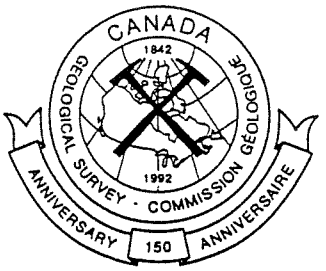


GEOLOGICAL SURVEY OF CANADA

OPEN FILE 2577

FAVMOD
for
integration of spatial geoscience information

Y. Leclerc, C. Chung

1993

Energy, Mines and Resources Canada
Énergie, Mines et Ressources Canada
Canada

Contribution to Canada-Alberta Agreement on Mineral Development, 1992-1995. Project funded by the Geological Survey of Canada.

Contribution à l'Entente Canada-Alberta sur l'exploitation minérale 1992-1995. Ce projet a été financé par la Commission géologique du Canada.

FAVMOD

for

Integration of Spatial Geoscience Information

Yvon Leclerc and Chang-Jo Chung***

**: Faculty of Environmental Design, University of Calgary, Calgary, Alberta*

***: Mineral Resources Division, Geological Survey of Canada, Ottawa, Ontario*

October, 1992

Geological Survey of Canada

Ottawa, Canada.

TABLE OF CONTENTS

1. Description of FAVMOD

FAVMOD software was written as a first step toward developing a prototype computer system for integration of spatial geoscience data at the Mineral Resources Division of the Geological Survey of Canada during a summer student term (COSEP) in 1992. The methods implemented in FAVMOD were developed by Chung and Fabbri (1992),[1] and Chung and Moon (1991).[2] Considered as a prototype for further development, your recommendations and feedback about FAVMOD would be greatly appreciated.

The software FAVMOD software is specially designed to combine layers of spatial geoscience information according to the model, referred to as the ***proposition***, as defined by the user. .The combination of these several layers, referred to as ***evidences,*** may be performed using one of the following techniques offered by FAVMOD:

i. Multivariate Regression Analysis
ii. Probabilistic Approach using Bayesian Rule
ii. Dempster-Shafer Belief's Functions
iv. Fuzzy Logic Approach
v. Certainty Factors Interpretation

The software offers menus for the fourth and fifth techniques, howerver, the techniques are not yet functional.
Each layer becomes an evidence for the proposition. For each evidence, each class is transformed into a known constant interval. The transformation is performed via the favorability function. The value generated by the favorability function represents the degrees of importance of the classes with reference to the proposition. This value is either data driven or expert prior knowledge of the evidence.

To integrate geo-reference information, FAVMOD uses some GIS (Geographic Information System) capabilities provided by the commercial available software named SPANS. To properly utilize FAVMOD, the user has to have basic knowledge of the theory that lies behind the five integration rules and of SPANS GIS.

---

[1] Chung, C-J, Fabbri, A. G., Representation of Geoscience Information for Data Integration, Presented at the 29th IGC, Kyoto, Japan, 1992.

[2] Chung, F. C.-J., Moon, W. M., Combination Rules of Spatial Geoscience Data for Mineral Exploration, Geoinformatics, Vol. 2, No. 2, pp. 159-169, 1991.

This document includes:

- The User Manual
- The Software Constraints
- The Installation Procedure and System Requirements

2. User Manual

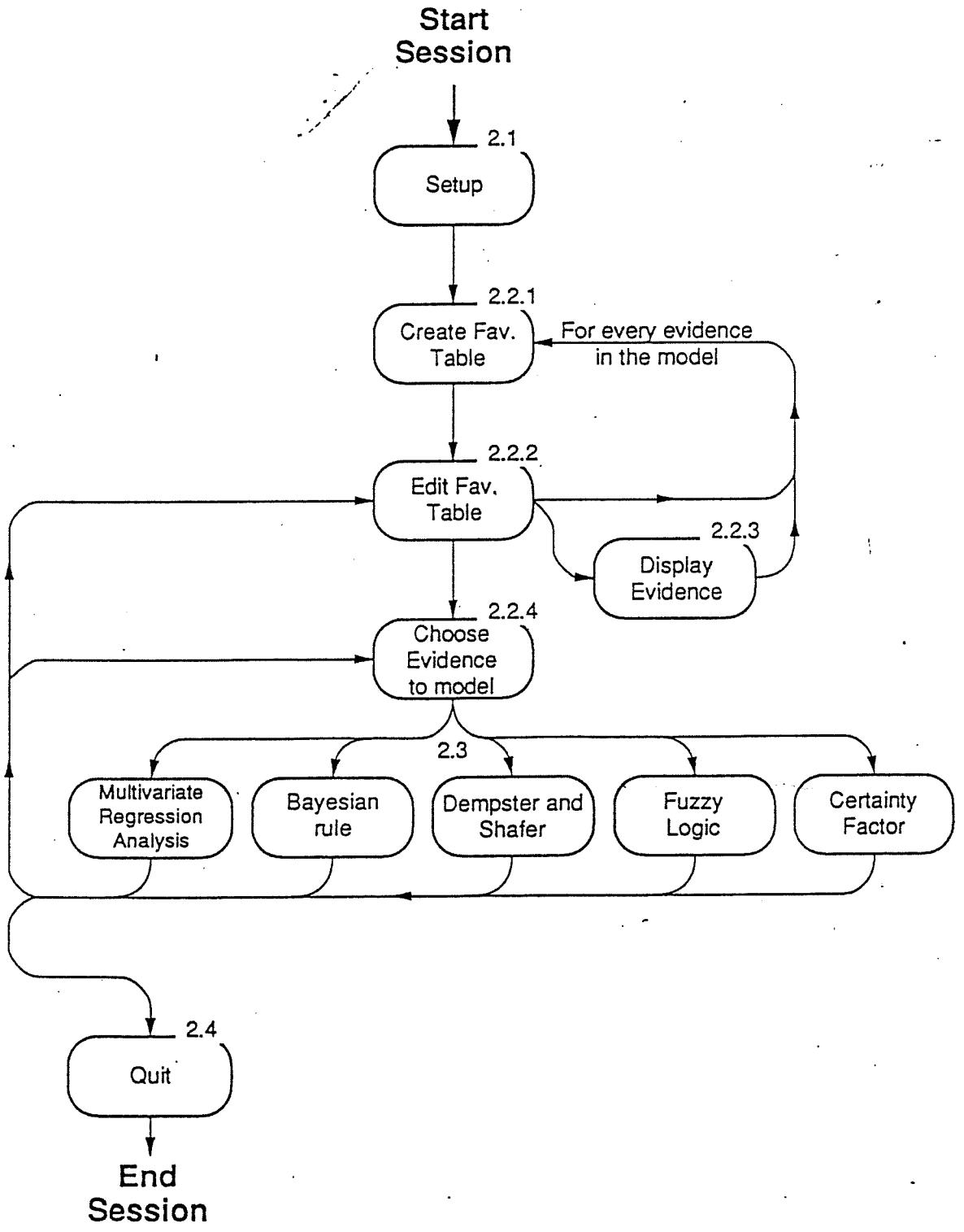


Fig 1. Typical FAVMOD Session

To facilitate the use of this section, every step of the typical session flow chart (Fig. 1) is associated with its proper section. Notice the corresponding section index at the top of each step box such as section 2.1 for 'Setup', or 2.2.1 for 'Create Favorability Table'.
The following details the flow of a typical FAVMOD session:

Each step of a typical session is called from the FAVMOD simple introduction menu (Fig. 2).

Setup	*Evidence*	*Rules*	*Quit*

Fig. 2 - FAVMOD Main Menu with *Evidence* selected

2.1 The '*Setup*' Menu

Setting up the software parameters is essential at the start of every session. The '*Setup*' option sets the following variables:

study area directory: _____
unit pixel size: _____
default SPANS quad level:__
default SPANS window: _
minimum favorability value: _____
maximum favorability value: _____

A <CR> key is necessary to confirm the input of a field and to edit the next one down. The <INS>, <DEL>, and cursors keys are also valid editing keys.

The *study area directory* is the universe recognized by SPANS.

The *unit pixel size* is in km^2. The unit pixel size is used for two integration techniques: the Bayesian and the multivariate regression analysis rule. To demonstrate the utility of the unit pixel, consider this next example.

In Fig. 3, S denotes a study area of size s, and U a subarea of size u. Suppose we have n random points, $p_1, p_2, \ldots, p_n$, enclosed in S and p is the probability that U contains at least one of the n points. It can be shown that:

$$p = \text{Prob } \{ \text{ at least one of the n points is in } u \ \} = 1-(1-w)^n$$
$$\text{where } w = \text{minimum}(u/s\ ,\ 1)$$

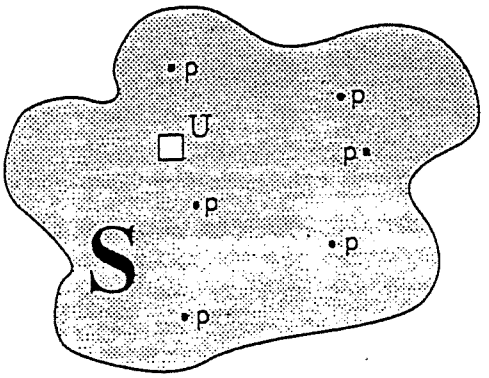


Fig. 3 - Study Area S

The value p is the favorability value that can be edited prior to execution of the Bayesian integration rule. In the case of the multivariate regression analysis, the technique employs np (formula below), the number of points in S needed to achieve the probability p'. The p' is the probability that u contains at least one of the np points.

$$np = \text{integer}(\ 0.5 + \log(1-p') / \log(1-w')\ )$$
$$\text{where } w' = \text{minimum}(u/s\ ,\ 1)$$

The default *quad level* and *window* in the '*Setup*' menu are respectively SPANS spatial dimension for one pixel and the window in which the favorability table should be built. The entry of these values allows FAVMOD to present their contents for editing purposes at numerous points in the execution.

The *minimum* and *maximum favorability values* are essential parameters. The software utilizes these values for display purposes. The GIS SPANS only projects 256 colors onto its screen (8514 Card), thus necessitating a linear stretch process to convert the favorability values (or final model results), which are represented in real numbers, to integers varying between 2 and 254. The color 0 is reserved for un mapped data and 255 for labels. The *minimum* and *maximum* can be either positive or negative real numbers, such as [0, 1.0] for a probability interval or [-1.0, 1.0] for certainty factors.

2.2 The '*Evidence*' Menu

This menu opens up in a vertical menu design to create a favorability table (2.2.1), edit a favorability table (2.2.2), display an evidence (2.2.3), and finally, to specify the evidence in the model (2.2.4).

2.2.1 Create a Favorability Table

Setup	Evidence	Rules	Quit
	Create Favorability Table		
	Edit Favorability Table		
	Display Evidence		
	Choose Evidence to Model		

Fig. 4 - The 'Create Favorability Table' option

The 'Create Favorability Table' option (Fig. 4) builds the favorability table (Fig. 5). The table is designed to assign favorability values to unique mapped polygons of an evidence. The following is an example of a table, in SPANS TBA format:

```
ID FTGEO
TITLE GEOLOGY - Favorability Index Table
MAPID ??
WINDOW 00.0 0.0 0.0
TABTYPE 3
FTYPE free
KEYFIELD 1
KEYBASE 0
NRECORD 4
1    3    4.0   0   class      Associated class
2    60   40.0  0   name       Class name
3    1    10.3  0   area       Total area (square km)
4    3    7.0   0   pnts       Points per class
5    1    12.7  0   prob       Probability value
6    1    12.7  0   bel        Belief function value
7    1    12.7  0   pls        Plausible function value
8    1    12.7  0   fuzzy      Fuzzy logic function value
9    1    12.7  0   certfact   Certainty factors function value
10   3    4.0   0   rcprob     Reclassified probability value
11   3    4.0   0   rcbel      Reclassified belief funct. value
12   3    4.0   0   rcpls      Reclassified plausible funct. value
13   3    4.0   0   rcfuzzy    Reclassified fuzzy logic value
14   3    4.0   0   rccfact    Reclassified certainty fact. value
DATA
1   "Intrusive          "   77.777   1   0.000129   0.0000   1.0000   0.0000   0.0000   3 1 254 1 1
2   "Mixed Volcanic     "   52.360   1   0.000191   0.0000   1.0000   0.0000   0.0000   4 1 254 1 1
3   "Acidic Volcanic    "   214.455  5   0.000233   0.0000   1.0000   0.0000   0.0000   7 1 254 1 1
4   "Sedimentary        "   152.255  1   0.000000   0.0000   1.0000   0.0000   0.0000   1 1 254 1 1
```

Fig. 5 - Example of a Favorability Table

The table contains the favorability value required in each combination rule. Consider the third class, the 'Acidic Volcanic'. The probability , belief , plausible , fuzzy logic, certainty factor values are respectively 0.000129, 0.0000, 1.0000, 0.0000, 0.0000. These values are edited with the 'Edit Favorability Table' menu (see 2.2.2). The area in km^2 and the number of points per class are in column 3 and 4.

Selection of the 'Create Favorability Table' menu will display this next one:

window: __
input map: _____
input point file :_____
unit pixel size: _____
output table: ft_____

The *window* is the area in which the table should be built.

The *input map* is the SPANS map that contains all of the classes. This map can be in another universe as in the case of a control area map found onto a separate directory. In this case, the *study area directory* must be modified in accordance with the '*Setup*' menu (see 2.1). When using a control area map, the resulting favorability table must be copied onto the study area directory using a OS/2 window shell.

Example: copy c:\controla\ftgeo.tba c:\studya\ftgeo.tba

The first two letters of the favorability table 'ft' must be respected when copying tables over directories. Also, the table 'TITLE' field must be modified with a text editor. This field must contain the map it represents followed by a <SPACE> character. Consider this next favorability table:

```
ID ftgeo
TITLE GEOLOGY - Favorability table ...
...
...
```

The 'TITLE' field is followed by 'GEOLOGY'. This indicate that the table 'ftgeo' represents the map called 'GEOLOGY'. When generated in another universe, this map name may well not be the same as in the study area. Modify this map name accordingly; do not forget to leave a <SPACE> after.

The *input point file* is the SPANS points file containing the known occurrences of the proposition, such as know occurrences of minerals' deposits for exploration purposes.

The *unit pixel size* is involved in the calculation of the favorability value for the Bayesian rule (see 2.1).

The *output table* is the generated favorability table (Fig. 5).

After typing all the parameters, FAVMOD unveils a SPANS batch command file to execute in the SPANS window.

2.2.2 Editing Favorability Table

This option (Fig. 6) permits the editing of a favorability table, particularly the favorability values. Alternate ways of editing a favorability table such as using an editor in a OS/2 window shell is strictly prohibited. The 'Edit Favorability Table' option calls the Q. EDITOR. Some useful commands are:

<ALT-F>	Displays the editor main menu.
<CTL-K, X>	Exits and saves the favorability table.
<CTL-K, Q>	Exits and does not save the favorability table.

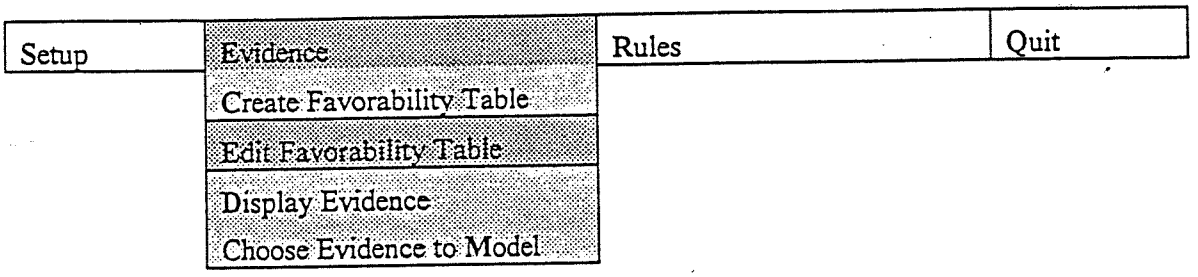


Fig. 6 - The 'Edit Favorability Table' option

The FAVMOD software automatically updates the last 5 columns of the table when it has been modified. These columns represent the reclassified values of the favorability values that are used to display evidence (see 2.2.3). The reclassified values are calculated using an index varying between 1 and 254, derived from a linear stretch of the *minimum* and *maximum favorability value* specified in the 'Setup' menu.

2.2.3 Display Evidence (optional)

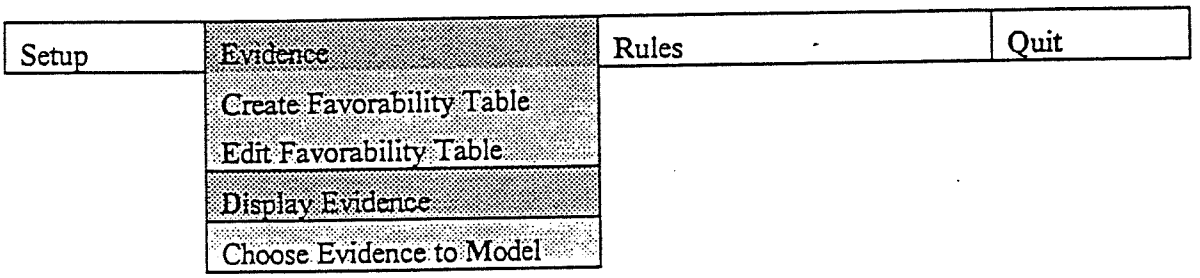


Fig. 7 - The 'Display Evidence' option

This option (Fig. 7) displays an evidence by creating a SPANS map. The procedure is fairly simple. Depending on the integration rule, FAVMOD selects the reclassified favorability value from the favorability table and assigns it to each polygon class of the specified map. After typing all the parameters necessary to execute the display option, FAVMOD unveils a SPANS batch command file to execute in the SPANS window.

