH_KEY,EXIT_KEY_
PRESSED);
            if ( length(TEMP_DCSN) > 0 ) then
               begin
                  if ( CURR_DCSN = nil ) then
                     begin
                        new(INPUT_DCSN);
                        INPUT_DCSN^.DECISION := TEMP_DCSN;
                        INPUT_DCSN^.NEXT_DCSN := nil;
                        INPUT_DCSN^.VALU_LIST := nil;
                        TEMP_DCSN := '';
                        if ( MAIN_DATA.NUM_DCSN = 0 ) then
                           begin
                              MAIN_DATA.NUM_DCSN := 1;
                              MAIN_DATA.DCSN_LIST := INPUT_DCSN;
                           end
                        else
                           begin
                              PREV_DCSN := nil;
                              CURR_DCSN := MAIN_DATA.DCSN_LIST;
                              while ( CURR_DCSN^.NEXT_DCSN <> nil ) dc
                                 begin
                                    PREV_DCSN := CURR_DCSN;
                                    CURR_DCSN := CURR_DCSN^.NEXT_DCSN;
                                 end;
                              CURR_DCSN^.NEXT_DCSN := INPUT_DCSN;
                              MAIN_DATA.NUM_DCSN := MAIN_DATA.NUM_DCSN
+ 1;
                           end;
                        PREV_DCSN := CURR_DCSN;
                        CURR_DCSN := INPUT_DCSN;
                        DCSN_ADDED := True;
                     end   {if CURR_DSCN = nil}
                  else   {CURR_DCSN <> nil}
                     CURR_DCSN^.DECISION := TEMP_DCSN;
               end;   {if length(TEMP_DCSN) > 0}
            case (WHICH_KEY) of
                  ord(UpKey):
                        begin
                           Y:= Y - 1;
                           DCSN_ROW := DCSN_ROW - 1;
                           if ( Y < DCSN_YPOS ) then
                              if ( DCSN_ROW <= 0 ) then
                                 begin
                                    TOP_DCSN := 1;
                                    DCSN_ROW := 1;
                                    NO_MORE_DCSN_INPUT := True;
                                    A := 2;   {set up call to ATTRIBUTE}
                                    ATTR_ROW := TOP_ATTR + 4;
                                 end
                              else       {you can scroll up in window}
```

```
                                    begin
                                       Y := Y + 1;
                                       TOP_DCSN := DCSN_ROW;

WRITE_DCSN(TOP_DCSN,DCSN_ROW,CURR_DCSN,PREV_DCSN);
                                    end
                                 else        {normal up}
                                    begin
                                       CURR_DCSN := PREV_DCSN;
                                       TEMP_DCSN := CURR_DCSN^.DECISION;
                                       PREV_DCSN := MAIN_DATA.DCSN_LIST;
                                       I := 2;
                                       while ( I < DCSN_ROW ) do
                                          begin
                                                                PREV_DCSN    :=
PREV_DCSN^.NEXT_DCSN;
                                             I := I + 1;
                                          end;
                                    end;
                              end;   {UpKey}
                        ord(EnterKey),ord(DnKey):
                              begin
                                 Y := Y + 1;
                                 if ( Y > (DCSN_YPOS+2) ) then
                                    if ( DCSN_ROW > MAIN_DATA.NUM_DCSN ) then
                                       begin
                                          A := 1;
                                          DCSN_ROW := MAIN_DATA.NUM_DCSN+1;
                                          TOP_DCSN := DCSN_ROW - 2;
                                          NO_MORE_DCSN_INPUT := True;
                                       end
                                    else           {you can scroll down in
window}
                                       begin
                                          Y := Y - 1;
                                          TOP_DCSN := TOP_DCSN + 1;
                                          DCSN_ROW := TOP_DCSN + 2;

WRITE_DCSN(TOP_DCSN,DCSN_ROW,CURR_DCSN,PREV_DCSN);
                                       end
                                 else        {normal down}
                                    if ( DCSN_ROW > MAIN_DATA.NUM_DCSN ) then
                                       begin
                                          A := 1;
                                          DCSN_ROW := MAIN_DATA.NUM_DCSN;
                                          TOP_DCSN := DCSN_ROW - 2;
                                          NO_MORE_DCSN_INPUT := True;
                                       end
                                    else
                                       begin
                                          PREV_DCSN := CURR_DCSN;
                                          CURR_DCSN := CURR_DCSN^.NEXT_DCSN;
```

```
                                        if (DCSN_ADDED) then
                                           TEMP_DCSN := ''
                                        else
                                          TEMP_DCSN := CURR_DCSN^.DECISION;
                                        DCSN_ROW := DCSN_ROW + 1;
                                     end;
                              end;    {Enter,DnKey}
                        ord(EraseKey):
                              begin
                                 sound(100);
                                 delay(50);
                                 nosound;
                              end;
                           else  NO_MORE_DCSN_INPUT := True;
                     end;    {case WHICH_KEY}
                     EXIT_KEY_PRESSED := False;
                  until ( NO_MORE_DCSN_INPUT );
                  EXIT_KEY_PRESSED := True;
                  gotoxy(DCSN_XPOS+MAX_DECISION_LENGTH,DCSN_YPOS);  write(' ');
                  gotoxy(DCSN_XPOS+MAX_DECISION_LENGTH,DCSN_YPOS+2); write(' ');
               end;    {procedure DECISION_INPUT}

            {*****************************************}
            {*  main procedure INPUT_SYSYEM_DATA  *}
            {*****************************************}
            begin
               INPUT_WNDW[1].X := QSTN_XPOS;    INPUT_WNDW[1].Y := QSTN_YPOS;
               INPUT_WNDW[2].X := ATTR_XPOS;    INPUT_WNDW[2].Y := ATTR_YPOS;
               INPUT_WNDW[3].X := DCSN_XPOS;    INPUT_WNDW[3].Y := DCSN_YPOS;
               INPUT_WNDW[4].X := EXTR_XPOS;    INPUT_WNDW[4].Y := EXTR_YPOS;
               PREV_DCSN := nil;                CURR_DCSN := MAIN_DATA.DCSN_LIST;
               PREV_ATTR := nil;                CURR_ATTR := MAIN_DATA.ATTR_LIST;
               PREV_EXTR := nil;                 CURR_EXTR := nil;
               if  (DCSN_ROW  >  (MAIN_DATA.NUM_DCSN+1))  then     DCSN_ROW  :=
            MAIN_DATA.NUM_DCSN+1;
               if (DCSN_ROW < 1) then  DCSN_ROW := 1;
               if (TOP_DCSN < 1) then  TOP_DCSN := 1;
               if  (ATTR_ROW  >  (MAIN_DATA.NUM_ATTR+1))  then     ATTR_ROW  :=
            MAIN_DATA.NUM_ATTR+1;
               if (ATTR_ROW < 1) then  ATTR_ROW := 1;
               if (TOP_ATTR < 1) then  TOP_ATTR := 1;
               A := 1;
               CHOICE := ' ';
               EXIT_KEY_PRESSED := False;
               DO_NOTHING := False;
               INPUT_SCREEN;
               X := QSTN_XPOS;    Y := QSTN_YPOS;
               QUESTION_INPUT;
               repeat
                  CHOICE := ' ';
                  gotoxy(X,Y);
                  if ( NOT EXIT_KEY_PRESSED ) then
```

```
            WHICH_KEY := WHICH_KEY_PRESSED(CHOICE)
         else
            EXIT_KEY_PRESSED := False;
         case ( WHICH_KEY ) of
              ord(F1Key): begin  HELP;  INPUT_SCREEN;  end;
              ord(F2Key): begin CLEAR_SYSTEM_PARAMETERS; INPUT_SCREEN;
end;
              ord(F3Key): begin   {goto INPUT_VALUES}   {INPUT_SCREEN}
end;
               ord(AltQ): A := 1;
               ord(AltS): A := 2;
               ord(AltD): A := 3;
             ord(TabKey),ord(PgDnKey):
                          begin
                             A := A + 1;
                             if ( A > 3 ) then
                                A := 1;
                             X    :=    INPUT_WNDW[A].X;        Y    :=
INPUT_WNDW[A].Y;
                          end;
            ord(ShftTab),ord(PgUpKey):
                          begin
                             A := A - 1;
                             if ( A < 1 ) then
                                A := 3;
                             X    :=    INPUT_WNDW[A].X;        Y    :=
INPUT_WNDW[A].Y;
                          end;
            ord(EnterKey),ord(UpKey),ord(DnKey): ;
                     else  DO_NOTHING := True;
         end;   {case WHICH_KEY}
         if ( NOT DO_NOTHING ) then
            case (A) of
               1: QUESTION_INPUT;
               2: ATTRIBUTE_INPUT;
               3: DECISION_INPUT;
            end   {case A}
         else
            DO_NOTHING := False;
      until ( WHICH_KEY = ord(Escape) );
      with MAIN_DATA do
        if (length(QUESTION) = 0) and (NUM_ATTR = 0) and (NUM_DCSN = 0)
then
           NO_DATA_FLAG := True
        else
           NO_DATA_FLAG := False;
   end;   {procedure INPUT_SYSTEM_DATA}