The 'Display Evidence' is optional in the process of formulating the proposition.

2.2.4 Choose the Evidence to Model

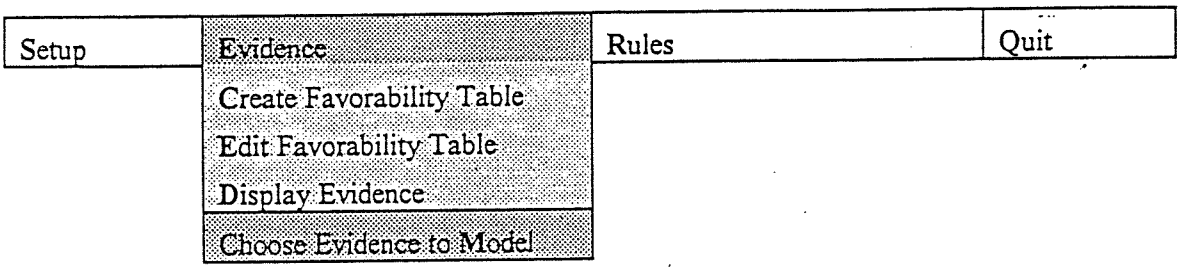


Fig. 8 - The 'Choose the Evidence to Model' option

This option chooses the evidence in the proposition or in the model. Selecting the 'Choose the Evidence to Model' displays a menu similar to the following:

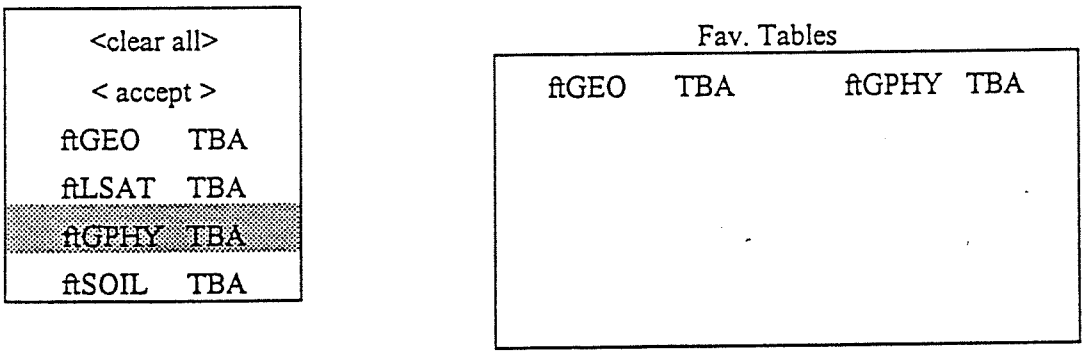


Fig. 9 - Choosing the evidence in the proposition

This sample menu has 4 evidences. On the left the *evidence menu* displays the favorability tables that were previously built. The ⇓ and ⇑ keys move the bar vertically. Pressing <CR> or <SPACE> on a selected favorability table name will copy it onto the right menu 'Fav. Tables'.

This menu shows the evidence or the favorability table that will be used in the model. In Fig. 9 the user has chosen to include 'ftGEO' and 'ftGPHY' in the model.

Note the first two options in the *evidence menu*. The **<clear all>** will clear the list in the 'Fav. Tables' menu. The **<accept> option confirms the selection of the evidence.** After receiving confirmation of the evidences, FAVMOD unveils two SPANS commands to execute in the SPANS window.

The <ESC> key terminates the operation at any time.

2.3 The '*Rules*' Menu

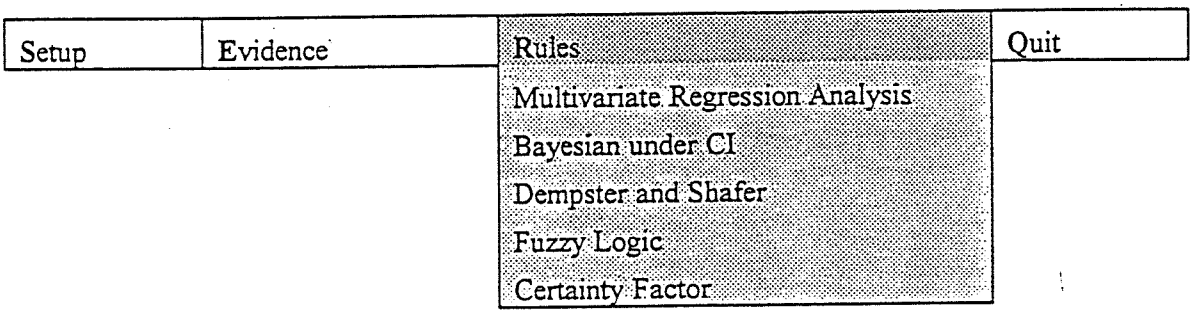


Fig. 10 - The '*Rules*' menu

The '*Rules*' menu offers the choice of selecting one integration rule to generate the final proposition. The following parameters are prompted:

Bayesian output map: _____
quad level: __
window: __

The integration rule *output map* depends on the selected integration rule. It is the final map generated in SPANS MAP format.

The *quad level* is the spatial resolution of the output map.

The *window* is the selected area of interest, making reference to a valid SPANS window ID.

After receiving confirmation of these parameters, FAVMOD unveils a SPANS command batch file to execute in the SPANS window. Resulting from the '*Rules*' operation is a table that can be viewed and manipulated. The names of these tables are as follows:

Generated Table	*Integration Rule*
BAYES.tba	Bayesian
REG.tba	Multivariate Regression Analysis
DEMPSTER.tba	Dempster and Shafer
FUZZY.tba	Fuzzy Logic
CERT.tba	Certainty Factor

2.4 The '*Quit*' Option

This option allows exit from FAVMOD. Remember to select the '*Setup*' option when starting the next session.

2.5 Browsing the Menus

⇑, ⇓, ⇐, ⇒

The software uses cursor keys to enable options and other menus from the main menu. <Right> and <Left> cursor keys are functional in horizontal menus rather than the <Up> and <Down> cursor keys in vertical menus. After positioning the highlighted bar on the appropriate option, the <CR> (carriage return) is necessary to allow the selection.

ABC

An alternative to cursor keys is the hi lighted letter in the main horizontal menu and in every other vertical menu. Pressing the corresponding highlighted letter will automatically enable the appropriate menu option.

<ESC>

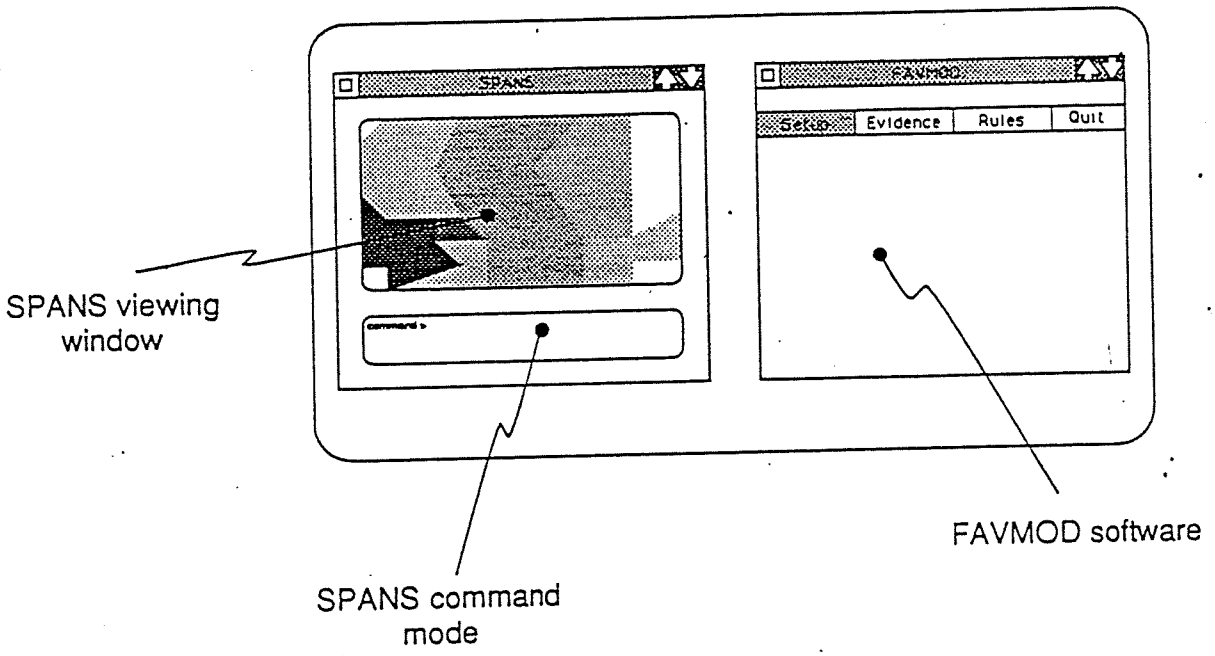
This key will cancel the execution of an option or backtrack to the previous menu.

The Mouse

The mouse in not yet functional, but is essential in switching between SPANS and FAVMOD when commands or batch file need to be run in the GIS.

2.6 Setting the OS/2 Environment

FAVMOD requires that itself and SPANS be running in two windows at the same time. The most apparent and efficient way to set these windows is one beside each other such as indicated in the next diagram:



The window running SPANS should have both the viewing and command windows opened.

3. Software Constraints

The software has the following memory constraints:

Constraint Description	Maximum
matching *.TBA or *.MAP in one universe	100
evidence in a Model	16
unique polygons in overlaying evidence	512
classes to display (with 8514 card of 256 colors)	254

The source code can be modified (file 'int.h') and recompiled to increase or decrease these maximums.

4. Installation Procedure and System Requirements

The steps involved in installing FAVMOD are as follows:

1. Create a directory for storing FAVMOD.
   Example: md c:\favmod

2. Copy the software files from the provided source disk.
   Example: copy a:\favmod.exe c:\favmod
   copy a:\q.exe c:\favmode
   copy a:\favmod.ico c:\favmode

3. Install FAVMOD software in the config.sys PATH variable by using any text editor.

4. Under OS/2 Program Manager, create a icon space pointing to FAVMOD.EXE and assign the icon named c:\favmod\favmod.ico to the software.

In order to run FAVMOD, OS/2 2.0 (or higher) and SPANS 5.2 (or higher) are necessary.

5. Source Code

FAVMOD.H

```
#define     TRUE                    1
#define     FALSE                   0
#define     NOMORE                  256
#define     BOOL                    int
#define     MAP_FILTER              "*.map"
#define     TBA_FILTER              "*.tba"
#define     TBB_FILTER              "*.tbb"
#define     FAVTABLE_FILTER         "ft*.tba"
#define     MAX_FNAMES              100
#define     MAX_EVIDENCES           15
#define     MAX_UNIQUE_POLYGONS     512
#define     MAX_CLASSES             256
#define     MAXSTRREP               80
#define     MAXSTRTBA               500
#define     PRECISION               0.0000001

/*macro that toggle a BOOL variable */
#define     ToggleFlag( flag) (( flag) ? FALSE : TRUE);

/* variable used to manipulate the favorability table */
typedef struct _TABLESstruct {
   int      class;
   char     name[ 40];
   double   area;
   int      pnts;
} TABLES;
extern TABLES table[ MAX_CLASSES];

/* contain the filename of the favorability tables */
extern char   fnames[ MAX_FNAMES][ 13];

/* used in manipulating SPANS unique condition table
struct UNIQUE_TABLE {
   double         area;
   int            class[ MAX_EVIDENCES];
};
extern struct UNIQUE_TABLE  unipoly[ MAX_UNIQUE_POLYGONS];


/* global variables */
extern double               area_per_class[ MAX_EVIDENCES][ MAX_CLASSES];
extern double               area_per_evidence[ MAX_EVIDENCES];
extern double               S_T;
extern double               PD;                                  /* P( D )
*/
extern char                 favtablesfn[ MAX_EVIDENCES][ 9];
extern char                 evidencesfn[ MAX_EVIDENCES][ 9];
extern int                  total_pnts;
extern int                  no_favtables;
extern int                  uni_poly;
extern double               bayest[ MAX_EVIDENCES][ MAX_CLASSES];
extern double               belt[ MAX_EVIDENCES][ MAX_CLASSES];
extern double               plst[ MAX_EVIDENCES][ MAX_CLASSES];
extern double               fuzzyt[ MAX_EVIDENCES][ MAX_CLASSES];
extern double               certfactt[ MAX_EVIDENCES][ MAX_CLASSES];
extern char                 syspath[ 50];
extern char                 unitsize[ 10];
extern char                 quadlevel[ 3];
extern char                 window[ 3];
extern char                 minfav[ 10];
extern char                 maxfav[ 10];
```

```
/* Variables used to track control and screen position.  */
typedef struct _SCREENINFO
{
    int  top;                          /* Row under top form line      */
    int  bot;                          /* Row above bottom form line   */
    int  mid;                          /* Middle line of form          */
    int  help;                         /* Line number of help line     */
    int  mode;                         /* Current screen mode          */
    int  xMax;
    int  yMax;
    BOOL fColor;
} SCREENINFO;
extern SCREENINFO si;
extern void mod();

/* favorability table index, used to display evidences */
extern struct index {
   double    min, max, avrg;
} favindex[ 256];

/* choices of integration rules */
enum rulesmnuCHOICES { REGRESSION, BAYES, DEMPSTER, FUZZY, CERTAINTY};
```

FAVMOD.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

   main()

MODULE INTERNAL FUNCTION(s): All are called from main()

 main
  | MenuHori              : external (menu.c)
  |
  | InitProg
  |
  | Setup
  |
  | BuildEvidence         : external (evidence.c)
  |
  | Rules                 : external (rules.c)
  |
  | WrapupProg
*****************************************************************************/

#include <graph.h>
#include <direct.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <io.h>
#include <errno.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

#include "favmod.h"
#include "favindex.h"
#include "evidence.h"
#include "rules.h"
#include "menu.h"
#include "utils.h"

void main( void );

char          syspath[ 50]       = "c:\\";   /* SPANS universe directory   */
char          unitsize[ 10]      = "1.0";    /* pixel unit size            */
char          quadlevel[ 3]      = "12";     /* SPANS quad level           */
char          window[ 3]         = "01";     /* study or control area win. */
char          minfav[ 10]        = "0.000";  /* minimum favorability value */
char          maxfav[ 10]        = "1.000";  /* maximum favorability value */

short    oldfgd;                             /* old foreground colors      */
long     oldbgd;                             /* old background colors      */
struct   rccoord oldpos;                     /* old cursor position        */
```

```
/*****************************************************************************
name: InitProg()

goal: To initialize some global variables and to save old system environment.

input: none
returns: nothing

Global variables changed:
        oldfgd, oldbgd, oldpos  : screen colors and position
        mnuAtrib                : visual attributes of the program's menus
        no_evi, uni_poly        : # of evidences in the model and # of
                                  unique polygons in the output model map
*****************************************************************************/
void InitProg()
{
   /* saving screen colors */
   oldfgd = _gettextcolor();
   oldbgd = _getbkcolor();
   oldpos = _gettextposition();

   /* modifying screen colors */
   _settextcolor( _TWHITE);
   _setbkcolor( _TBLACK);
   _clearscreen( _GCLEARSCREEN );
   _displaycursor( _GCURSOROFF );

   /* modifying menu colors */
   mnuAtrib.fgBorder       = _TWHITE;
   mnuAtrib.bgBorder       = _TBLACK;
   mnuAtrib.fgNormal       = _TWHITE;
   mnuAtrib.bgNormal       = _TBLUE;
   mnuAtrib.fgSelect       = _TBLUE;
   mnuAtrib.bgSelect       = _TWHITE;
   mnuAtrib.fgNormHilite   = _TBRIGHTWHITE;
   mnuAtrib.bgNormHilite   = _TBLUE;
   mnuAtrib.fgSelHilite    = _TBRIGHTWHITE;
   mnuAtrib.bgSelHilite    = _TWHITE;
   mnuAtrib.fCentered      = FALSE;
   mnuAtrib.chNW           = '+';
   mnuAtrib.chNE           = '+';
   mnuAtrib.chSE           = '+';
   mnuAtrib.chSW           = '+';
   mnuAtrib.chNS           = ' ';
   mnuAtrib.chEW           = ' ';

   no_favtables = 0; uni_poly = 0;

} /* InitProg() */

/*****************************************************************************
name: WrapupProg()

goal: Restore the PC's environment.