Procedure RETRIEVE_DATA;
   const
```

```
      INIT_XPOS = 22;   INIT_YPOS = 7;
   var
              X,Y: integer;
             DONE,
       VALID_FILE,
      FILE_EXISTS,
       DO_NOTHING: boolean;
           INFILE: text;

   procedure GET_FILENAME;
      begin
         if (WHICH_KEY <> ord(Escape)) then
            begin
               repeat
                  VALID_FILE := True;
                  EXIT_KEY_PRESSED := False;

GET_INPUT_STRING(X,Y+2,8,FILENAME,WHICH_KEY,EXIT_KEY_PRESSED);
                  if (WHICH_KEY = ord(EnterKey)) and (length(FILENAME
> 0) then
                     begin
                        I := 2;
                        if (FILENAME[1] in ['A'..'Z','a'..'z']) then
                           while  (  I  <=  length(FILENAME)  )  an
(VALID_FILE) do
                                          if   (FILENAME[I]    i
['0'..'9','A'..'Z','a'..'z',
                                                '_','0'..'9']) then
                                    I := I + 1
                                 else
                                    VALID_FILE := False
                        else
                           VALID_FILE := False;
                        if (VALID_FILE) then
                           begin
                              assign(INFILE,PATH+FILENAME+EXT);
                              (*$I-*)
                              reset(INFILE);
                              (*$I+*)
                              FILE_EXISTS := IOresult = 0;
                              if (NOT FILE_EXISTS) then
                                 begin
                                    VALID_FILE := False;
                                    gotoxy(10,Y+10);
                                    write('File does not exist.  Pleas
try again!');

                                    delay(2000);
                                    CLEAR_WINDOW(5,17,75,17);
                                 end;
                           end
                        else
                           begin
```

```
                              gotoxy(10,Y+10);
                              write('Illegal Filename. Try Again!');
                              delay(2000);
                              gotoxy(X,Y+10);
                              CLEAR_WINDOW(5,17,75,17);
                           end;
                     end
                  else
                     VALID_FILE := False;
               until (WHICH_KEY = ord(Escape)) or (VALID_FILE);
            end;
         textcolor(High_Light);
         gotoxy(X,Y+2);   write(FILENAME,EXT,'          ');
end;   {procedure GET_FILENAME}

procedure READ_DATA_FILE;
   var
        I,J,K,L: integer;
       NEW_ATTR,
      CURR_ATTR: ATTR_PTR;
       NEW_EXTR,
      CURR_EXTR: EXTR_PTR;
       NEW_DCSN,
      CURR_DCSN: DCSN_PTR;
       NEW_VALU,
      CURR_VALU: VALU_PTR;
   begin
      gotoxy(10,17);
      write('Now Reading: ',PATH,FILENAME,EXT);
      assign(INFILE,PATH+FILENAME+EXT);
      reset(INFILE);
      with MAIN_DATA do
         begin
            read( INFILE, I, CHOICE );
            QUESTION[0] := chr(I);
            readln( INFILE, QUESTION );
            readln( INFILE, NUM_DCSN, NUM_ATTR, NUM_VALU );
         end;
      if ( MAIN_DATA.NUM_DCSN > 0 ) then
         begin
            new(NEW_DCSN);
            MAIN_DATA.DCSN_LIST := NEW_DCSN;
         end;
      J := 0;
      while (J < MAIN_DATA.NUM_DCSN) and (NOT EOF(INFILE)) do
         begin
            readln(INFILE);
            CURR_DCSN := NEW_DCSN;
            CURR_DCSN^.DECISION := '';
            CURR_DCSN^.NEXT_DCSN := nil;
            CURR_DCSN^.VALU_LIST := nil;
            read( INFILE, I, CHOICE );
```

```
      CURR_DCSN^.DECISION[0] := chr(I);
      readln( INFILE, CURR_DCSN^.DECISION );
      if ( MAIN_DATA.NUM_VALU > 0 ) then
         begin
            new(NEW_VALU);
            CURR_DCSN^.VALU_LIST := NEW_VALU;
         end;
      K := 0;
      while ( K < MAIN_DATA.NUM_VALU ) and (NOT EOF(INFILE)) do
         begin
            CURR_VALU := NEW_VALU;
            CURR_VALU^.VALUE := 0.0;
            CURR_VALU^.NEXT_VALU := nil;
            with CURR_VALU^ do
               if (K <= 4) then
                  read( INFILE, VALUE )
               else
                  begin
                     K := 0;
                     readln( INFILE, VALUE );
                  end;
            new(NEW_VALU);
            CURR_VALU^.NEXT_VALU := NEW_VALU;
            K := K + 1;
         end;   {while K < NUM_VALU}
      CURR_VALU^.NEXT_VALU := nil;
      if ( MAIN_DATA.NUM_VALU > 0 ) then
         begin
            dispose(NEW_VALU);
            if (K > 0) then
               readln(INFILE);
         end;
      new(NEW_DCSN);
      CURR_DCSN^.NEXT_DCSN := NEW_DCSN;
      J := J + 1;
   end;   {while J < NUM_DCSN}
CURR_DCSN^.NEXT_DCSN := nil;
if ( MAIN_DATA.NUM_DCSN > 0 ) then
   dispose(NEW_DCSN);
if ( MAIN_DATA.NUM_ATTR > 0 ) then
   begin
      new(NEW_ATTR);
      MAIN_DATA.ATTR_LIST := NEW_ATTR;
   end;
J := 0;
while (J < MAIN_DATA.NUM_ATTR ) and (NOT EOF(INFILE)) do
   begin
      readln(INFILE);
      CURR_ATTR := NEW_ATTR;
      CURR_ATTR^.ATTRIBUTE := '';
      CURR_ATTR^.NEXT_ATTR := nil;
      CURR_ATTR^.NUM_EXTR := 0;
```

```
                CURR_ATTR^.EXTR_LIST := nil;
                read( INFILE, I, CHOICE );
                CURR_ATTR^.ATTRIBUTE[0] := chr(I);
                readln( INFILE, CURR_ATTR^.ATTRIBUTE, CURR_ATTR^.NUM_EXTR
);
                if ( CURR_ATTR^.NUM_EXTR > 0 ) then
                   begin
                      new(NEW_EXTR);
                      CURR_ATTR^.EXTR_LIST := NEW_EXTR;
                   end;
                K := 0;
                while ( K < CURR_ATTR^.NUM_EXTR ) do
                   begin
                      CURR_EXTR := NEW_EXTR;
                      CURR_EXTR^.EXTREME := '';
                      CURR_EXTR^.NEXT_EXTR := nil;
                      CURR_EXTR^.VALU_LIST := nil;
                      read( INFILE, I, CHOICE );
                      CURR_EXTR^.EXTREME[0] := chr(I);
                      readln( INFILE, CURR_EXTR^.EXTREME );
                      if ( MAIN_DATA.NUM_VALU > 0 ) then
                         begin
                            new(NEW_VALU);
                            CURR_EXTR^.VALU_LIST := NEW_VALU;
                         end;
                      L := 0;
                      while  (  L  <  MAIN_DATA.NUM_VALU  )  and  (NOT
EOF(INFILE)) do
                         begin
                            CURR_VALU := NEW_VALU;
                            CURR_VALU^.VALUE := 0.0;
                            CURR_VALU^.NEXT_VALU := nil;
                            with CURR_VALU^ do
                            if (L <= 4) then
                               read( INFILE, VALUE )
                            else
                               begin
                                  L := 0;
                                  readln( INFILE, VALUE );
                               end;
                            new(NEW_VALU);
                            CURR_VALU^.NEXT_VALU := NEW_VALU;
                            L := L + 1;
                         end;   {while L < NUM_VALU}
                      CURR_VALU^.NEXT_VALU := nil;
                      if ( MAIN_DATA.NUM_VALU > 0 ) then
                         begin
                            dispose(NEW_VALU);
                            if (L > 0) then
                               readln(INFILE);
                         end;
                      new(NEW_EXTR);
```

```
                        CURR_EXTR^.NEXT_EXTR := NEW_EXTR;
                        K := K + 1;
                     end;    {while K < NUM_EXTR}
                  CURR_EXTR^.NEXT_EXTR := nil;
                  if ( CURR_ATTR^.NUM_EXTR > 0 ) then
                     dispose(NEW_EXTR);
                  J := J + 1;
                  new(NEW_ATTR);
                  CURR_ATTR^.NEXT_ATTR := NEW_ATTR;
               end;    {while J < NUM_ATTR}
            CURR_ATTR^.NEXT_ATTR := nil;
            if ( CURR_ATTR^.NUM_EXTR > 0 ) then
               dispose(NEW_ATTR);
            close(INFILE);
            TOP_ATTR := 1;    ATTR_ROW := 1;
            TOP_EXTR := 1;    EXTR_ROW := 1;
            TOP_DCSN := 1;    DCSN_ROW := 1;
            with MAIN_DATA do
               if (length(QUESTION) = 0) and (NUM_ATTR = 0) and (NUM_DCS
= 0) then
                  NO_DATA_FLAG := True
               else
                  NO_DATA_FLAG := False;
         end;    {procedure READ_DATA_FILE}

      begin
         Menu_Screen(1);
         textcolor(High_Light);
         gotoxy(1,25);     write('ESC');
         gotoxy(16,25);    write(chr(17),chr(196),chr(217));
         textcolor(Norm_Text);
         gotoxy(4,25);     write('-Main Menu');
         gotoxy(19,25);    write('-ReadFile!');
         DRAW_BORDER(3,5,78,19,ord(Shadow));
         X := INIT_XPOS;    Y := INIT_YPOS;
         gotoxy(X-11,Y);      write('DIRECTORY:');
         gotoxy(X-10,Y+2);    write('FILENAME:');
         textcolor(High_Light);
         gotoxy(X,Y);
         I := length(PATH);
         if (PATH[I] <> '\') then
            begin
               I := I + 1;
               PATH[0] := chr(I);
               PATH[I] := '\';
            end;
         write(PATH);
         gotoxy(X,Y+2);    write(FILENAME);
         gotoxy(X+8,Y+2);  write(EXT);
         gotoxy(X-7,Y+5);    write(MAIN_DATA.QUESTION);
         WHICH_KEY := ord(ErrorKey);
         repeat
```

```
      GET_FILENAME;
    until (VALID_FILE) or (WHICH_KEY = ord(Escape));
    if (WHICH_KEY <> ord(Escape)) and (VALID_FILE) then
      READ_DATA_FILE;
    textcolor(Norm_Text);
  end;   (procedure RETRIEVE_DATA)


Procedure SAVE_DATA;
  const
    INIT_XPOS = 22;   INIT_YPOS = 7;
  var
          X,Y: integer;
         DONE,
    VALID_FILE,
    FILE_EXISTS,
     DO_NOTHING: boolean;
      OUTFILE: text;

  procedure GET_FILENAME;
    begin
      if (WHICH_KEY <> ord(Escape)) then
        begin
          repeat
            VALID_FILE := True;
            EXIT_KEY_PRESSED := False;

GET_INPUT_STRING(X,Y+2,8,FILENAME,WHICH_KEY,EXIT_KEY_PRESSED);
            if (WHICH_KEY = ord(EnterKey)) and (length(FILENAME)
> 0) then
              begin
                I := 2;
                if (FILENAME[1] in ['A'..'Z','a'..'z']) then
                  while  (  I  <=  length(FILENAME)  )  and
(VALID_FILE) do
                                    if   (FILENAME[I]   in
['0'..'9','A'..'Z','a'..'z',
                                       '_','0'..'9']) then
                      I := I + 1
                    else
                      VALID_FILE := False
                else
                  VALID_FILE := False;
                if (VALID_FILE) then
                  begin
                    assign(OUTFILE,PATH+FILENAME+EXT);
                    (*$I-*)
                    reset(OUTFILE);
                    (*$I+*)
                    FILE_EXISTS := IOresult = 0;
                    if (FILE_EXISTS) then
                      begin
```

```
                                        gotoxy(10,Y+10);
                                         write('File   Already   Exists
Overwrite (Y/N)?');
                                        repeat
                                           CHOICE := readkey;
                                           CHOICE := upcase(CHOICE);
                                        until ( CHOICE in ['Y','N'] );
                                        if ( CHOICE = 'N' ) then
                                           begin
                                              close(OUTFILE);
                                              VALID_FILE := False;
                                           end;
                                        gotoxy(10,Y+10);
                                        CLEAR_WINDOW(5,17,75,17);
                                     end;
                                end
                             else
                                begin
                                   gotoxy(10,Y+10);
                                   write('Illegal Filename. Try Again!');
                                   delay(2000);
                                   gotoxy(X,Y+10);
                                   CLEAR_WINDOW(5,17,75,17);
                                end;
                          end
                       else
                          VALID_FILE := False;
                    until (WHICH_KEY = ord(Escape)) or (VALID_FILE);
                 end;
              textcolor(High_Light);
              gotoxy(X,Y+2);   write(FILENAME,EXT,'          ');
   end;   {procedure GET_FILENAME}

   procedure WRITE_DATA_TO_FILE;
      var
         CURR_ATTR: ATTR_PTR;
         CURR_EXTR: EXTR_PTR;
         CURR_DCSN: DCSN_PTR;
         CURR_VALU: VALU_PTR;
      begin
         DATA_SAVED := True;
         gotoxy(10,17);
         write('Now Saving: ',PATH,FILENAME,EXT);
         assign(OUTFILE,PATH+FILENAME+EXT);
         rewrite(OUTFILE);
         with MAIN_DATA do
            begin
               writeln( OUTFILE, length(QUESTION):1, ' ', QUESTION );
               writeln(  OUTFILE,  NUM_DCSN:1,  ' ',  NUM_ATTR:1,  ' '
NUM_VALU:1 );
            end;
         CURR_DCSN := MAIN_DATA.DCSN_LIST;
```

```
        while (CURR_DCSN <> nil ) do
           begin
              writeln(OUTFILE);
              with CURR_DCSN^ do
                 writeln( OUTFILE, length(DECISION):1, ' ', DECISION );
                 CURR_VALU := CURR_DCSN^.VALU_LIST;
                 I := 0;
                 while (CURR_VALU <> nil ) do
                    begin
                       I := I + 1;
                       with CURR_VALU^ do
                          if (I <= 4) then
                             write( OUTFILE, ' ', VALUE:1:4 )
                          else
                             begin
                                I := 0;
                                writeln( OUTFILE, ' ', VALUE:1:4 );
                             end;
                       CURR_VALU := CURR_VALU^.NEXT_VALU;
                    end;
                 if (I > 0) then
                    writeln(OUTFILE);
              CURR_DCSN := CURR_DCSN^.NEXT_DCSN;
           end;
        CURR_ATTR := MAIN_DATA.ATTR_LIST;
        while (CURR_ATTR <> nil ) do
           begin
              writeln(OUTFILE);
              with CURR_ATTR^ do
                 writeln( OUTFILE, length(ATTRIBUTE):1, ' ', ATTRIBUTE,
' ', NUM_EXTR:1 );
              CURR_EXTR := CURR_ATTR^.EXTR_LIST;
              while (CURR_EXTR <> nil ) do
                 begin
                    with CURR_EXTR^ do
                       writeln(  OUTFILE,  length(EXTREME):1,  '  ',
EXTREME );
                    CURR_VALU := CURR_EXTR^.VALU_LIST;
                    I := 0;
                    while (CURR_VALU <> nil ) do
                       begin
                          I := I + 1;
                          with CURR_VALU^ do
                             if (I <= 4) then
                                write( OUTFILE, ' ', VALUE:1:4 )
                             else
                                begin
                                   I := 0;
                                   writeln( OUTFILE, ' ', VALUE:1:4 );
                                end;
                          CURR_VALU := CURR_VALU^.NEXT_VALU;
                       end;
```