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void WrapupProg()
{
   /* restore colors and cursor position */
   _settextcolor( oldfgd );
   _setbkcolor( oldbgd );
   _clearscreen( _GCLEARSCREEN );
   _settextposition( oldpos.row, oldpos.col );
} /* WrapupProg() */
```

```
/*****************************************************************************
name: MakeFavIndexTable()

goal: To make the favorability index table.  The table usually varies between
      0 and 1 and always consist of 254 classes or steps.  These classes are
      assigned to each polygon in SPANS mainly because of the GIS limitation
      in manipulating only integer and not real numbers.

input: none
returns: nothing

Global variables changed:
      favindex : the favorability index used in the program.  The index
              has 254 classes that vary between the values specified by
              the variables MINFAV and MAXFAV.
*****************************************************************************/
void MakeFavIndexTable()
{
int      i;          /* utility counter                                    */
double   step;       /* the increment between each class                   */

   /* first palette index is reserve to unmapped area */
   /* last palette index is reserve for text color    */
   favindex[ 0].min = 0.0; favindex[ 0].max = 0.0; favindex[ 0].avrg = 0.0;
   favindex[ 255].min = 0; favindex[ 255].max = 0; favindex[ 255].avrg = 0;

   /* construction the full index */
   step = fabs( (atof( maxfav) - atof( minfav))/ 254);
   favindex[ 1].min  = atof( minfav);
   favindex[ 1].max  = atof( minfav);
   favindex[ 1].avrg = atof( minfav);

   for ( i=2; i < 254; ++i) {
      favindex[ i].min  = favindex[ i-1].max + PRECISION;
      favindex[ i].max  = favindex[ i].min + step;
      favindex[ i].avrg = ( favindex[ i].min + favindex[ i].min) / 2;
   }
   favindex[ 254].min  = atof( maxfav);
   favindex[ 254].max  = DBL_MAX;
   favindex[ 254].avrg = atof( maxfav);

} /* MakeFavIndexTable() */
```

```
/*************************************************************************
name: Setup()

goal: To ask the user for some program parameters such as:
        - the directory containing the study area
        - the size of the unit pixel
        - default SPANS quad level
        - default SPANS window
        - the favorability table minimum value
        - the favorability table maximum value

input: none
returns: nothing

Global variables changed:
   syspath   : the directory containing the study area
   unitsize  : the size of the unit pixel
   quadlevel : default SPANS quad level
   window    : default SPANS window
   minfav    : the favorability table minimum value
   maxfav    : the favorability table maximum value
*************************************************************************/
void Setup()
{
int      sel;
enum parmsChoices { SYSPATH, UNITPIXEL, QUADLEVEL, WINDOW,
                    MINFAV, MAXFAV, CHANGE_SYS_PARMS, END};

   /* ask for the unit size, the system directory path, and                */
   /*   classification scheme of the favorability index, quad tree level,  */
   /*   and default window                                                  */

   _settextposition( 5, 2); _outtext( "system directory path: ");
   _settextposition( 7, 2); _outtext( "unit pixel size (km2): "); OutBgText(
unitsize);
   _settextposition( 9, 2); _outtext( "quad tree level: "); OutBgText(
quadlevel);
   _settextposition(11, 2); _outtext( "window: "); OutBgText( window);
   _settextposition(13, 2);  _outtext( "minimum favorability value: ");
OutBgText( minfav);
   _settextposition(15, 2);  _outtext( "maximum favorability value: ");
OutBgText( maxfav);


   /* loop until the last entry has been input */
   sel = SYSPATH;
   while( sel != END && sel != ESC)
      switch( sel) {

        case SYSPATH:
           do {
              sel = EditStr( 5, 2+23, 24, validpath, syspath);
              if ( !EXIST( syspath)) {
                ProgramError( "Invalid path, please enter again.");
                _settextposition(13, 2);
                _settextposition(13, 2);
                _outtext( "minimum favorability value: "); OutBgText( minfav);
              }
           } while ( !EXIST( syspath) && sel != ESC);
           sel = (sel == ESC) ? ESC : UNITPIXEL;
           break;

        case UNITPIXEL:
           sel = EditStr( 7, 23+2, 9, numbersonly, unitsize);
           _settextposition( 7, 23+2); OutBgText( unitsize);
           sel = (sel == ESC) ? ESC : QUADLEVEL;
           break;
```

```
            case QUADLEVEL:
               sel = EditStr( 9, 17+2, 2, numbersonly, quadlevel);
               _settextposition( 9, 17+2); OutBgText( quadlevel);
               sel = (sel == ESC) ? ESC : WINDOW;
               break;

            case WINDOW:
               sel = EditStr( 11, 2+8, 2, validfilename, window);
               _settextposition(11, 8+2); OutBgText( window);
               sel = (sel == ESC) ? ESC : MINFAV;
               break;

            case MINFAV:
               chdir( syspath);
               sel = EditStr( 13, 2+28, 9, numbersonly, minfav);
               _settextposition( 13, 2+28); OutBgText( minfav);
               sel = (sel == ESC) ? ESC : MAXFAV;
               break;

            case MAXFAV:
               chdir( syspath);
               sel = EditStr( 15, 2+28, 9, numbersonly, maxfav);
               _settextposition( 15, 2+28); OutBgText( maxfav);
               sel = (sel == ESC) ? ESC : CHANGE_SYS_PARMS;
               break;

            case CHANGE_SYS_PARMS:
               chdir( syspath);
               MakeFavIndexTable();
               sel = END;
               break;
         }

      ClearBox( 3, 1, 21, 77);

   } /* Setup() */
```

```
/*****************************************************************************
name: main()

goal: Start the main program called 'FAVMOD.exe'.  The main routine
      offers the user the opening menu.

input: none
returns: nothing
Global variables changed:
*****************************************************************************/
void main()
{
int sel = 0;                    /* menu's selection */
ITEM mainmnu[] = {              /* main menu of the program */
   { 2, "  Setup"},
   { 0, "Evidence"},
   { 2, "  Rules"},
   { 2, "  Quit"},
   { 0 , ""}
}; enum mainmnuCHOICES { SETUP, EVIDENCES, RULES, QUIT};

   InitProg();                  /* initialize the program's vars and parms */

   /* start the program main loop until the user quit the program */
   do {
      sel = MenuHori( 2, 2, mainmnu, sel);
      switch( sel) {
        case SETUP:             /* Sets up the program's parameters          */
           Setup();
           break;
        case EVIDENCES:         /* create and edit fav. tables               */
           BuildEvidence();
           break;
        case RULES:             /* ask the use to select the integration rule*/
           Rules();
           break;
      }
   } while ( sel != QUIT);