```
                    CURR_EXTR := CURR_EXTR^.NEXT_EXTR;
                end;
             CURR_ATTR := CURR_ATTR^.NEXT_ATTR;
          end;
       close(OUTFILE);
    end;    {procedure WRITE_DATA_TO_FILE}

begin
    if (NOT NO_DATA_FLAG) then
       begin
          Menu_Screen(3);
          textcolor(High_Light);
          gotoxy(1,25);     write('ESC');
          gotoxy(16,25);    write(chr(17),chr(196),chr(217));
          textcolor(Norm_Text);
          gotoxy(4,25);     write('-Main Menu');
          gotoxy(19,25);    write('-SaveFile!');
          DRAW_BORDER(3,5,78,19,ord(Shadow));
          X := INIT_XPOS;    Y := INIT_YPOS;
          gotoxy(X-11,Y);      write('DIRECTORY:');
          gotoxy(X-10,Y+2);    write('FILENAME:');
          gotoxy(5,Y+5);       write('QUESTION:');
          textcolor(High_Light);
          gotoxy(X,Y);
          I := length(PATH);
          if (PATH[I] <> '\') then
             begin
                I := I + 1;
                PATH[0] := chr(I);
                PATH[I] := '\';
             end;
          write(PATH);
          gotoxy(X,Y+2);    write(FILENAME);
          gotoxy(X+8,Y+2); write(EXT);
          gotoxy(X-7,Y+5);    write(MAIN_DATA.QUESTION);
          WHICH_KEY := ord(ErrorKey);
          repeat
             GET_FILENAME;
          until (VALID_FILE) or (WHICH_KEY = ord(Escape));
          if (WHICH_KEY <> ord(Escape)) and (VALID_FILE) then
             WRITE_DATA_TO_FILE;
          textcolor(Norm_Text);
       end
    else          {no data to save!}
       begin
          DRAW_BORDER(24,19-6,60,22-6,ord(Shadow));
          sound(500);   delay(50);   nosound;
          textcolor(High_Light);
          gotoxy(31,20-6);    write('No data entered yet!');
          textcolor(Norm_Text);
          delay(2000);
          clrscr;
```

```
        end;
   end;   {procedure SAVE_DATA}


procedure Calc_Rules;
begin
   Menu_Screen(2);
   Gotoxy(1,25);TextColor(High_Light);Write('ESC');
   TextColor(Norm_Text);Write('-Main Menu');
   window(2,3,79,23);
   if (No_data_Flag) then
     begin
      {gotoxy(25,09);write('Error, No data on file!')}
       DRAW_BORDER(24,19-6,60,22-6,ord(Shadow));
       sound(500);   delay(50);   nosound;
       textcolor(High_Light);
       gotoxy(31,20-6);   write('No data entered yet!');
       textcolor(Norm_Text);
       delay(2000);
       clrscr;
     end
   else
      Calc(Main_Data);
end;{Calc_Rules Procedure}


procedure View_Rules;
begin
   Menu_Screen(4);
   Gotoxy(1,25);TextColor(High_Light);Write('ESC');
   TextColor(Norm_Text);Write('-Main Menu');
   window(2,3,79,23);

   repeat
     clrscr;
     gotoxy(25,09);Write('Press "C" to view certain rules');
     gotoxy(25,10);Write('or "P" to  view possible rules');
     ch := upcase(readkey);
     case ch of
         'C' :  view(certain);
         'P' :  view(possible);
         end;
   until ch = esc_key;
end;{View_Rules Procedure}


Procedure CLEAR_SYSTEM_PARAMETERS;
   var
      CURR_DCSN: DCSN_PTR;
      CURR_ATTR: ATTR_PTR;
      CURR_EXTR: EXTR_PTR;
   procedure CLEAR_VALU(var PREV_VALU: VALU_PTR);
```

```
      var CURR_VALU: VALU_PTR;
      begin
         CURR_VALU := PREV_VALU;
         if (CURR_VALU^.NEXT_VALU <> nil) then
             CLEAR_VALU(CURR_VALU^.NEXT_VALU);
         dispose(PREV_VALU);
      end;
   procedure CLEAR_EXTR(var PREV_EXTR: EXTR_PTR);
      var CURR_EXTR: EXTR_PTR;
      begin
         CURR_EXTR := PREV_EXTR;
         if (CURR_EXTR^.NEXT_EXTR <> nil) then
            CLEAR_EXTR(CURR_EXTR^.NEXT_EXTR);
         if (MAIN_DATA.NUM_VALU > 0) then
            CLEAR_VALU(CURR_EXTR^.VALU_LIST);
         dispose(PREV_EXTR);
      end;
   procedure CLEAR_DCSN(var PREV_DCSN: DCSN_PTR);
      var CURR_DCSN: DCSN_PTR;
      begin
         CURR_DCSN := PREV_DCSN;
         if (CURR_DCSN^.NEXT_DCSN <> nil) then
            CLEAR_DCSN(CURR_DCSN^.NEXT_DCSN);
         if (MAIN_DATA.NUM_VALU > 0) then
            CLEAR_VALU(CURR_DCSN^.VALU_LIST);
         dispose(PREV_DCSN);
      end;
   procedure CLEAR_ATTR(var PREV_ATTR: ATTR_PTR);
      var CURR_ATTR: ATTR_PTR;
      begin
         CURR_ATTR := PREV_ATTR;
         if (CURR_ATTR^.NEXT_ATTR <> nil) then
            CLEAR_ATTR(CURR_ATTR^.NEXT_ATTR);
         if (PREV_ATTR^.NUM_EXTR > 0 ) then
            CLEAR_EXTR(CURR_ATTR^.EXTR_LIST);
         dispose(PREV_ATTR);
      end;

   begin
      DRAW_BORDER(10,12,70,17,ord(Shadow));
      sound(500);   delay(50);   nosound;
      textcolor(High_Light);
      gotoxy(15,14);   write('Are you sure you want to clear all data
(Y/N): ');
      textcolor(Norm_Text);
      repeat
         CHOICE := readkey;
         CHOICE := upcase(CHOICE);
      until (CHOICE in ['Y','N']);
      if (CHOICE = 'Y') then
         write('Yes')
      else
```

```
         write('No');
      delay(75);
      if (CHOICE = 'Y' ) then
         begin
            {dispose off all pointers &}
            {free the memory for re-use }
            if (MAIN_DATA.NUM_DCSN > 0) then
               CLEAR_DCSN(MAIN_DATA.DCSN_LIST);
            if (MAIN_DATA.NUM_ATTR > 0) then
               CLEAR_ATTR(MAIN_DATA.ATTR_LIST);
            NO_DATA_FLAG := True;  DATA_SAVED := False;
            FILENAME := DEF_FILENAME;
            MAIN_DATA.QUESTION := '';
            MAIN_DATA.NUM_ATTR := 0;
            MAIN_DATA.ATTR_LIST := nil;
            MAIN_DATA.NUM_DCSN := 0;
            MAIN_DATA.DCSN_LIST := nil;
            MAIN_DATA.NUM_VALU := 0;
            TOP_ATTR := 1;   ATTR_ROW := 1;
            TOP_EXTR := 1;   EXTR_ROW := 1;
            TOP_DCSN := 1;   DCSN_ROW := 1;
         end;
   end;   {procedure CLEAR_SYSTEM_PARAMETERS}


procedure Help;
      begin
         Menu_Screen(5);
         GotoXY(26,12);
         Writeln('No help currently available!');
         Repeat until readkey = ESC_KEY;
      end;{Help Procedure}


{*************************************************************}
{*************************************************************}
{***                                                       ***}
{***                    MAIN PROGRAM                       ***}
{***                                                       ***}
{*************************************************************}
{*************************************************************}
Begin
   LOGO_DISPLAY;
   NO_DATA_FLAG := True;
   DATA_SAVED := False;
   FILENAME := DEF_FILENAME;
   EXT := DEF_EXT;
   getdir(0,PATH);
   MAIN_DATA.QUESTION := '';
   MAIN_DATA.NUM_ATTR := 0;
   MAIN_DATA.ATTR_LIST := nil;
   MAIN_DATA.NUM_DCSN := 0;
```

```
   MAIN_DATA.DCSN_LIST := nil;
   MAIN_DATA.NUM_VALU := 0;
   TOP_ATTR := 1;    ATTR_ROW := 1;
   TOP_EXTR := 1;    EXTR_ROW := 1;
   TOP_DCSN := 1;    DCSN_ROW := 1;
   Norm_Text := LightGray;
   Norm_BackGround := Black;
   High_Light := White;
   Row := 0;
   repeat
      QUIT := False;
      Row := Menu_Choice(Row);
      case Row of
         0 :  INPUT_SYSTEM_DATA;
         1 :  Retrieve_data;
         2 :  Calc_Rules;
         3 :  SAVE_DATA;
         4 :  View_rules;
         5 :  Help;
         6 :  QUIT := True;
      end;
      if (QUIT) then
         if (NOT DATA_SAVED) and (NOT NO_DATA_FLAG) then
            begin
               gotoxy(12,20);
               write('Data NOT saved! Do you want to save your data
(Y/N): ');
               repeat
                  CHOICE := readkey;
                  CHOICE := upcase(CHOICE);
               until ( CHOICE = 'Y' ) or ( CHOICE = 'N' );
               if ( CHOICE = 'Y' ) then
                  SAVE_DATA
               else
                  DATA_SAVED := True;
            end
         else  { data saved or main_data = '' )
            DATA_SAVED := True;
   until (QUIT) and (DATA_SAVED);
   clrscr;
End.
```

```
unit DM_Calc;

(*************************************************************************
***                                                                    ***
***  DM_Calc is a Turbo Pascal unit. It calculates all possible        ***
***  combinations of fuzzy sets and generates rules and fuzzy          ***
***  values. It was written by Kevin B. Walker.                        ***
***                                                                    ***
*************************************************************************)

interface

uses crt, IO_PROC;

procedure Calc(var Main_Data : Question_Rec);

implementation

procedure Calc(var Main_Data: Question_Rec);

const
  certain = True;
  possible = False;

var
  p : attr_ptr;
  i : integer;
  Actv_Dcsn : Dcsn_Ptr;
  Temp_Attr : Attr_Ptr;
  memP : pointer;
  C_file : text;
  P_file : text;
  I_Alpha, J_Alpha : real;

  procedure Enter_Values;

  var
     val : real;
     i,j,k,n : integer;
     Attr : Attr_ptr;
     Extr : Extr_Ptr;
     Dcsn : Dcsn_Ptr;
     Valu : Valu_Ptr;

  begin
     ClrScr;
     Write('Enter Number of data sets >');
     Readln(Main_data.Num_valu);
     For i := 1 to Main_data.Num_Valu do
     begin
        ClrScr;
        Writeln('Data set #',i);
```

```
        Attr := Main_data.Attr_List;
        Dcsn := Main_data.Dcsn_List;
        for j := 1 to Main_data.Num_Attr do
        begin
           Extr := Attr^.Extr_List;
           for k := 1 to Attr^.Num_Extr do
           begin
              New(Valu);
              Write('How ',Extr^.Extreme,' is ',Attr^.Attribute,'?
>');
              Readln(Valu^.Value);
              Valu^.Next_Valu := Extr^.Valu_List;
              Extr^.Valu_List := Valu;
              Extr := Extr^.Next_Extr;
           end;{for k}
           Attr := Attr^.Next_Attr;
        end;{for j}
        for n := 1 to Main_data.Num_Dcsn do
        begin
           New(Valu);
           Write('How much do you believe ',Dcsn^.Decision,'? >');
           Readln(Valu^.Value);
           Valu^.Next_Valu := Dcsn^.Valu_List;
           Dcsn^.Valu_List := Valu;
           Dcsn := Dcsn^.Next_Dcsn;
        end;{for n}
     end;{for i}
     clrscr;
  end;{Enter Values}



     {*****************************}
     function Max(x,y:real):real;
     begin
        if x > y then Max := x
        else Max := y
     end;{ Max ********************}

     {*****************************}
     function Min(x,y:real):real;
     begin
        if x < y then Min := x
        else Min := y
     end;{ Min ********************}

  {*********************************}
  function Calc_I(Extr_Val:Valu_Ptr;
                  Dcsn_Val:Valu_Ptr)
                  :real;
  var
```

```
   temp_max, temp_min : real;
   i : integer;

begin
   temp_max := -100.0;
   temp_min := 100.0;
   for i := 1 to Main_Data.Num_valu do
   begin
      temp_max := Max(1-Extr_Val^.value, Dcsn_Val^.value);
      temp_min := Min(temp_max,temp_min);
      Extr_Val := Extr_Val^.Next_Valu;
      Dcsn_Val := Dcsn_Val^.Next_Valu;
   end;{for i}
   Calc_I := temp_min;
end;{ Calc I**********************}

{*********************************}
function Calc_J(Extr_Val:Valu_Ptr;
                Dcsn_Val:Valu_Ptr)
                :real;
var
   temp_max, temp_min : real;
   i : integer;

begin
   temp_max := -100.0;
   temp_min := 100.0;
   for i := 1 to Main_Data.Num_valu do
   begin
      temp_min := Min(Extr_Val^.value, Dcsn_Val^.value);
      temp_max := Max(temp_max,temp_min);
      Extr_Val := Extr_Val^.Next_Valu;
      Dcsn_Val := Dcsn_Val^.Next_Valu;
   end;{for i}
   Calc_J := temp_max;
end; {Calc J **********************}

{***************************************}
function Calc_Intersect(A_Attr:Attr_Ptr;
                        B_Attr:Attr_Ptr)
                        :Attr_Ptr;
var
   temp_min : real;
   i : integer;
   AnB : Attr_Ptr;
   AB_val, A_val, B_val : Valu_Ptr;

   {*******************************}
   procedure trash(Attr:Attr_Ptr);
   {Moves down the linked list of data values disposing the memory
    allocated for each one and then disposes the attribute and
    extreme memory as well}
```

```
     var
        i :integer;
        Temp_Valu : Valu_Ptr;

     begin  {trash}
        if Attr^.Actv_Extr^.Extreme = 'temp' then
        begin
           for i := 1 to Main_Data.Num_Valu do
           begin
              Temp_Valu := Attr^.Actv_Extr^.Valu_List;
              Attr^.Actv_Extr^. Valu_List := Temp_Valu^.Next_Valu;
              dispose(Temp_Valu);
           end;
           dispose(Attr);
        end
        else
          Writeln('Tried to trash good data!');
     end; { Trash ********************}

  begin
     if A_Attr^.Actv_Extr^.Extreme = 'none' then
        Calc_Intersect := B_Attr            { A label of 'none'
indicates}
     else                                   {that extreme is not
selected}
        if B_Attr^.Actv_Extr^.Extreme = 'none' then  {and no
intersection}
           Calc_Intersect := A_Attr                      {is
calculated}
     else
     begin
        temp_min := 100.0;
        new(AnB);
        AnB^.Actv_Extr^.Extreme := 'temp';  {Named temp so we can
dispose}
        new(AnB^.Actv_Extr^.Valu_List);                  {of it
later}

        A_val := A_Attr^.Actv_Extr^.Valu_List;  {Assign extreme values
from}
        B_Val := B_Attr^.Actv_Extr^.Valu_List;       {associated
attributes}
        AB_val := AnB^.Actv_Extr^.Valu_List;

        for i := 1 to Main_Data.Num_valu do
        begin
           AB_val^.Value := Min(A_Val^.value, B_Val^.value);
           A_Val := A_Val^.Next_Valu;        {Move down the linked
list}
           B_Val := B_Val^.Next_Valu;        {computing the min of
each}
           New(AB_val^.Next_Valu);           {fuzzy set and placing
```

```
the}
          AB_val := AB_val^.Next_Valu;          {result in the var
AnB}
        end;{for i}
        AB_val^.Next_Valu := NIL;
        Calc_Intersect := AnB;                  {Assign a return
value}
     end; {else}
     if A_Attr^.Actv_Extr^.Extreme = 'temp' then      {Release memory
containg two}
        Trash(A_Attr);                                {intersected
sets}
     if B_Attr^.Actv_Extr^.Extreme = 'temp' then
        Trash(B_Attr);
  end; { Calc_Intersect ********************}