   WrapupProg();                /* restore old PC's environment              */

} /* main() */
```

EVIDENCE.H

```
void BuildEvidence( void);
```

EVIDENCE.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

   BuildEvidence()

MODULE INTERNAL FUNCTION(s): All are called from BuildEvidence()

   BuildEvidence                    : called from favmod.c
   |
   | CreateFavTable
   | | MakeFavTBA
   | | | GetAreaClassPoint
   | | | GenerateTBAFile
   | | | | CalculateP_D
   | | | | ReclassifiedModelValue
   |
   | EditFav
   | | ReBuildTBA
   | | | ReclassifiedModelValue
   |
   | BuildEvi
   | | GenerateEvidence
   |
   | ChooseEvidences                : external (model.c)
*****************************************************************************/
#include <string.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <conio.h>
#include <process.h>
#include <io.h>
#include <errno.h>
#include <math.h>
#include <sys\types.h>
#include <sys\stat.h>

#include "favmod.h"
#include "menu.h"
#include "utils.h"

ITEM rulesmnu[] = {                          /* integration rules available */
   {13, "Multivariate Regression Analysis"},
   { 7, "       Bayesian under CI"},
   {10, "          Dempster's"},
   {10, "          Fuzzy Logic"},
   { 7, "       Certainty Factor"},
   { 0 , ""}
};
void ChooseEvidences( void);  /* from the 'model.c' module */

/* this variable is a temporary table used to store the class,            */
/* points/class, and the area for each evidence                           */
TABLES table[ 0] = { ( int)0, " ", ( float)0, ( int)0};

/* another temporary var used to manipulate the names of the evidences    */
char   fnames[ MAX_FNAMES][ 13];
```

```
/*****************************************************************************
name: GetAreaClassPoint( char *repf, char *tbaf)

goal: To get the number of classes contained in a map with each associated
      area and points/ class.  This is done by analyzing a REP and a TBA
      SPANS file.

input: repf : the report file name used to determine the classes and areas
       tbaf : the TBA file name used the determine the number of points in each
              classes

returns: nothing

Global variables changed: table : the classes, areas, and points/classes
*****************************************************************************/
void GetAreaClassPoint( char *repf, char *tbaf)
{
FILE  *inrepf, *intbaf;        /* REP and TBA file pointer                    */
char  tmpstr[ MAXSTRTBA];      /* temporary string                            */
char  startinfo[ 80];          /* flag string indicating                      */
int   startread=FALSE, i=0;    /* flag indicating when to read some info      */
int   countclass[ 256];        /* indicates the number of classes             */


   /* Initialize the string flaging the start of the data in the REP file */
   strcpy( startinfo, " ---------");

   /* Opens the *.rep file containing the classes and area. */
   /* Reads in each class with its associated area.         */

   /* Opens up the files */
   if ( (inrepf = fopen( repf, "rt")) == NULL)
      ProgramError( strcat( "Cannot open file:", repf));

   else {
      /* Reads in the report file header. */
      do {
        if ( fgets( tmpstr, MAXSTRREP - 1, inrepf) == NULL) {
           ProgramError( strcat( "Cannot read file:", repf));
           break;
        }

      } while ( strncmp( tmpstr, startinfo, 10) != (int )NULL);

      /* Reads in the classes and associated areas. */
      if ( fgets( tmpstr, MAXSTRREP - 1, inrepf) == NULL)
        ProgramError( strcat( "Cannot read file:", repf));
      while ( strncmp( tmpstr, startinfo, 10) != (int )NULL) {
        /* get the class and the class name */
        table[ i].name[ 0] = 0;
        sscanf( tmpstr, "%i %[^0-9]", &table[i].class, table[ i].name);

        /* get the area */
        table[ i].area = StrGetLastFloat( tmpstr, LASTNUM, 0);

        ++i;
        if ( fgets( tmpstr, MAXSTRREP - 1, inrepf) == NULL)
           ProgramError( strcat( "Cannot read file:", repf));
      }
      fclose( inrepf);
      table[ i].class = NOMORE;
```

```
        /* reads in the TBA file to determine the points/class if the filename*/
        /* exist                                                              */
        /* Reads in the number of points/area with the tba file. */
        if ( (intbaf = fopen( tbaf, "rt")) == NULL)
          ProgramError( strcat( "Cannot open file:", tbaf));
        else {
          for (i=0; i<256; countclass[ i]=0, ++i);
          while ( TRUE) {

             if ( fgets( tmpstr, MAXSTRTBA - 1, inrepf) == NULL)
                if( feof( intbaf ) )
                  break;
                else
                  ProgramError( strcat( "Cannot read file:", tbaf));
             ++countclass[ ( int)StrGetLastFloat( tmpstr, LASTNUM, 0)];
          }
          fclose( intbaf);
          for (i=0; table[ i].class != NOMORE; ++i)
             table[ i].pnts = countclass[ table[ i].class];
        }
    }

} /* GetAreaClassPoint() */


/*****************************************************************************
name: GenerateTBAFile( char *tbafn, char *mapfn)

goal: To generate or write out a TBA file.  The TBA file is used a index
      to create the evidence maps.

input: tbafn    : the name of the TBA file to be created
       mapfn    : the evidence name used to create the table

returns: nothing
Global variables changed: none
*****************************************************************************/
void GenerateTBAFile( char *tbafn, char *mapfn)
{
char     tmpfn[ 13];                    /* temporary string to edit a fn    */
int      i;                             /* utility counter                  */
FILE     *outtba;                       /* output TBA file pointer          */
char     dumstr[ 20];                   /* utility string                   */
double   prob;                          /* used in the computation of index */
double   bel;                           /* Dempster-Shafer Belief function  */
double   pls;                           /* Dempster-Shafer Pluasible funct. */
double   fuzzy;                         /* fuzzy logic set function         */
double   certfact;                      /* Certainty factor function value  */

char     TBAHeader[ 24][ 65] = {        /* SPANS TBA file header format     */
           "ID x",
           "TITLE xxx - Favorability Index Table",
           "MAPID ??",
           "WINDOW 00 0 0 0 0",
           "TABTYPE 3",
           "FTYPE free",
           "KEYFIELD 1",
           "KEYBASE 0",
           "NRECORD x",
           " 1  3   4.0 0   class    Associated class",
           " 2 60  40.0 0   name     Class name",
           " 3  1  10.3 0   area     Total area (square km)",
           " 4  3   7.0 0   pnts     Points per class",
           " 5  1  12.7 0   prob     Probability value",
           " 6  1  12.7 0   bel      Belief function value",
           " 7  1  12.7 0   pls      Plausible function value",
           " 8  1  12.7 0   fuzzy    Fuzzy logic function value",
```

```
            " 9  1  12.7 0   certfact Certainty factors function value",
            "10  3   4.0 0   rcprob   Reclassified probability value",
            "11  3   4.0 0   rcbel    Reclassified belief funct. value",
            "12  3   4.0 0   rcpls    Reclassified plausible funct. value",
            "13  3   4.0 0   rcfuzzy  Reclassified fuzzy logic value",
            "14  3   4.0 0   rccfact  Reclassified certainty fact. value",
            "DATA"};

    /* fills in the header information */

    strcpy( dumstr, "ID ");
    strcat( dumstr, tbafn);
    strcpy( TBAHeader[ 0], dumstr);

    mapfn [ strchr( mapfn, ' ') - mapfn] = 0;
    sprintf( TBAHeader[ 1], "TITLE %s - Favorability Index Table", mapfn);

    /* count number of classes and store it */
    i=0;
    while ( table[ i].class != NOMORE) ++i;
    itoa( i, dumstr, 10);
    strcpy( &TBAHeader[ 8][8], dumstr);

    /* opens up the tba file for writing */
    strcpy( tmpfn, tbafn);
    strcat( tmpfn, ".tba");
    outtba = fopen( tmpfn, "w+");

    /* writes out the TBA header */
    for (i=0; i < 24; ++i)
       fprintf( outtba, "%s\n", TBAHeader[ i]);

    /* writes out the TBA data */
    for (i=0; table[ i].class != NOMORE; ++i) {

       prob = CalculateP_D( (double )table[ i].area, table[ i].pnts);
       /* certfact = GetCertFact(); ... */
       /* fuzzy = GetFuzzyLogic(); ... with GetPls() and GetBel()*/
       bel      = atof( minfav);
       pls      = atof( maxfav);
       certfact = 0.0;
       fuzzy    = 0.0;

       /* writes out the favorability value along with their reclassified   */
       /*   values that ranges between FAVMIN and FAVMAX                    */
       fprintf( outtba, "%4i '%40s' %12.7lf %7i %12.7lf %12.7lf %12.7lf"
                        " %12.7lf %12.7lf %4i %4i %4i %4i %4i\n",
                        table[ i].class, table[ i].name, table[ i].area,
                        table[ i].pnts, prob, bel, pls,
                        fuzzy, certfact,
                        ReclassifiedModelValue( prob),
                        ReclassifiedModelValue( bel),
                        ReclassifiedModelValue( pls),
                        ReclassifiedModelValue( fuzzy),
                        ReclassifiedModelValue( certfact));
    }
    fclose( outtba);

} /* GenerateTBAFile() */
```

```
/*************************************************************************
name: MakeFavTBA( char *mapfn, char *pntsfn, char *outtbafn)

goal: The function is designed to create a SPANS command file to create a
      probability index.  At the termination of the command file the function
      calls another function to write out the results in a TBA file format.

input: mapfn    : the name of the map used to generate the index
       pnts     : the point file involved in the calculation of the index
       outtbafn : the name of the output index table

returns: nothing
Global variables changed: none
*************************************************************************/
void MakeFavTBA( char *mapfn, char *pntsfn, char *outtbafn)
{
FILE     *execfile;     /* SPANS command file pointer                    */
long     i;             /* utility counter                               */
int      c='';          /* keyboard input                                */

/* SPANS command file format necessary to build a   */
/* probability index.                               */
/*     Position             Argument                */
/*      0,10        map with different classes      */
/*      0,23             map window                 */
/*      0,29          output file report            */
/*      1,12              point file                */
/*      1,24     output table with appended classes */
/*      1,36        map with different classes      */
/*      1,48             map window                 */
/*      2,10          exported output table         */
char     probindex[ 3][ 55] = {
           "areal  m =          w =   r =       s =n p =n",
           "reclpnt  p =         o =         m =         w =  ",
           "attexp a =         f =0 d =\" \" h =n" };

   /* modifying SPANS command file to the user specifications */
   memcpy( probindex[ 0] + 10, mapfn, strchr( mapfn, ' ') - mapfn);
   memcpy( probindex[ 0] + 23, window, strlen( window));
   memcpy( probindex[ 0] + 29, "temp", 4);
   memcpy( probindex[ 1] + 12, pntsfn, strchr( pntsfn, ' ') - pntsfn);
   memcpy( probindex[ 1] + 24, "temp", 4);
   memcpy( probindex[ 1] + 36, mapfn, strchr( mapfn, ' ') - mapfn);
   memcpy( probindex[ 1] + 48, window, strlen( window));
   memcpy( probindex[ 2] + 10, "temp", 4);

   /* writing out the batch file */
   execfile = fopen( "mkfav.cmd", "w+");
   fprintf( execfile, "%s\n%s\n%s\n", probindex[ 0], probindex[ 1],
                                      probindex[ 2]);
   fclose( execfile);

   /* indicating the user to execute the batch file */
   _settextposition( 20, 2); _outtext("SPANS: exec f=mkfav");
   remove( "temp.tba");

   /* loop until the table is completely created */
   while ( TRUE) {

      for (i=0; i<70000; ++i);
      _settextposition( 21, 2); _outtext("Waiting for SPANS ... \\");

      /* if the rename succeed, the file is created and free in access */
      if (  CompletedJob( "temp.tba")) break;

      for (i=0; i<70000; ++i);
      _settextposition( 21, 2); _outtext("Waiting for SPANS ... -");
```

```
      if ( kbhit()) {
        c = getch();
        if ( c == ESC) break;
      }

      for (i=0; i<70000; ++i);
      _settextposition( 21, 2); _outtext("Waiting for SPANS ... /");
   }

   if ( c != ESC) {
      /* gets the different classes, their areas (in km2), and the number  */
      /*    of points in each class                                        */
      GetAreaClassPoint("temp.rep", "temp.tba");
      remove( "temp.tba"); remove( "temp.rep");

      /* Writes the TBA file */
      GenerateTBAFile( outtbafn, mapfn);

      _settextposition( 23, 2); _outtext("The table is now built.");
      _settextposition( 24, 2); _outtext("Press any key to continue...");
      while ( !kbhit());
      getch();

   }
   ClearBox( 10, 2, 15, 35);

} /* MakeFavTBA() */

/*****************************************************************************
name: CreateFavTable()

goal: To get the parameters from the user necessary to make a favorability
      table and to call another function that will create this table.

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void CreateFavTable()
{
char     outtbafn[ 9]="";     /* favorability table filename                */
char     mapfn[ 13];          /* map filename used to make the fav. table   */
char     pntsfn[ 13];         /* points file used to make the fav. table    */
int      nofiles = 0;         /* # of matching *.MAP and *.TBB files        */
int      sel;                 /* menu selection                             */

enum parmsChoices { WINDOW, INMAP, INPNTS, UNITSIZE, OUTTABLE, EXECUTE, END};

   /* display the parameters necessary to generate a favorability table    */
   _settextposition( 10, 2); _outtext("window:");
   _settextposition( 12, 2); _outtext("input map:");
   _settextposition( 14, 2); _outtext("input point file:");
   _settextposition( 16, 2); _outtext("unit pixel size: ");
   _settextposition( 18, 2); _outtext("output table:");

   /* gets each one of the parameters */
   sel = WINDOW;
   while( sel != END && sel != ESC)
      switch( sel) {

        case WINDOW:
           if ( (nofiles=GetFileNames( MAP_FILTER))!= 0) {
              sel = EditStr( 10, 10, 2, validfilename, window);
              sel = (sel == ESC) ? ESC : INMAP;
           }
```

```
            else {
               ProgramError( "No map(s) exist on your directory (*.MAP).");
               sel = END;
            }
            break;

         case INMAP:
            sel = SelectOneFileMenu( 12, 13, nofiles);
            if ( sel != ESC) {
               strcpy( mapfn, fnames[ sel]);
               ClearBox( 12, 13, 13, 15);
               _settextposition( 12, 13); OutBgText( mapfn);
               _settextposition( 14, 2); _outtext("input point file:");
               _settextposition( 16, 2); _outtext("unit pixel size: ");
               _settextposition( 18, 2); _outtext("output table:");
               sel = INPNTS;
            }
            break;

         case INPNTS:
            sel = ESC;
            if ( (nofiles = GetFileNames( TBB_FILTER)) != 0) {
               if ( ( sel=SelectOneFileMenu( 14, 20, nofiles)) != ESC) {
                  strcpy( pntsfn, fnames[ sel]);
                  ClearBox( 14, 20, 11, 15);
                  _settextposition( 14, 20); OutBgText( pntsfn);
                  _settextposition( 16, 2); _outtext("unit pixel size: ");
                  _settextposition( 18, 2); _outtext("output table:");
                  sel = UNITSIZE;
               }
            }
            else {
               ProgramError( "No point table exist on your directory (*.TBB).");
               sel = END;
            }
            break;

         case UNITSIZE:
            sel = EditStr( 16, 2+17, 9, numbersonly, unitsize);
            _settextposition( 18, 2); _outtext("output table:");
            sel = (sel == ESC) ? ESC : OUTTABLE;
            break;

         case OUTTABLE:
            /* every output fav. table starts with the prefix 'FT' */
            strcpy( outtbafn, "ft");
            _settextposition( 18, 16); OutBgText( outtbafn);
            sel = EditStr( 18, 16+2, 6, validfilename, outtbafn+2);
            _settextposition( 18, 16); OutBgText( outtbafn);
            sel = (sel == ESC) ? ESC : EXECUTE;
            break;

         case EXECUTE:
            MakeFavTBA( mapfn, pntsfn, outtbafn);
            sel = END;
            break;

      }

   ClearBox( 10, 2, 15, 70);

} /* CreateFavTable() */
```

```
/*****************************************************************************
name: GenerateEvidence( char *indextbl, char *inmap,
                        int rule, char *evi1, char *evi2)

goal: To generate evidence maps.  The function generate a SPANS command
      file designed to build one or two evidence map depending on the choice
      of integration rule.

input: indextble : favorability index table (*.TBA)
       inmap     : input map to be converted with the favorability index
       rule      : the integration rule (choice of REGRESSION, BAYES,
                 DEMPSTER, FUZZY, or CERTAINTY)
       evi2      : the name of the first evidence map
       evi1      : the name of the second evidence map

returns: nothing
Global variables changed: none
*****************************************************************************/
void GenerateEvidence( char *indextbl, char *inmap,
                       int rule, char *evi1, char *evi2)
{
FILE     *execfile;     /* SPANS command file used to create evidence maps */
long     i;             /* utility counter                                 */
int      c='□;          /* keyboard input                                  */
char     mapfile[ 12];  /* evidence map filename                           */
char     favpos[ 3];    /* alphanumeric position of the int.rule in table  */
char     plspos[ 3];    /* alphanumeric position of the plausible funct.   */

/* SPANS command file format to build a evidence map  */
/*     Position             Argument                  */
/*      0,10         favorability index table         */
/*      1,11              input map file              */
/*      1,23              output map file             */
/*      1,35                  window                  */
/*      1,57          favorability index table        */
/*      1,79          column # of favorability value  */
/*      1,85                quad level                */
char  build_evi[ 2][ 90] = {
          "attimp f =        ",
          "ovr_tbb m =         o =         w =   c =0  b =y s =y f =         j =1 k =0 l =
q =  "};


   /* adapting the SPANS batch file to the needs */
   memcpy( build_evi[ 0] + 10, indextbl, strchr( indextbl, ' ') - indextbl);
   memcpy( build_evi[ 1] + 11, inmap, strchr( inmap, ' ') - inmap);
   memcpy( build_evi[ 1] + 23, evi1, strchr( evi1, ' ')- evi1);
   memcpy( build_evi[ 1] + 35, window, strlen( window));
   memcpy( build_evi[ 1] + 57, indextbl, strchr( indextbl, ' ') - indextbl);
   memcpy( build_evi[ 1] + 85, quadlevel, strlen( quadlevel));

   /* specifying the column number of the favorability value in the table  */
   switch( rule) {
      case REGRESSION:
        strcpy( favpos, "1 ");
        break;
      case BAYES:
        strcpy( favpos, "10");
        break;
      case DEMPSTER:
        strcpy( favpos, "11");
        strcpy( plspos, "12");
        break;
      case FUZZY:
        strcpy( favpos, "13");
        break;
```

```
      case CERTAINTY:
        strcpy( favpos, "14");
        break;
    }
    memcpy( build_evi[ 1] + 79, favpos, 2);

    /* write the batch file into "MKEVI.CMD" */
    execfile = fopen( "mkevi.cmd", "w+");
    fprintf( execfile, "%s\n%s\n", build_evi[ 0], build_evi[ 1]);
    if ( rule==DEMPSTER) {
       memcpy( build_evi[ 1] + 79, plspos, 2);
       memcpy( build_evi[ 1] + 23, evi2, strchr( evi2, ' ')- evi2);
       fprintf( execfile, "%s\n", build_evi[ 1]);
    }
    fclose( execfile);

    /* waiting for the creation of the evidence map ... */
    _settextposition( 20, 2); _outtext("SPANS: exec f=mkevi");
    strcpy( mapfile, evi1);
    mapfile[ strchr( evi1, ' ') - evi1] = 0;
    strcat( mapfile, ".map");

    remove( mapfile);

    while ( TRUE) {

       for (i=0; i<70000; ++i);
       _settextposition( 21, 2); _outtext("Waiting for SPANS ... \\");

       /* terminate the loop if the map if created */
       if ( CompletedJob( mapfile)) break;
       for (i=0; i<70000; ++i);
       _settextposition( 21, 2); _outtext("Waiting for SPANS ... -");

       if ( kbhit()) {
         c = getch();
         if ( c == ESC) break;
       }

       for (i=0; i<70000; ++i);
       _settextposition( 21, 2); _outtext("Waiting for SPANS ... /");
    }

    if ( c != ESC) {
       _settextposition( 23, 2); _outtext("The evidence(s) now exist.");
       _settextposition( 24, 2); _outtext("Press any key to continue...");
       while ( !kbhit());
       getch();

    }

    /* clear portion of the screen */
    ClearBox( 10, 5, 14, 50);

} /* GenerateEvidence() */
```

```
/*************************************************************************
name: BuildEvi( int rule)

goal: To get the parameters necessary to make a evidence map and to call
      functions to make this map.

input: rule : one of the 5 integration rules (REGRESSION, BAYES, DEMPSTER,
             FUZZY, or CERTAINTY")

returns: nothing
Global variables changed: none
*************************************************************************/
void BuildEvi( int rule)
{
char     indextbl[ 13];           /* the favorability index used in the em */
char     evidence[ 13]="evi";     /* evidence map filename                 */
char     evibel[ 13]  ="bel";     /* evidence belief map filename          */
char     evipls[ 13]  ="pls";     /* evidence plausible map filename       */
char     inmap[ 13];              /* the input map to be converted filename*/
char     eviident='□';           /* indicates the type of integration rule*/
int      nofiles = 0;             /* number of files filtered              */
int      sel;                     /* index specifying the selected filename*/

enum parmsChoices { WINDOW, QUADLEVEL, INTABLE, INMAP, EXECUTE, END};

   /* displaying the user input variables */
   _settextposition( 13, 2); _outtext("window:");
   _settextposition( 14, 2); _outtext("quad level:");
   _settextposition( 15, 2); _outtext("favorability index table:");
   _settextposition( 16, 2); _outtext("input map:");
   if ( rule == DEMPSTER) {
      _settextposition( 17, 2); _outtext( "output BELIEF.   evidence map: \0");
      _settextposition( 18, 2); _outtext( "output PLAUSIBLE evidence map: \0");
   }
   else {
      _settextposition( 17, 2); _outtext("output evidence map: ");
   }

   /* gets the parameters one by one */
   sel = WINDOW;
   while( sel != END && sel != ESC)
      switch( sel) {

        case WINDOW:
           if ( (nofiles=GetFileNames( TBA_FILTER))!= 0) {
              sel = EditStr( 13, 10, 2, validfilename, window);
              sel = (sel == ESC) ? ESC : QUADLEVEL;
           }
           else {
              ProgramError( "No table(s) exist on your directory (*.TBA).");
              sel = END;
           }
           break;

        case QUADLEVEL:
           sel = EditStr( 14, 14, 2, numbersonly, quadlevel);
           _settextposition( 14, 14); OutBgText( quadlevel);
           sel = (sel == ESC) ? ESC : INTABLE;
           break;
```

```
case INTABLE:
   sel = SelectOneFileMenu( 15, 28, nofiles);
   if ( sel != ESC) {
      strcpy( indextbl, fnames[ sel]);
      /* Adds a letter(R,A,B P,F,or C) to the input favorability  */
      /* index table name.  That new name is the evidence map     */
      /* name.                                                    */
      if ( rule == DEMPSTER) {
        strncpy( evibel, fnames[ sel], 8);
        evibel[ strchr(fnames[ sel], ' ')-fnames[sel]] = 'B';
        evibel[ 8] =0; strcat( evibel, " MAP");
        strncpy( evipls, fnames[ sel], 8);
        evipls[ strchr(fnames[ sel], ' ')-fnames[sel]] = 'P';
        evipls[ 8] = 0; strcat( evipls, " MAP");
      }
      else {
        switch( rule) {
           case REGRESSION:
             eviident = 'R';
             break;
           case BAYES:
             eviident = 'A';
             break;
           case FUZZY:
             eviident = 'F';
             break;
           case CERTAINTY:
             eviident = 'C';
             break;
        }

        strncpy( evidence, fnames[ sel], 8);
        evidence[ strchr(fnames[ sel], ' ')-fnames[sel]] = eviident;
        evidence[ 8] = 0; strcat( evidence, " MAP");
      }

      ClearBox( 15, 28, 11, 15);
      _settextposition( 15, 28); OutBgText( indextbl);
      _settextposition( 16, 2 ); _outtext("input map:");
      if ( rule == DEMPSTER) {
        _settextposition( 17, 2); _outtext("output BELIEF    evidence map: ");
        OutBgText( evibel);
        _settextposition( 18, 2); _outtext("output PLAUSIBLE evidence map: ");
        OutBgText( evipls);
      }
      else {
        _settextposition( 17, 2); _outtext("output evidence map: ");
        OutBgText( evidence);
      }
      sel = INMAP;
   }
   break;

case INMAP:
   sel = ESC;
   if ( (nofiles = GetFileNames( MAP_FILTER)) != 0)
     if ( ( sel=SelectOneFileMenu( 16, 13, nofiles)) != ESC) {
       strcpy( inmap, fnames[ sel]);
       ClearBox( 16, 13, 11, 15);
       _settextposition( 16, 13); OutBgText( inmap);
       if ( rule == DEMPSTER) {
          _settextposition( 17, 2); _outtext("output BELIEF    evidence map: ");
          OutBgText( evibel);
          _settextposition( 18, 2); _outtext("output PLAUSIBLE evidence map: ");
          OutBgText( evipls);
       }
```

```
                else {
                    _settextposition( 17, 2); _outtext("output evidence map: ");
                    OutBgText( evidence);
                }
                sel = EXECUTE;
              }
            else {
               ProgramError( "No map(s) table exist on your directory (*.TBB).");
               sel = END;
            }
            break;

         /* generates the evidence with proper parameters */
         case EXECUTE:
            if ( rule == DEMPSTER)
               GenerateEvidence( indextbl, inmap, rule, evibel, evipls);
            else
               GenerateEvidence( indextbl, inmap, rule, evidence);

            sel = END;
            break;
      }

   ClearBox( 10, 2, 15, 70);

} /* BuildEvi() */


/*****************************************************************************
name: ReBuildTBA( char *tbafn)

goal: To rebuild the favorability index table.   The reclassified
      favorability values needs to be recalculated when the user edit
      changes the favorability values of the models in the TBA file.

input: tbafn : the favorability table filename

returns: nothing
Global variables changed: none
*****************************************************************************/
void ReBuildTBA( char *tbafn)
{
FILE        *tba;                              /* TBA to rebuild          */
FILE        *newtba;                           /* rebuilt TBA             */
char        tempstr[ MAXSTRTBA]="empty";       /* temporary string        */
char        readstr[ MAXSTRTBA]="also empty";/* used to read strings      */
int         i;                                 /* counter                 */
double      area, prob, bel, pls,
           certfact, fuzzy;                 /* area and model values     */
int         pnts;                              /* points per classes      */

   /* opens up the TBA that needs to be rebuild */
   if ( ( tba = fopen( tbafn, "rt")) == NULL)
      ProgramError( strcat( "Cannot open file:", tbafn));
   else {

      /* opens up the rebuilt TBA (call temp.int until we rename it) */
      newtba = fopen( "temp.int", "w+");
      /* reads in the tba header */
      for (i=0; i < 24; ++i)
        if ( fgets( readstr, MAXSTRTBA - 1, tba) == NULL)
           ProgramError( strcat( "Cannot read file:", tbafn));
        else
           fputs( readstr, newtba);
```

```
      /* Reads in the TBA DATA section  and rebuild the reclassified value */
      /* every PDA in the tba file                                         */
      while ( fgets( readstr, MAXSTRTBA - 1, tba) != NULL) {

        area     = StrGetLastFloat( readstr, FROMLAST, 11);
        pnts     = (int )StrGetLastFloat( readstr, FROMLAST, 10);
        prob     = StrGetLastFloat( readstr, FROMLAST, 9);
        bel      = StrGetLastFloat( readstr, FROMLAST, 8);
        pls      = StrGetLastFloat( readstr, FROMLAST, 7);
        fuzzy    = StrGetLastFloat( readstr, FROMLAST, 6);
        certfact = StrGetLastFloat( readstr, FROMLAST, 5);

        /* copy the associated class and the class name into tempstr      */
        /* by detecting both " ' " (ASCII=39) characters                  */
        strncpy( tempstr, readstr,
           strchr( strchr( readstr, 39)+1, 39) - readstr+1);

        /* writes out the file again */
        fprintf( newtba, "%s %12.7lf %7i %12.7lf %12.7lf %12.7lf"
                      " %12.7lf %12.7lf %4i %4i %4i %4i %4i\n",
                      tempstr, area,
                      pnts, prob, bel, pls,
                      fuzzy, certfact,
                      ReclassifiedModelValue( prob),
                      ReclassifiedModelValue( bel),
                      ReclassifiedModelValue( pls),
                      ReclassifiedModelValue( fuzzy),
                      ReclassifiedModelValue( certfact));

      }

      /* closes files and rename the temporary one as the old one */
      fclose( tba);
      fclose( newtba);
      remove( tbafn);
      rename( "temp.int", tbafn);
   }

} /* ReBuildTBA() */


/*****************************************************************************
name: EditFav( void)

goal: To edit a favorability index by calling the editor Q.EXE.

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void EditFav( void)
{
char  *args[4], prog[ 6];  /* used to call a child process                */
char  fn[ 13];             /* the filename that will be edited            */
int   nofiles;             /* the matching *.TBA number of files          */
int   seltba;              /* the selected TBA file                       */
int   endstr;              /* indicating the end position in the filename */

struct stat filestat1, filestat2;
```

```
    /* ask the user to specify a favorability index (SPANS *.TBA format)     */
    /* by firstly searching the directory for some *.TBA file                 */
    if ( (nofiles=GetFileNames( FAVTABLE_FILTER)) != 0) {
       _settextposition( 10, 2); _outtext("table to edit:");
       if ( ( seltba=SelectOneFileMenu( 10, 17, nofiles)) != ESC) {

         /* eliminate the space(s) between the filename and filetype and   */
         /* prepare the child process parameters                           */
         endstr = strchr( fnames[ seltba], ' ') - fnames[ seltba];
         strncpy( fn, fnames[ seltba], endstr); fn[ endstr] = 0;
         strcat( fn, ".tba");
         strcpy( prog, "q.exe");
         args[0] = prog;
         args[1] = fn;
         args[2] = NULL;

         /* get the time of creation */
         stat( fn, &filestat1 );

         /* call the child process to execute Q.EXE */
         _displaycursor( _GCURSORON);
         spawnvp( P_WAIT, prog, args );
         _displaycursor( _GCURSOROFF);

         /* get the time of creation so we can check if the file           */
         /*     was modified                                               */
         stat( fn, &filestat2);
         if ( filestat1.st_atime != filestat2.st_atime)
            ReBuildTBA( fn);

       }
    }
    else
       ProgramError( "No favorability table(s) exist on your directory (*.TBB).");

    /* clear part of screen */
    ClearBox( 10, 2, 15, 70);

} /* EditFav() */
```

```
/*****************************************************************************
name: BuildEvidence()

goal: To present a menu to build the evidence maps.  Presented options are
      to CREATE a favorability index, EDIT a favorability index, to
      BUILD a evidence map, or to CHOOSE evidence to model.

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void BuildEvidence()
{
int   sel = 0;                                  /* menu selection       */
int   rules;                                    /* prefered model(rule) */
ITEM evidencemnu[] = {                          /* menu structure       */
   { 0, "Create Favorability Table"},
   { 0, "Edit Favorability Table"},
   { 0, "Display Evidence"},
   {19, "Choose Evidence to Model"},
   { 0, ""}
}; enum evidencemnuCHOICES { CREATE, EDIT, BUILD, CHOOSE_EVI};


   /* display and execute options to build and edit evidence maps          */
   /*    until ESC is pressed                                              */
   do {
      sel = MenuVerti( 3, 13, evidencemnu, sel);
      switch( sel) {

        case CREATE:
           /* Create a favorability index */
           CreateFavTable();
           break;

        case EDIT:
           /* Edit a favorability index */
           EditFav();
           break;

        case BUILD:
           /* Build a evidence map with a favorability index to display */
           rules = MenuVerti( 6, 6, rulesmnu, 0);
           if ( rules != ESC) BuildEvi( rules);
           ClearBox( 6, 6, 6, 45);
           break;

        case CHOOSE_EVI:
           /* Chooses some evidence to model */
           ChooseEvidences();
           break;
      }
   } while ( sel != ESC);

   ClearBox( 3, 13, 3, 25);

} /* BuildEvidence() */
```