  {*****************************************}
  function Recalc (Attr:Attr_Ptr) : Attr_Ptr;

  {Calls
      Calc_Intersect
  }
  begin  {function recalc}
     if Attr^.Next_Attr <> NIL then
        Recalc := Calc_Intersect(Attr, Attr^.Next_Attr)
     else
        Recalc := Attr;
  end;{ Recalc ******************************}

  {*****************************************}
  procedure process;

  {Calls
      Recalc}
  var
     One_or_More,MoreThanOne : boolean;
     Fst_Ltr : string;
     i : integer;
     I_Value, J_Value : real;
     Test_Attr, Attr, Calc_Attr : Attr_Ptr;
     Calc_Extr_Valu : Valu_Ptr;
     OutBuf, I_Val_Str, J_Val_Str : string ;


  begin {Process}
  Test_Attr := Main_Data.Attr_List;
  One_or_More := False;

  while Test_Attr^.Next_Attr <> NIL do
    begin
       if Test_Attr^.Actv_Extr^.Extreme = 'none' then
         {nothing}
```

```
        else
          One_or_More := True;
        Test_Attr := Test_Attr^.Next_Attr;
      end;{while}
      if One_or_More then
      begin
        OutBuf := '';
        Calc_Attr := Recalc(Main_Data.Attr_List);
        Calc_Extr_Valu := Calc_Attr^.Actv_Extr^.Valu_List;
        I_Value := Calc_I(Calc_Extr_Valu,Actv_Dcsn^.Valu_List);
        J_Value := Calc_J(Calc_Extr_Valu,Actv_Dcsn^.Valu_List);
        str(I_value:1:2, I_Val_Str);
        str(J_value:1:2, J_Val_Str);
        One_or_More := False;
        MoreThanOne := False;
        Attr := Main_Data.Attr_List;
        for i := 1 to Main_Data.Num_Attr do
        begin
           if Attr^.Actv_Extr^.Extreme <> 'none' then
             begin
                if One_Or_More then
                begin
                  MoreThanOne := True;
                  OutBuf := OutBuf + 'and ';
                  OutBuf := OutBuf + Attr^.Actv_Extr^.Extreme + ' ';
                end
                else
                begin
                   One_or_More := True;
                   Fst_Ltr := copy(Attr^.Actv_Extr^.Extreme,1,1);
                   OutBuf := OutBuf + upcase(Fst_ltr[1]);
                   OutBuf := OutBuf +
copy(Attr^.Actv_Extr^.Extreme,2,14) + ' ';
                end;{else} { above allows capitalization }
             end;
           Attr := Attr^.Next_Attr;
        end;{for i}
        if One_or_More then
          begin
            if MoreThanOne then
              OutBuf := OutBuf + 'suggest '
            else
              OutBuf := OutBuf + 'suggests ';
            OutBuf := OutBuf + Actv_Dcsn^.Decision;
            OutBuf := OutBuf + ' with belief level ';
            if I_Value >= I_Alpha then
                Writeln(C_file,OutBuf,I_Val_Str);
            if J_Value >= J_Alpha then
                Writeln(P_file,OutBuf,J_Val_Str);
          end;{second if one or more}
      end;{first if one or more}
    end;{ Process ******************************}
```

```
  {*******************************}
  procedure Visit(Attr :Attr_Ptr);
  var
     Last_Extr : Extr_Ptr;

  begin

    if (Attr^.Next_Attr <> NIL) then                  {if more
attributes}
    begin
       visit(Attr^.Next_Attr);                    {go to the next
Attribute}
    end;
    if (Attr^.Actv_Extr^.Next_Extr <> NIL) then        {if more
extremes}
    begin
       Last_Extr := Attr^.Actv_Extr;              {set last var for
return}
       Attr^.Actv_Extr := Attr^.Actv_Extr^.Next_Extr;
       visit(Attr);               {go to the next extreme, same
attribute}
        Attr^.Actv_Extr := Last_Extr;
    end;
    if (Attr^.Next_Attr = NIL ) then
      process;                        {only process when at last
attribute}
  end; { Visit ************************}



begin { Calc }
  Assign(C_file,'C_RULES.TXT');
  Assign(P_file,'P_RULES.TXT');
  rewrite(C_file);
  rewrite(P_file);

  Enter_Values;                                    { diagnostic procedure}
  gotoxy(20,09);Write('Enter a lower limit for "Certain" rules');
  gotoxy(25,10);Write('>');Readln(I_Alpha);
  clrscr;
  gotoxy(20,09);Write('Enter a lower limit for "Possible" rules');
  gotoxy(25,10);Write('>');Readln(J_Alpha);
  clrscr;
  mark(memP);
  Temp_Attr := Main_Data.Attr_List;
  for i := 1 to Main_Data.Num_Attr do   {sets each Attribute's
Actv_Extr}
  begin                                 {      to a default named
'none'}
     new(Temp_Attr^.Actv_Extr);
     Temp_Attr^.Actv_Extr^.Extreme := 'none';
     Temp_Attr^.Actv_Extr^.Next_Extr := Temp_Attr^.Extr_List;
```

```
      Temp_Attr := Temp_Attr^.Next_Attr;
    end;
    Actv_Dcsn := Main_Data.Dcsn_List;
    for i := 1 to Main_Data.Num_Dcsn do   {Calls visit once for each
Decision}
    begin
      visit(Main_Data.Attr_List);                { visit goes through
all }
      Actv_Dcsn := Actv_Dcsn^.Next_Dcsn;         {combinations of
extremes}
      writeln(P_file);
      writeln(C_file);
    end;
    release(memP);
    close(C_file);
    close(P_file);
end;{calc}

begin

end.{DM_Calc}
```

```
unit DM_View;

(*************************************************************************
***                                                                    ***
***   DM_View is a Turbo Pascal Unit. It allows a user to view         ***
***   rules generated by the DM_Calc unit. It was written by           ***
***   Kevin Walker.                                                    ***
***                                                                    ***
*************************************************************************)

interface

uses crt, IO_PROC;

type
   RuleType = (Certain , Possible);

procedure view(Rule : Ruletype);

implementation
procedure view(Rule : Ruletype);

const
   Screensize = 21;
   ScreenWidth = 78;
   Space_Key = ' ';

var
   Infile : text;
   buff   : string;
   i, lines : integer;
   quit : boolean;
   FileExists : boolean;

begin
   clrscr;
   if rule = certain then
      assign(infile, 'c_rules.txt')
   else
      assign(infile, 'p_rules.txt');

   {$I-}
   reset(infile);
   {$I+}
   FileExists := (IOResult = 0);

   if (FileExists) then
    begin
      quit := false;
      lines := 1;
      repeat
      if (not EOF(infile)) then
```

```
          if (lines < screensize) then
           begin
             readln(infile,buff);
             writeln(buff);

             lines := lines + 1 + (length(buff) div 80);
           end
          else
           begin
             writeln('<<Press space to continue>>');
             repeat until readkey = space_key;
             lines := 1;
             gotoxy(1,screensize-1);
             for i := 1 to ScreenWidth do write(' ');
           end
        else
         begin
           writeln('<<Press Escape to exit>>');
           repeat until readkey = esc_key;
           quit := true;
         end;
        until quit;
        close(infile);
      end
       else
        begin
         gotoxy(25,09);write('Error, File not found.');
         gotoxy(25,10);Write('Press Enter to continue');
         Readln;
        end

end;{procedure view}

begin

end.
```

```
Unit IO_PROC;

(*************************************************************************
***                                                                    ***
*** IO_PROC is a Turbo Pascal unit. It takes input from the user      ***
*** and builds the data structure. It was written by  Danny          ***
*** Picazo. A newer version of this file exists.                      ***
***                                                                    ***
*************************************************************************)

Interface

   Uses
      CRT,GRAPH;

   Const
      {Color Constants}
      {===============}
      Norm_Text = LightGray;
      High_Light = White;
      Norm_BackGround = Black;

      {Keyboard Return Code Constants}
      {==============================}
      NULL = #0;
      ESC_KEY = #27;     CR = #13;
      INS_KEY = #82;     DEL_KEY = #83;     BACKSPC = #8;
      TAB_KEY = #9;      SHFT_TAB = #15;
      UP_ARROW = #72;    DN_ARROW = #80;    LF_ARROW = #75;    RT_ARROW =
#77;
      HOME_KEY = #71;    END_KEY = #79;     PGUP_KEY = #73;    PGDN_KEY =
#81;
      F1_KEY = #59;      F2_KEY = #60;      F3_KEY = #61;      F4_KEY =
#62;
      ALT_Q = #16;       ALT_D = #32;       ALT_A = #30;       ALT_S =
#31;

      {Border ASCII Char Codes}
      {======================}
      LT = 201;    RT = 187;    HORIZ = 205;
      LB = 200;    RB = 188;    VERT = 186;
      SHADOW_CHAR = 176;

   Type
      MESSAGES = array[0..6] of string;

      ATTR_PTR = ^ATTRIBUTE_REC;
      EXTR_PTR = ^EXTREME_REC;
      DCSN_PTR = ^DECISION_REC;
      VALU_PTR = ^VALUE_REC;

      EXTREME_REC = record
```

```
        EXTREME: string;
      NEXT_EXTR: EXTR_PTR;
      VALU_LIST: VALU_PTR;
            end;

   VALUE_REC = record
          VALUE: REAL;
      NEXT_VALU: VALU_PTR;
            end;

   ATTRIBUTE_REC = record
       ATTRIBUTE: string;
       NEXT_ATTR: ATTR_PTR;
        NUM_EXTR: integer;
       EXTR_LIST: EXTR_PTR;
       Actv_Extr: Extr_Ptr;
             end;

   DECISION_REC = record
       DECISION: string;
      NEXT_DCSN: DCSN_PTR;
      VALU_LIST: VALU_PTR;
            end;

   QUESTION_REC = record
       QUESTION: string;
       NUM_ATTR: integer;
      ATTR_LIST: ATTR_PTR;
       NUM_DCSN: integer;
       NUM_VALU: integer;
      DCSN_LIST: DCSN_PTR;
            end;

   SHADOW_CHOICE = ( Shadow, NoShadow );

   KEY_TYPE = ( AlphaNum, BackSpace, TabKey, ShftTab, HomeKey,
EndKey, UpKey,
                DnKey, LfKey, RtKey, Escape, F1Key, F2Key, F3Key,
F4Key,
                EnterKey, ErrorKey, AltS, AltD, AltA, AltQ,
PgUpKey, PgDnKey,
                InsKey, DelKey, EraseKey );

Function  WHICH_KEY_PRESSED( var CHOICE: char ): integer;

Procedure CLEAR_WINDOW( X1, Y1, X2, Y2: integer );

Procedure DRAW_BORDER( X1, Y1, X2, Y2, SHADOW_FLAG: integer );

Procedure LOGO_DISPLAY;
```

```
   Procedure Menu_Screen( Item: integer );

   Function  Menu_Choice( Prev_Row : integer ): integer;

   Procedure GET_INPUT_STRING( X_INIT, Y_INIT, MAX_FIELD_LENGTH:
integer;
                               var INPUT_STRING: string;
                               var WHICH_KEY: integer;
                               var EXIT_KEY_PRESSED: boolean );


Implementation


   Function WHICH_KEY_PRESSED;
      var   TEMP: KEY_TYPE;

      begin
         TEMP := ErrorKey;
         CHOICE := readkey;
         case (CHOICE) of
            ' '..'~': TEMP := AlphaNum;
                  CR: TEMP := EnterKey;
             BACKSPC: TEMP := BackSpace;
             TAB_KEY: TEMP := TabKey;
             ESC_KEY: TEMP := Escape;
                else  begin
                         if ( CHOICE = NULL ) then
                            begin
                               CHOICE := readkey;
                               case (CHOICE) of
                                     ALT_Q: TEMP := AltQ;
                                     ALT_D: TEMP := AltD;
                                     ALT_S: TEMP := AltS;
                                     ALT_A: TEMP := AltA;
                                    F1_KEY: TEMP := F1Key;
                                    F2_KEY: TEMP := F2Key;
                                    F3_KEY: TEMP := F3Key;
                                    F4_KEY: TEMP := F4Key;
                                  UP_ARROW: TEMP := UpKey;
                                  DN_ARROW: TEMP := DnKey;
                                  LF_ARROW: TEMP := LfKey;
                                  RT_ARROW: TEMP := RtKey;
                                  PGUP_KEY: TEMP := PgUpKey;
                                  PGDN_KEY: TEMP := PgDnKey;
                                  HOME_KEY: TEMP := HomeKey;
                                   END_KEY: TEMP := EndKey;
                                  SHFT_TAB: TEMP := ShftTab;
                                   INS_KEY: TEMP := InsKey;
                                   DEL_KEY: TEMP := DelKey;
                                      else  TEMP := ErrorKey;
                               end;   {case special CHOICE}
```

```
                              end;   {if NULL }
                           end;   {case-else}
                     end;   {case CHOICE}
                  WHICH_KEY_PRESSED := ord(TEMP);
               end;   {function WHICH_KEY_PRESSED}


   Procedure CLEAR_WINDOW;
      begin
         window(X1,Y1,X2,Y2);
         clrscr;
         window(1,1,80,25);
      end;   {procedure CLEAR_WINDOW}


Procedure DRAW_BORDER;
   var
      I, SHDW_OFFSET: integer;

   begin
      CLEAR_WINDOW(X1,Y1,X2,Y2);
      SHDW_OFFSET := 0;
      if ( SHADOW_FLAG = ord(Shadow) ) then
         SHDW_OFFSET := 1;
      textcolor(Norm_Text);
      gotoxy(X1,Y1);                                    write( chr(LT) );
      gotoxy(X2-SHDW_OFFSET,Y1);                        write( chr(RT) );
      gotoxy(X1,Y2-SHDW_OFFSET);                        write( chr(LB) );
      gotoxy(X2-SHDW_OFFSET,Y2-SHDW_OFFSET);            write( chr(RB) );
      for I := (X1+1) to (X2-1-SHDW_OFFSET) do
         begin
            gotoxy(I,Y1);                          write( chr(HORIZ) );
            gotoxy(I,Y2-SHDW_OFFSET);              write( chr(HORIZ) );
         end;
      for I := (Y1+1) to (Y2-1-SHDW_OFFSET) do
         begin
            gotoxy(X1,I);                          write( chr(VERT) );
            gotoxy(X2-SHDW_OFFSET,I);              write( chr(VERT) );
         end;
      if ( SHADOW_FLAG = ord(Shadow) ) then
         begin
            for I := Y1+1 to Y2 do
               begin   gotoxy(X2,I);   write( chr(SHADOW_CHAR) );
end;
            for I := X1+1 to X2 do
               begin   gotoxy(I,Y2);   write( chr(SHADOW_CHAR) );
end;
         end;
   end;   {procedure DRAW_BORDER}


Procedure GET_INPUT_STRING;
```

```
   var
                I,
      X, Y, XPOS,
      STR_LENGTH: integer;
          CHOICE: char;
     INSERT_CHAR: boolean;
   begin
      textcolor(Norm_Text);
      gotoxy(60,22);   write( 'MODE:' );
      textbackground(LightGray);   textcolor(Black);   write(
'Replace' );
      textbackground(Black);   textcolor(LightGray);
      X := X_INIT;   Y := Y_INIT;
      textbackground(LightGray);
      textcolor(Black);
      gotoxy(X,Y);
      STR_LENGTH := length(INPUT_STRING);
      if ( STR_LENGTH > 0 ) then
         begin
            write( INPUT_STRING );
            for I := 1 to (MAX_FIELD_LENGTH - STR_LENGTH) do
               write(' ');
         end
      else
         for I := 1 to MAX_FIELD_LENGTH do
            write(' ');
      X := X_INIT + STR_LENGTH;
      INSERT_CHAR := False;
      repeat
         gotoxy(X,Y);
         WHICH_KEY := WHICH_KEY_PRESSED(CHOICE);
         case ( WHICH_KEY ) of
             ord(AlphaNum): begin
                               X := X + 1;
                               XPOS := X - X_INIT;
                               if ( XPOS > MAX_FIELD_LENGTH ) or
                                  ( (STR_LENGTH >= MAX_FIELD_LENGTH)
and
                                   (INSERT_CHAR) ) then
                                 begin
                                   X := X - 1;  sound(100);  delay(50);
nosound;
                                 end
                               else
                                 if ( INSERT_CHAR ) then
                                    begin
                                      for I := STR_LENGTH downto XPOS
do
                                        INPUT_STRING[I+1] :=
INPUT_STRING[I];
                                      INPUT_STRING[XPOS] := CHOICE;
                                      STR_LENGTH := STR_LENGTH + 1;
```