MODEL.H

```
void ChooseEvidences( void);
```

MODEL.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

   ChooseEvidences()

MODULE INTERNAL FUNCTION(s): All are called from ChooseEvidences()

   ChooseEvidences                : called from BuildEvidences (evidence.c)
   |
   | TransferFavTables
   |
   | GetModelFavTables
   | | DisplayFavTables
   | | ClearAllFavTables
   | | | DisplayFavTables
   | | AddFavTables
   |
   | PutBackFavTables
   |
   | PrepareEvidences
   | | GetUniqueTableData
   | | | ModelEvidencesValid
   | | | ReadUniqueFile
   | | GetAreaPoints
   | | | CalculateP_D
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>
#include <errno.h>
#include <io.h>
#include <conio.h>

#include "favmod.h"
#include "menu.h"
#include "utils.h"

#define     PREPARE_FT          2

struct UNIQUE_TABLE
         unipoly[ MAX_UNIQUE_POLYGONS];              /* unique conditions */
double   area_per_class[ MAX_EVIDENCES][ MAX_CLASSES];/* area/class       */
double   area_per_evidence[ MAX_EVIDENCES];          /* area/evidence    */
double   S_T;                                        /* total area       */
double   PD;                                         /* P( D )           */
int      total_pnts;                                 /* max # of points  */
char     favtablesfn[ MAX_EVIDENCES][ 9];            /* fav.table fn     */
char     evidencesfn[ MAX_EVIDENCES][ 9];            /* evi. filenames   */
int      no_favtables=0;                             /* total # of fav.tbs*/
int      uni_poly;                                   /* # of unique poly. */
```

```
double                  bayest[ MAX_EVIDENCES][ MAX_CLASSES];
double                  belt[ MAX_EVIDENCES][ MAX_CLASSES];
double                  plst[ MAX_EVIDENCES][ MAX_CLASSES];
double                  fuzzyt[ MAX_EVIDENCES][ MAX_CLASSES];
double                  certfactt[ MAX_EVIDENCES][ MAX_CLASSES];


/*************************************************************************
name: DisplayFavTables( char favtables[][ 13], int row, int col)

goal: To display a list of evidences to a maximum of 16.

input: evidences  : the evidences to display
       row        : row to start displaying the evidences
       col        : row to start displaying the evidences

returns: nothing
         none:

Global variables changed: none
*************************************************************************/
void DisplayFavTables( char favtables[][ 13], int row, int col)
{
int   i;

   /* draws a box around the evidence */
   Box(  row-1, col-1, (int )(MAX_EVIDENCES/2) + 1, 27);
   _settextposition( row-1, col + 9); _outtext( "Fav.Tables");
   ClearBox( row, col, (int )(MAX_EVIDENCES/2), 25);

   /* displays the fav tables */
   for (i=0; i < no_favtables; ++i) {
      _settextposition( row + (int )i/2, col + ( int)(fmod( i, 2))*15);
      OutBgText( favtables[ i]);
   }

} /* DisplayFavTables() */


/*************************************************************************
name: ClearAllFavTables( char favtables[][ 13])

goal: To clear the display box and the favorability table list.

input: evidences : the evidence to clear

returns: nothing

Global variables changed: no_favtables : the # of evidences is set to zero
*************************************************************************/
void ClearAllFavTables( char favtables[][ 13])
{
   no_favtables = 0;
   DisplayFavTables( favtables, 9, 20);

} /* ClearAllFavTables() */
```

```
/*****************************************************************************
name: AddFavTables( int pos, char favtables[][ 13])

goal: Adds a evidence in the evidence list only if it doesnt exist.  It
      deletes the evidence if it exist.

input: pos       : postion of the evidence to add/delete in variable 'fnames'
       evidence : list of evidence

returns: nothing

Global variables changed: no_favtables : the # of favorability tables
*****************************************************************************/
void AddFavTables( int pos, char favtables[][ 13])
{
int      i;

   /* check if the new evidence already exist, if yes we delete it   */
   /* if not we add it                                               */

   for ( i=0; i<no_favtables; ++i)
      if ( !strcmp( favtables[ i], fnames[ pos])) break;

   if ( i != no_favtables) {  /* delete the evidence */
      memmove( favtables[ i], favtables[ i+1],
               sizeof( favtables[0])*(no_favtables-i-1));
      favtables[ no_favtables-1][ 0] = 0;
      --no_favtables;
   }
   else {               /* add the evidence    */
      if ( no_favtables < MAX_EVIDENCES) {
         strcpy( favtables[ no_favtables], fnames[ pos]);
         favtables[ no_favtables+1][ 0] = 0;
         ++no_favtables;
      }
   }
   /* redisplays the new evidence */
   DisplayFavTables( favtables, 9, 20);

} /* AddFavTables() */
```

```
/*****************************************************************************
name: GetModelFavTables( int nofiles, char favtables[][ 13])

goal: To display and manage a menu for selecting fav. tables.

input: nofiles    : # of matching *.MAP files on disk
       favtables  : previous selected favorability table list

returns:

  via function parameters:
        evidences          : final selected evidence list
  via function:

        PREPARE_FT         : the user has selected the '< accept >' option
                             in the evidence list that means acceptation
                             of the present list of evidence for modeling
                or
        ESC                : the use has cancelled this operation with the
                             ESC key

Global variables changed: none
*****************************************************************************/
int GetModelFavTables( int nofiles, char favtables[][ 13])
{
int      iPrev=0,iCur=0;/* counter tracking current and previous cursor pos*/
int      i;             /* utility counter                                 */
int      noitem;        /* maximum number of items displayed at one time   */
int      starti;        /* first top menu itme to be displayed             */
int      prevstarti=0;  /* previous top menu item to be displayed          */
unsigned uKey;          /* Unsigned key code                               */
int      row, col;      /* row and column used to display the file menu    */

long     bgColor;       /* screen background color                         */
short    fgColor;       /* screen forground color                          */
struct   rccoord rc;    /* cursor position                                 */

   row = 9;
   col = 3;

   /* saving old colors and cursor position */
   bgColor = _getbkcolor();
   fgColor = _gettextcolor();
   rc = _gettextposition();

   /* determine the maximum number of items to displayed */
   if ( nofiles > ( 18 - row)) noitem  = 18-row;
   else                        noitem  = nofiles;

   /* displaying a box for the menu */
   Box( 8, 2, noitem, strlen( fnames[1])+2);

   /* displays box and selected files in the model */
   DisplayFavTables( favtables, 9, 20);  /*7.23*/

   /* Put items on menu. */
   Itemize2( row, col, TRUE, fnames[ 0]);
   for( i = 1; i < noitem; i++ )
      Itemize2( row + i, col, FALSE, fnames[i]);

   /* displays  "MORE..." to indicate more files for selection */
   _settextcolor( mnuAtrib.fgBorder);
   _setbkcolor( mnuAtrib.bgBorder);
   if ( noitem != nofiles) {
      _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
      _outtext( "more...");
   }
```

```
else {
   _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
   _outtext( "        ");
}

/* while a selection or ESC key do ... */
while( TRUE)
{
   /* Wait until a uKey is pressed, then evaluate it. */
   uKey = GetKey( WAIT );
   switch( uKey )
   {
      case UPKEY:                    /* Up key        */
         iPrev = iCur;
         iCur = (iCur > 0) ? (--iCur % nofiles) : nofiles - 1;
         break;
      case DOWNKEY:                          /* Down key      */
         iPrev = iCur;
         iCur = (iCur < nofiles) ? (++iCur % nofiles) : 0;
         break;

      case ESC:
         _setbkcolor( bgColor );
         _settextcolor( fgColor );
         _settextposition( rc.row, rc.col );
         ClearBox( 3, 1, 23, 60);
         return( ESC);
      case SPACE:
      case CR:
         iPrev = iCur;
         if ( iCur == 1) {
            _setbkcolor( bgColor );
            _settextcolor( fgColor );
            _settextposition( rc.row, rc.col );
            return( PREPARE_FT);
         }
         if ( iCur == 0) ClearAllFavTables( favtables);
         else AddFavTables( iCur, favtables);
         break;
   }
   /* Redisplay current and previous menu item. */

   starti = noitem * ( int)floor( iCur/ noitem);

   if ( prevstarti == starti)
      /* de-hilights the appropriate selection */
      Itemize2( row + iPrev -( int)floor( iPrev / noitem) * noitem,
                col, FALSE, fnames[ iPrev]);
   else {
      prevstarti = starti;

      /* displays "MORE..." to indicate more files for selection */
      _settextcolor( mnuAtrib.fgBorder);
      _setbkcolor( mnuAtrib.bgBorder);
      if ( ( nofiles-starti) < noitem) {
         _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
         _outtext( "        ");
      }
      else {
         _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
         _outtext( "more...");
      }
      /* Put items on menu. */
      for( i = 0; i < noitem; i++ )
            Itemize2( row + i, col, FALSE, fnames[ starti + i]);
   }
   /* hilights the appropriate selection */
```

```
      Itemize2( row + iCur -( int)floor( iCur / noitem) * noitem,
                col, TRUE, fnames[ iCur]);

   } /* end while */

} /* GetModelFavTables() */


/*****************************************************************************
name: TransferFavTables( char favtables[][ 13])

goal: To transfer the fav.table filenames into a more workable array because
      the global variable 'favtablesfn' does not contains the file extension
      'MAP' attach to every evidence and makes it hard to manipulate such
      string.

input: favtables : the 'workable' list of evidence

returns: nothing
Global variables changed: none
*****************************************************************************/
void TransferFavTables( char favtables[][ 13])
{
int   i;                                /* utilitary counter                */

   for (i=0; i < no_favtables; ++i) {
      strcpy( favtables[ i], favtablesfn[  i]);
      strcat( favtables[ i], " MAP");
   }

} /* TransferFavTables() */


/*****************************************************************************
name: PutBackFavTables( char favtables[][ 13])

goal: To transfer back the 'workable' fav. table list into the global variable
      'favtablesfn'

input: favtables : the workable favorability table list

returns: nothing

Global variables changed: favtables : the favorability table list
*****************************************************************************/
void PutBackFavTables( char favtables[][ 13])
{
int   i;             /* utilitary counter */

   for (i=0; i < no_favtables; ++i) {
      strncpy( favtablesfn[ i], favtables[ i], 8);
      favtablesfn[ i][ 8] = 0;
   }

} /* PutBackFavTables() */


int FindFavTablePos( char *e)
{
int   i;

   for ( i=0; i < no_favtables; ++i) {
      if ( !strcmp( e, evidencesfn[ i])) break;
      if ( i == no_favtables-1)          break;
   }
   return( i);
} /* FindFavTablePos() */
```

```
      /* reads in the signification of each columns      */
      i=0;
      while ( TRUE) { /* reads in the header */
         if ( fgets( instr, MAXSTRTBA - 1, unif) == NULL) {
            ProgramError( "Invalid format for UNIQUE.TBA file");
            break;
         }
         else {
            if ( !strncmp( instr, "DATA", 4))
               break;
            sscanf( instr, "%*i %*i %*f %*i %s", evifn[ i++]);
         }
      }

      /* reads in the rest of the data */
      if ( no_favtables != i)
         ProgramError( "Please redo this operation with the correct evidences.");
      else
         /* checks if the selected evidence match the one in the unique
            condition file                                                */
         if ( ModelEvidencesValid( evifn))
            ReadUniqueFile( unif, evifn);
         else
            ProgramError( "Please redo this operation with the correct evidences.");
   }
   fclose( unif);

} /* GetUniqueTableData() */

/*************************************************************************
name: GetAreaPoints()

goal: To get the area and # of points for the classes of the every evidence.
      The function also checks the validity of the favorability table for
      every TBA.

input: none
returns: nothing
         none:
Global variables changed:
          - total_pnts : maximum # of points in the fav. table
          - PD         : probalistic favorability value computed with the
                         largest area and the biggest # of points
*************************************************************************/
void GetAreaPoints()
{
FILE     *intbaf;              /* TBA file containing points/class          */
char     tmpstr[ MAXSTRTBA];   /* temporary string                          */
char     ftfn[ 13];            /* temporary string                          */
int      ft;                   /* # of favorability tables                  */
int      allpnts;              /* # of points / class                       */
double   allarea;              /* ares of a class                           */
BOOL     readingerror;         /* flag TRUE when the TBA is not valid       */
int      class;

   /* initializing global variables */
   allpnts    = 0;
   total_pnts = 0;
   for ( ft=0; ft<MAX_EVIDENCES; ++ft)
      for ( class=0; class<MAX_CLASSES; ++class) {
         bayest[ ft][ class] = 0;
         belt[ ft][ class] = 0;
         plst[ ft][ class] = 0;
         fuzzyt[ ft][ class] = 0;
         certfactt[ ft][ class] = 0;
      }
```

```
    /* reads in the TBA file to determine the points/class if the filename  */
    /* exist                                                                */
    for ( ft=0; ft < no_favtables; ++ft) {
       /* prepares the filename to open (add TBA to it) */

       strcpy( ftfn, favtablesfn[ ft]);

       if ( strlen( ftfn) != 8)
          ftfn[ strchr( ftfn, ' ') - ftfn] = 0;
       strcat( ftfn, ".TBA");

       if ( (intbaf = fopen( ftfn, "rt")) == NULL) {
          ProgramError( strcat( "Cannot open file:", ftfn));
          break;
       }
       else {

          /* Skips lines until 'DATA' is hit */
          readingerror = FALSE;
          do
             if ( fgets( tmpstr, MAXSTRREP - 1, intbaf) == NULL) {
                ProgramError( strcat( "Cannot read file:", ftfn));
                readingerror = TRUE;
                break;
             }
          while ( strncmp( tmpstr, "DATA", 4) != (int )NULL);

          /* counting # points per class */
          if ( !readingerror) {
             allpnts = 0; allarea = 0;
             while ( fgets( tmpstr, MAXSTRTBA - 1, intbaf) != NULL) {
                class =  (int )StrGetLastFloat( tmpstr, FROMSTART, 1);
                bayest[ ft][ class] = StrGetLastFloat( tmpstr, FROMLAST, 9);
                belt[ ft][ class] = StrGetLastFloat( tmpstr, FROMLAST, 8);
                plst[ ft][ class] = StrGetLastFloat( tmpstr, FROMLAST, 7);
                fuzzyt[ ft][ class] = StrGetLastFloat( tmpstr, FROMLAST, 6);
                certfactt[ ft][ class] = StrGetLastFloat( tmpstr, FROMLAST, 5);

                allpnts += (int )StrGetLastFloat( tmpstr, FROMLAST, 10);
                allarea += StrGetLastFloat( tmpstr, FROMLAST, 11);
             }
             fclose( intbaf);
             if ( max( allpnts, total_pnts) == allpnts)
                total_pnts = allpnts;
          }
       }
    }

    PD = CalculateP_D( S_T, total_pnts);

} /* GetAreaPoints() */
```

```
void DisplayEvidences( int row, int col)
{
int   i;
char  ft[ 13];
FILE  *intbaf;
char  tmpstr[ MAXSTRTBA];
char  evi[ 13];
char  evifn[ 13];
int   pos;
int   len;

   pos = 0;
   /* draws a box around the evidence */
   Box(  row-1, col-1, (int )(MAX_EVIDENCES/2) + 1, 27);
   _settextposition( row-1, col + 4); OutBgText( "Evidences to Choose");
   ClearBox( row, col, (int )(MAX_EVIDENCES/2), 25);

   /* Gets the evidence from the fav. table and display them one by one    */

   for ( i=0; i<no_favtables; ++i) {


      strcpy( ft, favtablesfn[ i]);
      if ( strchr( ft, ' ') != NULL)
         ft[ strchr( ft, ' ') - ft] = 0;
      strcat( ft, ".TBA");

      /* opens the file and read the first line, that is the ID field   */
      if ( (intbaf = fopen( ft, "rt")) == NULL)
         ProgramError( strcat( "Cannot open file:", evifn));

      else {
         fgets( tmpstr, MAXSTRTBA - 1, intbaf);
         fgets( tmpstr, MAXSTRTBA - 1, intbaf);
         sscanf( tmpstr, "%*s%s", evi);

         strcpy( evidencesfn[ i], evi);
         strcpy( evifn, evi);
         strcat( evifn, ".MAP");

         for ( len=strlen( evi); len<8; ++len)
            evi[ len] = ' ';
         evi[ 8] = 0; strcat( evi, " MAP");

         if ( EXIST( evifn)) {
            _settextposition( row + (int )pos/2, col + ( int)(fmod( pos, 2))*15);
            _outtext( evi);
            ++pos;
         }
      }
   }

} /* DisplayEvidences() */
```

```
/*****************************************************************************
name: PrepareEvidences()

goal: To ask the user to execute some command in SPANS in order to generate
      the unique condition table and export it.  The function waits or loop
      until the unique condition table has been exported by the user and then
      calls some functions to get the data from it.

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void PrepareEvidences()
{
int      c=0;           /* input char from keyboard   */
int      evi;           /* evidence counter           */
long     wastetime;     /* counter for wasting time   */

   /* checks for at least two evidences in the model  */
   if ( no_favtables >= 2) {

      /* displays the evidence to specify in the unique condition table */
      DisplayEvidences( 9, 20);

      /* tells the user what to do in SPANS */
      _settextposition( 19, 2); _outtext("SPANS: Model/Overlay/Unique Conditions/
map=UNIQUE");
      _settextposition( 20, 2); _outtext("SPANS: Transform/Export/Library/Table/
table=UNIQUE");

      /* starts waiting for SPANS ... */
      /*
      remove( "unique.map");
      remove( "unique.tba");
      remove( "unique.tbb");
      */

      evi = 0;
      while ( TRUE) {

         for ( wastetime=0; wastetime<70000; ++wastetime);
         _settextposition( 22, 2); _outtext("Waiting for SPANS ... \\");

         if ( CompletedJob( "unique.tba"))
            break;

         for ( wastetime=0; wastetime<70000; ++wastetime);
         _settextposition( 22, 2); _outtext("Waiting for SPANS ... -");

         if ( kbhit()) {
            c = getch();
            if ( c == ESC) break;
         }

         for ( wastetime=0; wastetime<70000; ++wastetime);
         _settextposition( 22, 2); _outtext("Waiting for SPANS ... /");
      }

      if ( c != ESC) {

         /* gets the data from SPANS unique condition table and TBA files */
         GetUniqueTableData();
         GetAreaPoints();
```

```
         _settextposition( 24, 2); _outtext("Now ready to model the specific integration
rule.");
         _settextposition( 25, 2); _outtext("Press any key to continue...");
         while ( !kbhit());
         getch();
      }

   }
   else
      ProgramError( "Plese select at least two evidences in the model.");

   ClearBox( 5, 2, 20, 55);

} /* PrepareEvidences() */


/*****************************************************************************
name: ChooseEvidences()

goal: This function is responsible for calling functions that will
            - offer a menu to choose evidence
            - generate the unique condition table
            - get the data from the unique condition table

input: none
returns: nothing
Global variables changed: fnames : files with matching *.map
*****************************************************************************/
void ChooseEvidences()
{
char  favtables[ MAX_EVIDENCES][ 13] = { {""}}; /* evidence list          */
int   nofiles;                                  /*matching # of *.map files*/
int   retcode;                                  /* internal funct. ret.code*/


   /* get the *.TBA matching files and store the result in var 'fnames'    */
   if ( (nofiles=GetFileNames( FAVTABLE_FILTER)) != 0)
      {
      /* if some *.map exist then we add two functions to the filename     */
      /* list, these function either clear the evidence list or accept the */
      /* list                                                              */

      memmove( fnames[ 2], fnames[ 0], sizeof( fnames[ 0])*( nofiles+2));
      strcpy( fnames[ 0], "<clear  all>");
      strcpy( fnames[ 1], "<  accept  >");

      /* transfer the evidences names into a more workable array */
      TransferFavTables( favtables);

      /* to model any integration rule, more than one evidence must exist  */
      if ( nofiles < 2)
         ProgramError( "Not enough evidences in your universe (FT*.TBA).");
      else {
         retcode = GetModelFavTables( nofiles+2, favtables);
         PutBackFavTables( favtables);
         if ( retcode == PREPARE_FT) PrepareEvidences();
      }
   }
   else
      ProgramError( "No evidences exist on your directory (*.MAP).");

   ClearBox( 3, 3, 23, 70);