```
                                        INPUT_STRING[0] :=
chr(STR_LENGTH);
                                        gotoxy(X_INIT,Y_INIT);
                                        write(INPUT_STRING);
                                      end
                                    else
                                      begin
                                        write(CHOICE);
                                        if ( XPOS >= STR_LENGTH ) then
                                          STR_LENGTH := STR_LENGTH + 1;
                                        INPUT_STRING[XPOS] := CHOICE;
                                      end;
                                  end;   {Alphanum}
                        ord(HomeKey): X := X_INIT;
                         ord(EndKey): X := X_INIT + STR_LENGTH;
                          ord(LfKey): begin
                                        X := X - 1;
                                        if ( X < X_INIT ) then
                                          begin
                                            X := X + 1;  sound(100);  delay(50);
nosound;
                                          end;
                                      end;   {LfKey}
                          ord(RtKey): begin
                                        X := X + 1;
                                        if ( X > (X_INIT + STR_LENGTH) ) then
                                          begin
                                            X := X - 1;  sound(100);  delay(50);
nosound;
                                          end;
                                      end;   {RtKey}
                         ord(InsKey): begin
                                        if ( INSERT_CHAR ) then
                                          begin
                                            INSERT_CHAR := False;
                                            gotoxy(65,22);   write('Replace');
                                          end
                                        else
                                          begin
                                            INSERT_CHAR := True;
                                            gotoxy(65,22);   write('Insert ');
                                          end;
                                      end;   {InsKey}
                         ord(DelKey): begin
                                        XPOS := (X - X_INIT) + 1;
                                        if (STR_LENGTH > 0) and (XPOS <=
STR_LENGTH) then
                                          begin
                                            delete(INPUT_STRING,XPOS,1);
                                            STR_LENGTH := STR_LENGTH - 1;
                                            INPUT_STRING[0] := chr(STR_LENGTH);
                                            gotoxy(X_INIT,Y_INIT);
```

```
                                    write(INPUT_STRING,' ');
                                  end
                                else if (STR_LENGTH = 0) then
                                  begin
                                    WHICH_KEY := ord(EraseKey);
                                    EXIT_KEY_PRESSED := True;
                                  end;
                              end;   {DelKey}
            ord(BackSpace): begin
                              XPOS := X - X_INIT;
                              if (STR_LENGTH >= 1) and (XPOS > 0) then
                                begin
                                  delete(INPUT_STRING,XPOS,1);
                                  STR_LENGTH := STR_LENGTH - 1;
                                  INPUT_STRING[0] := chr(STR_LENGTH);
                                  gotoxy(X_INIT,Y_INIT);
                                  write(INPUT_STRING,' ');
                                  X := X - 1;
                                end
                              else if (STR_LENGTH = 0) then
                                begin
                                  WHICH_KEY := ord(EraseKey);
                                  EXIT_KEY_PRESSED := True;
                                end;
                            end;   {BackSpace}
                       else  EXIT_KEY_PRESSED := True {do nothing};
        end;   {case WHICH_KEY}
      until ( EXIT_KEY_PRESSED );
      INPUT_STRING[0] := chr(STR_LENGTH);
      X := X_INIT;   Y := Y_INIT;
      textbackground(Black);
      textcolor(LightGray);
      gotoxy(X,Y);
      write( INPUT_STRING );
      for I := 1 to (MAX_FIELD_LENGTH - length(INPUT_STRING)) do
        write(' ');
  end;   {procedure GET_INPUT_STRING}


procedure Menu_Screen(Item:integer);
 const
   Titles : messages = ('Input New Data','Retrieve Existing Data',
     'Calculate Rules','Save Data and Rules','View Rules',
     'Help','Main Menu');
  begin
      textbackground(Norm_BackGround);
      textcolor(Norm_Text);
      ClrScr;
      Draw_Border(1,2,80,24,ord(NoShadow));
      GotoXY(1,1);
      Write('Decision Maker: '); TextColor(White);
Write(Titles[Item]);
```

```
      TextColor(Norm_Text);
      Gotoxy(64,1);Write('KanDev 1992, v1.0');
  end;(Menu_Screen)


Function Menu_Choice(Prev_Row : integer): integer;
   type
     messages = array[0..6] of string;
   const
     Start_Col = 27;
     Start_Row = 10;
     Bottom_Row = 16;
     Menu_Item : Messages = ('Input New Data','Retrieve Existing
Data',
     'Calculate Rules','Save Data and Rules','View Rules',
     'Help','Exit Program');
   var
      ch : char;
      I,Scrn_Col, Scrn_Row : integer;
   begin (function Menu_choice)
      Menu_Screen(6);
      GotoXY(Start_Col-2,Start_Row-3);
      Write('Use ',chr(24),' and ',chr(25),' to select a menu item');
      GotoXY(Start_Col-2,Start_Row-2);
      Write('then press Enter to execute.');

Draw_Border(Start_Col-2,Start_Row-1,Start_Col+31,Bottom_Row+2,ord(Sha
ow));
      for I := 0 to 6 do
         begin
            GotoXY(start_col,Start_Row+I);
            Writeln(Menu_item[I]);
         end;
      Scrn_Col := start_col;
      Scrn_Row := Start_Row + Prev_Row;
      gotoxy(Scrn_Col, Scrn_Row);
      TextBackground(Norm_Text);
      TextColor(Norm_BackGround);
      Writeln(Menu_item[Prev_Row]);
      repeat
         GotoXY (Scrn_Col,Scrn_Row);
         textbackground(Norm_BackGround);
         textcolor(Norm_Text);
         ch := readkey;
         if ch = Null then
            begin
               ch := readkey;
               Write(menu_item[Scrn_Row - Start_Row]);
               case ch of
                  Up_Arrow : if ( Scrn_Row = Start_Row ) then
                                Scrn_Row := bottom_row
                             else
```

```
                                Scrn_Row := Scrn_Row - 1;
                        Dn_Arrow : if ( Scrn_Row = bottom_row ) then
                                Scrn_Row := Start_Row
                             else
                                Scrn_Row := Scrn_Row + 1;
                            else ; {do nothing}
                        end;{case}
                     GotoXY(Scrn_Col,Scrn_Row);
                     TextBackground(Norm_Text);
                     TextColor(Norm_BackGround);
                     Writeln(menu_item[Scrn_Row - Start_Row]);
                     GotoXY(Scrn_Col,Scrn_Row);
                end; {if}
        until ch = CR; {Carriage Return Pressed}
        Menu_choice := Scrn_Row - Start_Row;
    end;{Menu_Choice}


    Procedure LOGO_DISPLAY;
        var
            X,Y,I,J,
            XMAX, YMAX,
            GRAPHMODE,GRAPHDRIVER: integer;
        begin
            clrscr;
            GRAPHDRIVER := detect;
            initgraph(GRAPHDRIVER,GRAPHMODE,'C:');
            XMAX := GetMaxX;   YMAX := GetMaxY;
            X := 5; Y := 5; I := 1; J := 1;
            repeat
                settextstyle(1,0,7);
                if (J < (Ymax/2 - 80)) then
                    begin
                        setcolor(Blue);   outtextxy(I,J,'K  D  ');   {draw}
                        setcolor(Green);  outtextxy(I,J,' an ev');
                    end;
                I := I + X;  J := J + Y;
            until ( J > (YMAX/2)-40 );
            setcolor(Red);  outtextxy(I,J,'KanDev');
            setcolor(WHITE);
            settextstyle(1,0,2);
            outtextxy((I-30),(J+70),'Software Solutions Presents...');
            I := 1;
            repeat
                I := I + 1;  delay(100);
            until(keypressed) or (I > 50);   {wait up to 5 seconds}
            cleardevice; closegraph; clrscr;
        end;   {procedure LOGO_DISPLAY}


End.   {Unit IO_PROC}
```