} /* ChooseEvidences() */
```

RULES.H

```
void Rules( void);
```

RULES.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

   Rules()

MODULE INTERNAL FUNCTION(s): All are called from Rules()

Rules                          : called from main (favmod.c)
|
| ModelTechnique
| |
| | ExecuteModel
| | |
| | | GenerateModelTBA
| | | |
| | | | CalculateRegression
| | | | | dmatrix
| | | | | dvector
| | | | | FillRegMatrices
| | | | | | S_XY
| | | | | GaussJordan
| | | | | | ivector
| | | | | | free_ivector
| | | | | free_dmatrix
| | | | | free_dvector
| | | | |
| | | | CalculateBayes
| | | | | PABC_D
| | | | | PABC
| | | | |
| | | | CalculateDempster
| | | | | DempsterBel
| | | | | DempsterBelp
| | | | |
| | | | CalculateFuzzy
| | | | CalculateCertainty
| | | | ReclassifiedModelValue
*****************************************************************************/
#include <stdio.h>
#include <conio.h>
#include <graph.h>
#include <stdlib.h>
#include <math.h>
#include <errno.h>
#include <string.h>

#include "menu.h"
#include "favmod.h"
#include "utils.h"

double   mat[ MAX_EVIDENCES+1][ MAX_EVIDENCES+1];
double   na[  MAX_EVIDENCES+1];
double   results[  MAX_EVIDENCES+1];
```

```
/*****************************************************************************
 The following MACROS simplify the access of variables.  The
 definition for each is:

  S_A( EVI)          : Area covered (km2) the EVI-th evidence.

  S_Ai( POLY,EVI)    : Area covered (km2) by the class contained in the
                       POLY-th polygon of the EVI-th evidence.

  P_Ai( POLY,EVI)    : S_Ai( POLY, EVI) / total surface covered by the area

*****************************************************************************/
#define     SWAP( a, b)             { double temp=(a);(a)=(b);(b)=temp;}
#define     S_A( evi)               area_per_evidence[ evi]
#define     S_Ai( poly, evi)        area_per_class[ evi][ unipoly[ poly].class[ evi] ]
#define     P_Ai( poly, evi)        S_Ai( poly, evi) / S_T

#define     bayes_value( poly, evi)      bayest[ evi][ unipoly[poly].class[evi]]
#define     bel_value( poly, evi)        belt[ evi][ unipoly[poly].class[evi]]
#define     pls_value( poly, evi)        plst[ evi][ unipoly[poly].class[evi]]
#define     fuzzy_value( poly, evi)      fuzzyt[ evi][ unipoly[poly].class[evi]]
#define     certfact_value( poly, evi)   certfactt[ evi][ unipoly[poly].class[evi]]

/* menu definition for displaying the different integration rules (menu.h) */
extern ITEM rulesmnu[];


/*****************************************************************************
name: PABC_D( int poly)

goal: Assuming the three evidences A, B, C with respectively i, j, and k
      classes, this function calculates (see Chang-Jo Chung
      personal notes, p. 4)

   (P{Ai}*P{Bj}*P{Ck}) *P{D} * P{D|Ai}  * P{D|Bj}  * P{D|Ck}
                               -------    -------    -------
                                 P{D}       P{D}       P{D}

input: poly : unique polygon number

returns: the result of the above formula
Global variables changed: none
*****************************************************************************/
double PABC_D( int poly)
{
double   result;
int      evi;

   /* Calculating P{Ai} * P{Bj} * P{Ck}                    */
   result = 1;
   for ( evi=0; evi < no_favtables; ++evi)
      if ( unipoly[ poly].class[ evi] != 0)
        result *= P_Ai( poly, evi);

   /* Calculating  (P{Ai}*P{Bj}*P{Ck}) * P{D}              */
   result *= PD;

   /* Calculating  (P{Ai}*P{Bj}*P{Ck}) *P{D} * (P{D|Ai}/P{D}) *          */
   /*              (P{D|Bj}/P{D}) * (P{D|Ck}/P{D})                       */
   for ( evi=0; evi < no_favtables; ++evi)
      if ( unipoly[ poly]. class[ evi] != 0)
         result *= ( bayes_value( poly, evi) / PD);

   return( result);

} /* PABC_D() */
```

```
/*****************************************************************************
name: PABC( int poly)

goal: Assuming the three evidences A, B, C with respectively i, j, and k
      classes, this function calculates (see Chang-Jo Chung
      personal notes, p. 4)

                    S(Ai^Bj^Ck) / Total surface

      that is the area spans by the intersection of the 3 evidence for
      the specific classes contain in a unique polygon.

input: poly : unique polygon number

returns: S(Ai^Bj^Ck) / total surface of the study area
Global variables changed: none
*****************************************************************************/
double PABC( int poly)
{
int      evi, i;
double   S_AiBiCi;
BOOL     match;

   /* Calculating S(Ai^Bi^Ci)    */
   S_AiBiCi = 0;
   for ( i=0; i < uni_poly; ++i) {
      match = TRUE;
      for ( evi=0; evi < no_favtables; ++evi)
         if ( unipoly[ poly]. class[ evi] != 0 ) {
            if ( unipoly[ i]. class[ evi] != unipoly[ poly].class[ evi])
               match = FALSE;
         }
      if ( match) S_AiBiCi += unipoly[ i].area;
   }

   return( S_AiBiCi / S_T);

} /* PABC() */


/*****************************************************************************
name: CalculateBayes( int poly)

goal: To calculate Bayes model for each unique polygon.

input: poly : unique polygon number

returns: bayes value

Global variables changed: none
*****************************************************************************/
double CalculateBayes( int poly)
{

   return( PABC_D( poly) / PABC( poly));

} /* CalculateBayes() */
```

```
/*************************************************************************
name: DempsterBel( double a, double ap, double b, double bp)

goal: To calculate Dempster Belief function for each unique polygon that
      a pixel contains a deposit.

input:  a  : possibility that the pixel contains the deposit for layer A
        ap : possibility that the pixel does not contain the deposit for A
        b  : possibility that the pixel contains the deposit for layer B
        bp : possibility that the pixel does not contain the deposit for B

returns: belief function value

Global variables changed: none
*************************************************************************/
double DempsterBel( double a, double ap, double b, double bp)
{
   return( (a*b   + a*(1.0-b-bp)  + b*(1.0-a-ap) ) / (1.0-a*bp-ap*b));
}


/*************************************************************************
name: DempsterBelp( double a, double ap, double b, double bp)

goal: To calculate Dempster Belief function for each unique polygon that
      a pixel DOES NOT contain a deposit.

input:  a  : possibility that the pixel contains the deposit for layer A
        ap : possibility that the pixel does not contain the deposit for A
        b  : possibility that the pixel contains the deposit for layer B
        bp : possibility that the pixel does not contain the deposit for B

returns: belief function value

Global variables changed: none
*************************************************************************/
double DempsterBelp( double a, double ap, double b, double bp)
{
   return( (ap*bp + ap*(1.0-b-bp) + bp*(1.0-a-ap)) / (1.0-a*bp-ap*b));
}


/*************************************************************************
name: CalculateDempster( int poly, double *bel, double *pls)

goal: To calculate Dempster's and Shafer Belief and Plausible functions.

input: poly: the unique polygon number

returns: bel : Belief function value
         pls : Plausible function value

Global variables changed: none
*************************************************************************/
void CalculateDempster( int poly, double *bel, double *pls)
{
int      evi;


   *bel = DempsterBel(  bel_value( poly, 0),
                        1.0 - pls_value( poly, 0),
                        bel_value( poly, 1),
                        1.0 - pls_value( poly, 1));
   *pls = DempsterBelp( bel_value( poly, 0),
                        1.0 - pls_value( poly, 0),
                        bel_value( poly, 1),
                        1.0 - pls_value( poly, 1));
```

```
   for ( evi=2; evi < no_favtables; ++evi) {
      *bel = DempsterBel(  *bel, *pls, bel_value( poly, evi), 1.0 - pls_value( poly, evi));
      *pls = DempsterBelp( *bel, *pls, bel_value( poly, evi), 1.0 - pls_value( poly, evi));
   }

   *pls = 1.0 - *pls;

} /* CalculateDempster() */


/*****************************************************************************
name: GaussJordan( int n)

goal: Performs a Gass-Jordan Elimination on matrix 'mat' of dimension 'n'.

input: mat : the input matrix
       n   : the matrix dimension

returns: mat : the result is stored in the input matrix

Global variables changed: none
*****************************************************************************/
void GaussJordan( int n)
{
int      indxc[ MAX_EVIDENCES+1], indxr[ MAX_EVIDENCES+1], ipiv[ MAX_EVIDENCES+1];
int      i=0, icol=0, irow=0, j=0, k=0, l=0, ll=0;
double   big=0.0, dum=0.0, pivinv=0.0;

   for ( j=1; j <= n; j++) ipiv[j]=0;
   for ( i=1; i <= n; i++) {
      big = 0.0;
      for ( j=1; j <= n; j++)
         if ( ipiv[ j] != 1)
            for ( k=1; k <= n; k++) {
               if ( ipiv[ k] == 0) {
                  if ( fabs( mat[ j][ k]) >= big) {
                     big  = fabs( mat[ j][ k]);
                     irow = j;
                     icol = k;
                  }
               }
               else
                  if ( ipiv[ k] > 1) ProgramError( "GAUSSJ: Singular Matrix-1");
            }
      ++( ipiv[ icol]);
      if ( irow != icol)
         for ( l=1; l <= n; l++) SWAP( mat[ irow][ l], mat[ icol][ l])

      indxr[ i] = irow;
      indxc[ i] = icol;
      if ( mat[ icol][ icol] == 0.0) ProgramError( "GAUSSJ: Singular Matrix-2");
      pivinv = 1.0 / mat[ icol][ icol];
      mat[ icol][ icol] = 1.0;
      for ( l=1;  l  <= n; l++) mat[ icol][ l] *= pivinv;
      for ( ll=1; ll <= n; ll++)
         if ( ll != icol) {
            dum = mat[ ll][ icol];
            mat[ ll][ icol] = 0.0;
            for ( l=1; l <= n; l++) mat[ ll][ l] -= mat[ icol][ l]* dum;
         }
   }

   for ( l=n; l >= 1; l--) {
      if ( indxr[ l] != indxc[ l])
      for ( k=1; k <= n; k++) SWAP( mat[ k][ indxr[ l]], mat[ k][ indxc[ l]]);
   }
} /* GaussJordan() */
```

```
/*****************************************************************************
name: S_XY( int poly, int x, int y)

goal: To calculate the area spans by the intersection of two classes
      contained in a unique polygon.

input: poly: unique polygon number
       x   : index of the first class
       y   : index of the second class

returns: area spans by the intersection

Global variables changed: none
*****************************************************************************/
double S_XY( int poly, int x, int y)
{
int      i;
double   S_XiYj;

   /* Calculating S(Xi^Yj)    */
   S_XiYj = 0;
   for ( i=0; i < uni_poly; ++i) {
      if ( ( unipoly[ poly].class[ x] == unipoly[ i].class[ x]) &&
         ( unipoly[ poly].class[ y] == unipoly[ i].class[ y]))
         S_XiYj += unipoly[ i].area;
   }

   return( S_XiYj);

} /* S_XY() */

/*****************************************************************************
name: FillRegMatrices( int poly, int n, int used_evi[])

goal: To prepare the matrix and the vectors so the computation of the
      regression can be preformed (see Chang_Jo personal notes p. R3).

input: poly     : unique polygon number
       n        : # of evidences used in the computation
       used_evi : evidences used in the computation
returns: regmat  : matrix used in the computation
         regvect : vector used in the computation
Global variables changed: none
*****************************************************************************/
void FillRegMatrices( int poly, int n, int used_evi[])
{
int      i, j;

   mat[ 1][ 1] = 1.0;
   /* fills in the regression matrix */
   for ( i=2; i <= n; ++i) {
      mat[ 1][ i] = P_Ai( poly, used_evi[ i-2]);
      mat[ i][ 1] = P_Ai( poly, used_evi[ i-2]);
      mat[ i][ i] = P_Ai( poly, used_evi[ i-2]);
   }
   for ( i=1; i <= n; ++i)
      for ( j=1; j <= n; ++j)
         if ( (i != 1) && (j != 1) && (i != j))
            mat[ i][ j] =
            S_XY( poly, used_evi[ i-2], used_evi[ j-2]) / S_T;

   /* fills in the regression vector */
   na[ 1] = 1.0;
   for ( i=2; i <= n+1; ++i)
      na[ i] = ( log( 1.0 - bayes_value( poly, used_evi[ i-2])) /
                 log( 1.0 - atof(unitsize)/S_A( used_evi[ i-2]))) / total_pnts;
} /* FillRegMatrices() */
```

```
/*****************************************************************************
name: CalculateRegression( int poly)

goal: To compute the value of the regression model for every unique
      polygon.

input: poly : unique polygon number

returns: regvalue : regression value

Global variables changed: none
*****************************************************************************/
double CalculateRegression( int poly)
{
int      n;                          /* matrix rank used for computation   */
int      k, l;                       /* matrix row and column              */
int      used_evi[ MAX_EVIDENCES];   /* evidences with mapped areas        */
double   regvalue;                   /* regression model value             */

   n=1;
   /* determine the rank of the matrix */
   for ( k=0; k < no_favtables; ++k)
      if ( unipoly[ poly].class[ k]) {
         used_evi[ n-1] = k;
         ++n;
      }

   /* puts data in matrix mat and vector na */
   FillRegMatrices( poly, n, used_evi);

   /* invert the matrix mat */
   GaussJordan( n);

   /* multiplying the inverse mat with the points vector */
   for ( k=1; k <= n; k++) {
      results[ k] = 0.0;
      for ( l=1; l <= n; l++)
            results[ k] += ( mat[ k][ l] * na[ l]);
   }

   /* multiplying results by ( n/t) */
   regvalue = 0.0;
   for ( l=1; l <= n; l++)
         regvalue += ( results[ l] * ( total_pnts/S_T));

   return( regvalue);

} /* CalculateRegression */


/*****************************************************************************
name: CalculateFuzzy( int poly)

goal: To compute the value of the fuzzy set model for every unique
      polygon.

input: poly : unique polygon number

returns: fuzzy : fuzzy set value

Global variables changed: none
*****************************************************************************/
double CalculateFuzzy( int poly)
{
double fuzzy;
   fuzzy = 0.0; return( fuzzy);
} /* CalculateFuzzy() */
```

```
/*****************************************************************************
name: CalculateCertainty( int poly)

goal: To compute the value for the certainty factor model of every unique
      polygon.

input: poly : unique polygon number

returns: cfact : certainty factor

Global variables changed: none
*****************************************************************************/
double CalculateCertainty( int poly)
{
double cfact;

   cfact = 0.0;
   return( cfact);

} /* CalculateFuzzy() */


/*****************************************************************************
name: GenerateModelTBA( int modeltype)

goal: To generate SPANS table file that contains the result of modeling
      the information layers.

input: modeltype : the modeling function (BAYES, REGRESSION, DEMPSTER,
                   FUZZY, or CERTAINTY)

returns: nothing
Global variables changed: none
*****************************************************************************/
void GenerateModelTBA( int modeltype)
{
double   modelvalue=0;                  /* reg, bayes, certainty, and fuzzy
                                           model value                    */
double   bel, pls;                      /* belief and plausible functions  */
int      poly;                          /* unique polygon number           */
int      evi;                           /* evidence counter                */
char     tbafn[ 13];                    /* TBA model table filename        */
char     modelname[ 50];                /* model name (reg, bayes, ...)    */
int      i;                             /* utility counter                 */
FILE     *outtba;                       /* output TBA file pointer         */
char     tempstr[ 20];                  /* utility strings                 */
double   favvalue;
char     TBAHeader[ 13][ 65] = {        /* SPANS TBA file header format    */
            "ID ?",
            "TITLE ?",
            "MAPID ??",
            "WINDOW 00 0 0 0 0",
            "TABTYPE 5",
            "FTYPE free",
            "KEYFIELD 1",
            "KEYBASE 0",
            "NRECORD x",
            " 1  4   7.0 0   unique   Unique plolygon number",
            " 2  4   7.0 0   ?        ?",
            " 3  1  11.6 0   area     Area (square km)"};
```

```
   /* fills in the header information */
   itoa( uni_poly, tempstr, 10);
   strcpy( &TBAHeader[ 8][8], tempstr);
   switch( modeltype) {
      case( BAYES):
         strcpy( &TBAHeader[ 0][3], "bayes");
         strcpy( modelname, "Baysian Integration rule under CI");
         strcpy( &TBAHeader[ 1][6], modelname);
         strcpy( TBAHeader[ 10],   " 2  4   7.0 0   rcbayes  Reclassified Baysian value
under CI");
         strcpy( TBAHeader[ 11],   " 3  1  11.6 0   area     Area (square km)");
         strcpy( tbafn, "bayes.tba");
         break;
      case( REGRESSION):
         strcpy( &TBAHeader[ 0][3], "reg");
         strcpy( modelname, "Multivariate Regression Analysis");
         strcpy( &TBAHeader[ 1][6], modelname);
         strcpy( TBAHeader[ 10],   " 2  4   7.0 0   rcreg    Reclassified Regression
value");
         strcpy( TBAHeader[ 11],   " 3  1  11.6 0   area     Area (square km)");
         strcpy( tbafn, "reg.tba");
         break;
      case( DEMPSTER):
         strcpy( &TBAHeader[ 0][3], "dempster");
         strcpy( modelname, "Dempster-Shafer");
         strcpy( &TBAHeader[ 1][6], modelname);
         strcpy( TBAHeader[ 10],   " 2  4   7.0 0   rcbel    Reclassified Belief funct.
value");
         strcpy( TBAHeader[ 11],   " 3  4   7.0 0   rcpls    Reclassified Plausible funct.
value");
         strcpy( TBAHeader[ 12],   " 4  1  11.6 0   area     Area (square km)");
         strcpy( tbafn, "dempster.tba");
         break;
      case( FUZZY):
         strcpy( &TBAHeader[ 0][3], "fuzzy");
         strcpy( modelname, "Fuzzy Logic Set");
         strcpy( &TBAHeader[ 1][6], modelname);
         strcpy( TBAHeader[ 10],   " 2  4   7.0 0   rcfuzzy  Reclassified Fuzzy Logic
value");
         strcpy( TBAHeader[ 11],   " 3  1  11.6 0   area     Area (square km)");
         strcpy( tbafn, "fuzzy.tba");
         break;
      case( CERTAINTY):
         strcpy( &TBAHeader[ 0][3], "cert");
         strcpy( modelname, "Certainty Factor");
         strcpy( &TBAHeader[ 1][6], modelname);
         strcpy( TBAHeader[ 10],   " 2  4   7.0 0   rccert   Reclassified Certaity Factor
value");
         strcpy( TBAHeader[ 11],   " 3  1  11.6 0   area     Area (square km)");
         strcpy( tbafn, "cert.tba");
         break;
   }

   /* opens the tba file and starts writing it */
   outtba = fopen( tbafn, "w+");

   /* writes out the TBA header */
   for (i=0; i < 12 + (modeltype==DEMPSTER); ++i)
      fprintf( outtba, "%s\n", TBAHeader[ i]);

   /* writes out a description line for every evidence */
   for ( evi=0; evi < no_favtables; ++evi) {
      if ( modeltype == DEMPSTER) {

         fprintf( outtba, "%2i  1  12.7 0   B%7s Belief value of %8s\n",
              (evi*2)+5, evidencesfn[ evi], evidencesfn[ evi]);
         fprintf( outtba, "%2i  1  12.7 0   P%7s Plausible value of %8s\n",
```

```
                (evi*2)+6, evidencesfn[ evi], evidencesfn[ evi]);
        }
        else
           fprintf( outtba, "%2i  1  12.7 0   %8s Favorability value of %8s\n",
                evi+4, evidencesfn[ evi], evidencesfn[ evi]);

     }
     switch( modeltype) {
        case( REGRESSION):
           fprintf( outtba, "%2i  1  12.7 0   reg      %s\n", evi+4, modelname);
           break;
        case( BAYES):
           fprintf( outtba, "%2i  1  12.7 0   bayes    %s\n", evi+4, modelname);
           break;
        case( DEMPSTER):
           fprintf( outtba, "%2i  1  12.7 0   bel      Belief function value\n",
(no_favtables*2)+5);
           fprintf( outtba, "%2i  1  12.7 0   pls      Plausible function value\n",
(no_favtables*2)+6);
           break;
        case( FUZZY):
           fprintf( outtba, "%2i  1  12.7 0   fuzzy    %s\n", evi+4, modelname);
           break;
        case( CERTAINTY):
           fprintf( outtba, "%2i  1  12.7 0   cert     %s\n", evi+4, modelname);
           break;
     }

     /* writes out the TBA data */
     fprintf( outtba, "DATA\n");
     for ( poly=0; poly < uni_poly; ++poly) {

        switch( modeltype) {
           case( REGRESSION):
              modelvalue = CalculateRegression( poly);
              break;
           case( BAYES):
              modelvalue = CalculateBayes( poly);
              break;
           case( DEMPSTER):
              bel = 0; pls = 0;
              CalculateDempster( poly, &bel, &pls);
              break;
           case( FUZZY):
              modelvalue = CalculateFuzzy( poly);
              break;
           case( CERTAINTY):
              modelvalue = CalculateCertainty( poly);
              break;
        }

        switch( modeltype) {
           case( DEMPSTER):
              fprintf( outtba, "%7i %7i %7i %12.7lf", poly+1,
                       ReclassifiedModelValue( bel),
                       ReclassifiedModelValue( pls),
                       unipoly[ poly].area);
              for ( evi=0; evi < no_favtables; ++evi)
                 fprintf( outtba, " %12.7lf %12.7lf",
                 bel_value( poly, evi), pls_value( poly, evi));

              fprintf( outtba, " %12.7lf %12.7lf\n", bel, pls);
              break;
```

```
         default:
            fprintf( outtba, "%7i %7i %12.7lf", poly+1,
                     ReclassifiedModelValue( modelvalue), unipoly[ poly].area);
            for ( evi=0; evi < no_favtables; ++evi) {
               switch( modeltype) {
                  case( REGRESSION):
                  case( BAYES):
                     favvalue = bayes_value( poly, evi);
                     break;
                  case( FUZZY):
                     favvalue = fuzzy_value( poly, evi);
                     break;
                  case( CERTAINTY):
                     favvalue = certfact_value( poly, evi);
                     break;
               }
               fprintf( outtba, " %12.7lf", favvalue);
            }
            fprintf( outtba, " %12.7lf\n", modelvalue);
      }
   }
   fclose( outtba);

} /* GeneratesModelTBA */


/*****************************************************************************
name: ExecuteModel( int modeltype, char *outmap1, char *outmap2)

goal: To generate a command file that the user uses to create the final
      model map.

input: modeltype : REGRESSION, BAYES, DEMPSTER, FUZZY, or CERTAINTY
       outmap1   : the name of the final model map
       outmap2   : the secondary name of the final model map(DEMPSTER only)

returns: nothing
Global variables changed: none
*****************************************************************************/
void ExecuteModel( int modeltype, char *outmap1, char *outmap2)
{
int         c=0;           /* input char from keyboard                      */
long int    wastetime;     /* counter used to waste time                    */
FILE        *execfile;     /* SPANS command file for generating model map   */
char        endmsg[ 50];   /* message displayed when the model map is done  */
char        outmap[ 13];   /* name of the model map                         */

/* SPANS command file format to generate bayes' map    */
/*     Position             Argument                   */
/*      0,10               model table                 */
/*      1,11             input unique cond. map        */
/*      1,23             resulting model  map          */
/*      1,35                 window                    */
/*      1,57               model  table                */
/*      1,79          column # of favorability value   */
/*      1,84                quad level                 */
char  build_model[ 2][ 90] = {
        "attimp f =        ",
        "ovr_tbb m =         o =         w =   c =0  b =y s =y f =         j =1 k =0 l =?
q =  "};

   /* adapting the command file to the user parameters and creating        */
   /*   the file                                                           */
```

```
switch( modeltype) {
   case( REGRESSION):
      memcpy( build_model[ 0] + 10, "reg", 3);
      memcpy( build_model[ 1] + 57, "reg", 3);
      strcpy( endmsg, "The file 'reg.tba' contains the results.");
      break;
   case( BAYES):
      memcpy( build_model[ 0] + 10, "bayes", 5);
      memcpy( build_model[ 1] + 57, "bayes", 5);
      strcpy( endmsg, "The file 'bayes.tba' contains the results.");
      break;
   case( DEMPSTER):
      memcpy( build_model[ 0] + 10, "dempster", 8);
      memcpy( build_model[ 1] + 57, "dempster", 8);
      strcpy( endmsg, "The file 'dempster.tba' contains the results.");
      break;
   case( FUZZY):
      memcpy( build_model[ 0] + 10, "fuzzy", 5);
      memcpy( build_model[ 1] + 57, "fuzzy", 5);
      strcpy( endmsg, "The file 'fuzzy.tba' contains the results.");
      break;
   case( CERTAINTY):
      memcpy( build_model[ 0] + 10, "cert", 4);
      memcpy( build_model[ 1] + 57, "cert", 4);
      strcpy( endmsg, "The file 'cert.tba' contains the results.");
      break;

}

/* filling in SPANS command file */
memcpy( build_model[ 1] + 11, "unique", 6);
memcpy( build_model[ 1] + 23, outmap1, strlen( outmap1));
memcpy( build_model[ 1] + 35, window, strlen( window));
memcpy( build_model[ 1] + 79, "2", 1);
memcpy( build_model[ 1] + 84, quadlevel, strlen( quadlevel));

/* writing out SPANS command file */
execfile = fopen( "model.cmd", "w+");
fprintf( execfile, "%s\n%s\n", build_model[ 0], build_model[ 1]);
if ( modeltype == DEMPSTER) {
   memcpy( build_model[ 1] + 23, outmap2, strlen( outmap2));
   memcpy( build_model[ 1] + 79, "3", 1);
   fprintf( execfile, "%s\n", build_model[ 1]);
   strcpy( outmap, outmap2);
}
else
   strcpy( outmap, outmap1);
fclose( execfile);

/* Generating the necessary table to attach the model result to each */
/* unique polygons                                                   */
GenerateModelTBA( modeltype);


/* tells the user what to do in SPANS */
_settextposition( 17, 2); _outtext("SPANS: exec f=model");

/* starts waiting for SPANS ... */

strcat( outmap, ".map");
remove( outmap);

while ( TRUE) {

   for ( wastetime=0; wastetime<70000; ++wastetime);
   _settextposition( 18, 2); _outtext("Waiting for SPANS ... \\");
```

```
      if ( CompletedJob( outmap))
         break;

      for ( wastetime=0; wastetime<70000; ++wastetime);
      _settextposition( 18, 2); _outtext("Waiting for SPANS ... -");

      if ( kbhit()) {
         c = getch();
         if ( c == ESC) break;
      }
      for ( wastetime=0; wastetime<70000; ++wastetime);
      _settextposition( 18, 2); _outtext("Waiting for SPANS ... /");
   }

   if ( c != ESC) {
      _settextposition( 20, 2); _outtext("Integration of data now complete.");
      _settextposition( 21, 2); _outtext( endmsg);
      _settextposition( 22, 2); _outtext("Press any key to continue...");
      while ( !kbhit());
      getch();
   }

} /* ExecuteModel() */


/*****************************************************************************
name: ModelTechnique( int modeltype)

goal: To accept the necessary parameters from the user for modeling the
      specified integration rule and to call the necessary functions for
      doing so.

input: modeltype : the modeling function (BAYES, REGRESSION, DEMPSTER,
                   FUZZY, or CERTAINTY)

returns: nothing
Global variables changed: none
*****************************************************************************/
void ModelTechnique( int modeltype)
{
int      sel;                    /* menu selection                          */
char     outmap1[ 13] = "Bayes"; /* output model map                        */
char     outmap2[ 13] = "Pls";   /* output secondary map (DEMPSTER only)    */
char     modelinfo1[ 40] = " ";  /* information displayed at the user       */
char     modelinfo2[ 40] = " ";  /* information displayed at the user       */

enum parmsChoices { OUTMAP, WINDOW, QUADLEVEL, EXECUTE, END};

   /* ask for the unit size, the system directory path, and                 */
   /*   classification scheme of the favorability index, quad tree level,  */
   /*   and default window                                                  */

   switch( modeltype) {
      case( REGRESSION):
         strcpy( outmap1, "Reg");
         strcpy( modelinfo1, "regression output map: ");
         break;
      case( BAYES):
         strcpy( outmap1, "Bayes");
         strcpy( modelinfo1, "bayes output map: ");
         break;
```

```
      case( DEMPSTER):
         strcpy( outmap1, "Bel");
         strcpy( outmap2, "Pls");
         strcpy( modelinfo1, "Belief    output map: ");
         strcpy( modelinfo2, "Plausible output map: ");
         break;
      case( FUZZY):
         strcpy( outmap1, "Fuzzy");
         strcpy( modelinfo1, "fuzzy Logic output map: ");
         break;
      case( CERTAINTY):
         strcpy( outmap1, "Cert");
         strcpy( modelinfo1, "Certainty Factor output map: ");
         break;
   }

   if ( modeltype == DEMPSTER) {
      _settextposition( 9, 2); _outtext( modelinfo1); OutBgText( outmap1);
      _settextposition(11, 2); _outtext( modelinfo2); OutBgText( outmap2);
   }
   else {
      _settextposition( 9, 2); _outtext( modelinfo1); OutBgText( outmap1);

   }
   _settextposition(13, 2); _outtext( "quad tree level: "); OutBgText( quadlevel);
   _settextposition(15, 2); _outtext( "window: "); OutBgText( window);

   sel = OUTMAP;
   while( sel != END && sel != ESC)
      switch( sel) {

         case OUTMAP:
            switch( modeltype) {
               case( DEMPSTER):
                  sel = EditStr( 9,  strlen( modelinfo1)+2, 8, validfilename, outmap1);
                  if ( sel != ESC) {
                     sel = EditStr( 11, strlen( modelinfo2)+2, 8, validfilename, outmap2);
                     _settextposition( 9, 2); _outtext( modelinfo1); OutBgText( outmap1);
                     _settextposition( 11, 2); _outtext( modelinfo2); OutBgText( outmap2);
                     sel = (sel == ESC) ? ESC : QUADLEVEL;
                  }
                  break;
               default:
                  sel = EditStr( 9, strlen( modelinfo1)+2, 8, validfilename, outmap1);
                  _settextposition( 9, 2); _outtext( modelinfo1); OutBgText( outmap1);
                  sel = (sel == ESC) ? ESC : QUADLEVEL;
            }

         case QUADLEVEL:
            sel = EditStr( 13, 17+2, 2, numbersonly, quadlevel);
            _settextposition( 13, 17+2); OutBgText( quadlevel);
            sel = (sel == ESC) ? ESC : WINDOW;
            break;

         case WINDOW:
            sel = EditStr( 15, 2+8, 2, validfilename, window);
            _settextposition( 15, 2+8); OutBgText( window);
            sel = (sel == ESC) ? ESC : EXECUTE;
            break;

         case EXECUTE:
            ExecuteModel( modeltype, outmap1, outmap2);
            sel = END;
            break;
      }
   ClearBox( 3, 1, 22, 77);
} /* ModelTechnique() */
```

```
/*****************************************************************************
name: Rules()

goal: To apply on the five integration rules (BAYES, REGRESSION, DEMPSTER,
      FUZZY, or CERTAINTY).

input: none
returns: nothing
Global variables changed: none
*****************************************************************************/
void Rules()
{
int sel = 0;                                    /* menu's selection          */

   /* the user has to select one of the integration rules                    */
   sel = MenuVerti( 3, 7, rulesmnu, sel);
   switch( sel) {
      case REGRESSION:        /* multivariate regression analysis           */
         ModelTechnique( REGRESSION);
         break;
      case BAYES:             /* Bayes rule under conditional independence */
         ModelTechnique( BAYES);
         break;
      case DEMPSTER:          /* Dempter's rule of combination              */
         ModelTechnique( DEMPSTER);
         break;
      case FUZZY:             /* fuzzy logic set                            */
         /* ModelTechnique( FUZZY); */
         break;
      case CERTAINTY:         /* certainty factor                           */
         /* ModelTechnique( CERTAINTY); */
         break;
  }

  ClearBox( 3, 5, 5, 53);

} /* Rules() */
```

UTILS.H

```
#define           LASTNUM     1
#define           FROMSTART   0
#define           FROMLAST    -1
#define           TRUE        1
#define           FALSE       0
#define           BELL        7

/* struct KEYINDEX
      This structure is use to specify the valid keys when reading a string.
      For example if you only want upper case characters:
        lowerkey = 65
        upperkey = 92
      stands for 'A' to 'Z'
*/
struct keyindex {
   int lowerkey;        /* lower ASCII number of the key        */
   int upperkey;    /* upper ASCII number of key           */
};


extern struct keyindex validpath[ 7];
extern struct keyindex validfilename[ 4];
extern struct keyindex numbersonly[ 3];

/*
if format = 1 then argument = 0x0707 (underline cursor)
   format = 0 then argument = 0x0007 (full block cursor)
*/
#define ChangeCursor( format)     _settextcursor( format*1792+7)

/* N.B. include <io.h> and <errno.h> to use EXIST */
#define  EXIST( filename)            !access( filename, 00)

int      CompletedJob( char *fn);
int      EditStr( int x, int y, int maxlength,
                  struct keyindex validkeys[], char *str);
int      GetFileNames( char *filter);
void     ProgramError( char *errmsg);
void     OutBgText( char *text);
double   StrGetLastFloat( char *instr, int postype, int pos);
double   CalculateP_D( double area, int pnts);
int      ReclassifiedModelValue( double mvalue);
```

UTILS.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

   CalculateP_D
   CompletedJob
   EditStr
   EXIST                        : macro defined in utils.h
   GetFileNames
   OutBgText
   ProgramError
   ReclassifiedModelValue
   StrGetLastFloat

MODULE INTERNAL FUNCTION(s):

   CharValid
*****************************************************************************/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <graph.h>
#include <time.h>
#include <conio.h>
#include <sys\types.h>
#include <sys\stat.h>


#include "favmod.h"
#include "utils.h"
#include "menu.h"

#define         NO_MATCH        18
#define         NO_FILES        -1

/* The variables 'validpath', 'validfilename', and 'numbersonly' are used  */
/* throughout the program in conjonction with the function 'EditStr'. This */
/* function edits a string with input a structure similar to these         */
/* variables indicating which characters to accept from the console.       */
struct keyindex validpath[ 7] =      {{ 65, 90},     /*  'A'..'Z'    */
                                     { 97,122},      /* 'a'..'z'     */
                                     { 58, 58},      /* '\'          */
                                     { 92, 92},      /* ':'          */
                                     { 46, 46},      /* '.'          */
                                     { 48, 57},      /* '0'..'9'     */
                                     { 0}};
struct keyindex validfilename[ 4] = {{ 65, 90},     /*  'A'..'Z'    */
                                     { 97,122},      /* 'a'..'z'     */
                                     { 48, 57},      /* '0'..'9'     */
                                     { 0}};
struct keyindex numbersonly[ 3]   = {{ 48, 57},     /*  '0'..'9'    */
                                     { 45, 46},      /*  '-'..'.'    */
                                     { 0}};
```

```
/*****************************************************************************
name: int ReclassifiedModelValue( double mvalue)

goal: To reclassify the model value that varies between 'favmin' and 'favmax'
      to a integer value compatible with SPANS.

input: mvalue : the model value to reclassify

returns: rcl : reclassified model value

Global variables changed: none
*****************************************************************************/
int ReclassifiedModelValue( double mvalue)
{
int   rcl;            /* counter that is the reclassified model value  */

   /* the reclassified value of 1 is a model value of 0  */
   if ( ( mvalue >= favindex[ 1].min) && ( mvalue < PRECISION))
      return( 1);
   /* loop until the model value falls between the min and max of the      */
   /*    favorability table constructed with 'favmin' and 'favmax'         */
   rcl = 2;
   while ( rcl != 254) {
      if ( ( mvalue >= favindex[ rcl].min) &&
          ( mvalue <= favindex[ rcl].max))  break;
      ++rcl;
   }
   return( rcl);

} /* ReclassifiedModelValue() */


/*****************************************************************************
name: int CompletedJob( char *fn)

goal: The function is used to check if SPANS has created the specified file.
      If the file can be renamed, the file is then assumed to be created.

input: fn : the filename of the file to check.

returns:       1 : the file can be accessed or exist
             or
             0 : the file cannot be accessed or does not exist

Global variables changed: none
*****************************************************************************/
int CompletedJob( char *fn)
{

   /* a successful 'rename' of a file means its free of access */
   remove( "temp");
   if ( !rename( fn, "temp")) {
      rename( "temp", fn);
      return( TRUE);
   }

   return( FALSE);

} /* CompletedJob() */
```

```
/*************************************************************************
name: double StrGetLastFloat( char *instr, int readtype, int pos)

goal: The function gets a floating point number in the specified
      string <instr>.

input: instr    : the string containing a floating number
       readtype : LASTNUM, FROMSTART, or FROMLAST
                depending on readtype, the function return either a
                value 'pos' from the startfrom the start
       pos      :

#define           LASTNUM      1
#define           FROMSTART    0
#define           FROMLAST    -1

output: the last floating point number in <instr>
*************************************************************************/
double StrGetLastFloat( char *instr, int readtype, int pos)
{
char  strele[ 50][80];
int   noelements;
int   indexpos;

   noelements = sscanf( instr,

       "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%
s%s%s%s%s%s%s%s",
                    strele[0], strele[1], strele[2], strele[3], strele[4],
                    strele[5], strele[6], strele[7], strele[8], strele[9],
                    strele[10], strele[11], strele[12], strele[13], strele[14],
                    strele[15], strele[16], strele[17], strele[18], strele[19],
                    strele[20], strele[21], strele[22], strele[23], strele[24],
                    strele[25], strele[26], strele[27], strele[28], strele[29],
                    strele[30], strele[31], strele[32], strele[33], strele[34],
                    strele[35], strele[36], strele[37], strele[38], strele[39],
                    strele[40], strele[41], strele[42], strele[43], strele[44],
                    strele[45], strele[46], strele[57], strele[58], strele[49]);

   indexpos = 0;
   switch( readtype) {
      case( LASTNUM):
       indexpos = noelements-1;
       if ( indexpos < 0) indexpos = 0;
       break;
      case( FROMLAST):
       indexpos = noelements-1-pos;
       if ( indexpos < 0) indexpos = 0;
       break;
      case( FROMSTART):
       indexpos = pos-1;
       if ( noelements<indexpos) indexpos = 0;
       break;
   }

   return( atof( strele[ indexpos]));
}

double CalculateP_D( double area, int pnts)
{
double   prob;
   prob = 0.0;
   if ( ( pnts != 0) && ( area != 0.0))
      prob = 1.0 - (double )powl( (long double ) (1.0 - min( atof(unitsize)/area, 1.0)),
                             (long double ) pnts);
   return( prob);
} /* CalculateP_D() */
```

```
/*.....................................................................
. ProgramError : Display a system error to the screen.  The function also aborts
.         the execution of the program.                                .
.                                                                      .
. PARAMETERS: str : string containing the error message                .
.                                                                      .
. RETURNS: nothing                                                     .
.                                                                      .
. GLOBAL CHANGES: none                                                 .
.                                                                      .
. CALLS: none                                                          .
.....................................................................*/
void ProgramError( char *errmsg)
{

   /* display a box and the error message inside it */
   Box( 13, 27 - (int )( floor( strlen( errmsg)/ 2)) - 2, 2, strlen( errmsg) + 4 );
   _settextposition( 14, 27 - (int )floor( strlen( errmsg)/2)+1);
   printf( "%c%s", BELL, errmsg);
   _settextposition( 15, 27 - 14);
   _outtext( "Press any key to continue ...");

   /* wait for any to continue and clear the box */
   while ( !kbhit()); getch();
   ClearBox( 13, 27 - (int )floor( strlen( errmsg)/2) - 2, 2, strlen( errmsg) + 4 );

   _settextposition( 1, 1);

} /* ProgramError() */


/*.....................................................................
. CharValid : Checks if the input character is a valid one.            .
.                                                                      .
. PARAMETERS: valkeys     : Possible valid keys                        .
.             carac       : the key to check                           .
.                                                                      .
. RETURNS: valid          : TRUE or FALSE depending on the validity of carac .
.                                                                      .
. GLOBAL CHANGES: none                                                 .
.....................................................................*/
int CharValid( struct keyindex valkeys[], char carac)
{
int     i       = 0;
int     valid   = FALSE;

   while ( valkeys[ i].lowerkey != (char )NULL) {

      if ( ( carac >= (char )valkeys[ i].lowerkey) &&
           ( carac <= (char )valkeys[ i].upperkey)) {
        valid = TRUE;
        break;
      }
      ++i;
   }
   return( valid);

} /* CharValid */
```

```
void OutBgText( char *text)
{
long        bgColor;            /* Screen colors */
short       fgColor;

    /* Save the colors */
   bgColor = _getbkcolor();
   fgColor = _gettextcolor();

   /* output the text with the program's colors */
   _settextcolor( mnuAtrib.fgNormal);
   _setbkcolor( mnuAtrib.bgNormal);
   _outtext( text);

   /* restore previous colors */
   _settextcolor( fgColor);
   _setbkcolor( bgColor);

} /* OutBgText() */


/*****************************************************************************
name: EditStr( x, y, maxlength, struct keyindex validkeys[], str)

goal: To get and edit a input string from the keyboard.  The function allows
      full editing with the cursor, backspace, delete, and the insert keys.

input: x          : the x position to start editing the string
       y          : the y position to start editing the string
       maxlength  : the max number of characters allowed in editing
       validkeys  : valid keys accepted as input for the string
       str        : the str to be edited

output: str       : the edited string
      keyret    : ESC or 1 (successful editing)
*****************************************************************************/
int EditStr( int y, int x, int maxlength,
            struct keyindex validkeys[], char *str)
{
int         keyret=1, len = strlen( str);
static int  insert = TRUE;
int         pos=0;
int         i=0;
short       cursor;
long        bgColor;            /* Screen color, position, and cursor   */
short       fgColor;
int         key;

    /* Save screen information. */

   _displaycursor( _GCURSORON);
   cursor = _gettextcursor();
   ChangeCursor( insert);
   bgColor = _getbkcolor();
   fgColor = _gettextcolor();
   _settextcolor( mnuAtrib.fgNormal);
   _setbkcolor( mnuAtrib.bgNormal);


   pos = strlen( str);
   key = 0;

   while( key != ESC && key != CR) {

      for ( i=strlen( str); i < maxlength; ++i)
        str[ i] = ' ';
```

```
        str[ maxlength] = 0;
        _settextposition( y, x); _outtext( str);

        _settextposition( y,pos+x);
        switch( key = GetKey( WAIT)) {
          case HOMEKEY :
             pos = 0;
             break;
          case ENDKEY :
             pos = len;
             break;
          case INSKEY :
             insert = !insert;
             ChangeCursor( insert);
             break;
          case LEFTKEY :
             if ( pos > 0) pos--;
             break;
          case RIGHTKEY :
             if ( pos < len) pos++;
             break;
          case BS :
             if ( pos > 0) {
                memmove( &str[ pos-1], &str[ pos], len - pos + 1);
                pos--;
                len--;
             }
             break;
          case DELKEY :
             if ( pos < len) {
                memmove( &str[ pos], &str[ pos+1], len - pos - 1);
                len--;
             }
             break;
          case ESC:
             keyret = ESC;
          case CR :
             break;
          default :
            if ( CharValid( validkeys, ( char)key)) {
               if ( insert && ( len < maxlength)){
                memmove( &str[pos+1], &str[ pos], len - pos + 1);
                str[ pos++] = (char )key;
                len++;
               }
               else
                if ( !insert && ( pos < maxlength)) {
                   str[ pos++] = (char )key;
                   if ( pos > len)
                      len++;
                }
            }
            break;

        } /* switch */
        str[len] = 0;

      } /* endwhile */

      _settextcursor( cursor);
      _settextcolor( fgColor);
      _setbkcolor( bgColor);
      _displaycursor( _GCURSOROFF );
      return( keyret);

    } /* EditStr() */
```

```
/*****************************************************************************
name: GetFileNames( char *filter)

goal: The function performs a OS/2 DIRECTORY command with the filter and
      stores the matching files in fnames.

input: filter: the filter used when calling OS/2 DIR command

output: fnames: the matching filenames
      GetFileNames: the function returns the # of matching files
*****************************************************************************/
int GetFileNames( char *filter)
{
char  *args[6] = {
       "CMD.exe",
       "/C",
       "DIR",
       "        ",
       ">temp",
       NULL};

char  fnstr[ 80];
FILE  *tempf;
int   i=0;
int   retcode;

   /* cleans the input variable */
   for (i=0; i<MAX_FNAMES; ++i) fnames[ i][ 0] = 0;

   /* execute the filtered DIR and store the results in file TEMP */
   args[3] = filter;

   retcode = spawnvp( P_WAIT, args[ 0], args );
   if ( retcode==NO_MATCH || retcode==NO_FILES)
      return( 0);

   /* get the filtered filenames stored in TEMP */
   if ( (tempf = fopen( "temp", "rt")) == NULL)
      ProgramError( "Can not open, read, or write temporary file.");

   else {
      /* read the info found before any files */
      do
        if ( fgets( fnstr, 80 - 1, tempf) == 0)
           ProgramError( "Can not open, read, or write temporary file.");
      while ( fnstr[ 0] == SPACE || fnstr[ 0] == NEWLN );

      /* read the filename(s) */
      i=0;
      while ( TRUE) {
        strncpy( fnames[ i], fnstr, 12);
        fnames[ i][12] = ENDSTR;
        /* fnames[ i][8] = (char )DOT; */
        ++i;
        if ( fgets( fnstr, 80 - 1, tempf) == NULL)
        if( feof( tempf ) )
           break;
        else
           ProgramError( "Can not open, read, or write temporary file.");
        if ( strncmp( fnstr, " ", 1) == (int )0) break;
      }
   }
   if ( fclose( tempf) != 0)
      ProgramError( "Can not open, read, or write temporary file.");
   fnames[ i][0] = (char )0;
   return( i);
} /* GetFileNames */
```

MENU.H

```
#ifndef MENU_H
#define MENU_H

#define TRUE  1
#define FALSE 0
/* Include only once */
#define  BS          8
#define  FORMFEED      12
#define  CR          13
#define  ESC         27
#define  HOMEKEY       327
#define  ENDKEY            335
#define  UPKEY             328
#define  DOWNKEY       336
#define  PGUPKEY       329
#define  PGDNKEY       337
#define  LEFTKEY       331
#define  INSKEY            338
#define  RIGHTKEY      333
#define  DELKEY            339
#define  CTRLLEFTKEY   371
#define  SPACE             32
#define  NEWLN             10
#define       DOT          46
#define  ENDSTR            0
#endif /* MENU_H */

/* Action codes for getkey */

enum WAITACTION { NO_WAIT, WAIT, CLEAR_WAIT};

/* Text output colors. Note that monochrome can only use 3_TBLACK,
 * _TWHITE, _TBRIGHTWHITE, and _TUNDERLINE. Graphics black-and-white
 * can only use the first three of these. The first eight colors
 * can be used as background colors (although they may need to be
 * cast to long).
 */
enum TEXTCOLORS
{
    _TBLACK,         _TBLUE,          _TGREEN,         _TCYAN,
    _TRED,           _TMAGENTA,       _TBROWN,         _TWHITE,
    _TGREY,          _TLIGHTBLUE,     _TLIGHTGREEN,    _TLIGHTCYAN,
    _TLIGHTRED,      _TLIGHTMAGENTA, _TLIGHTYELLOW,  _TBRIGHTWHITE,
    _TUNDERLINE = 1
};

/* Structure and global variable for menu attributes */
typedef struct _MENU
{
    int     fgBorder, fgNormal, fgSelect, fgNormHilite, fgSelHilite;
    int     fgNormNotValid;
    long    bgBorder, bgNormal, bgSelect, bgNormHilite, bgSelHilite;
    long    bgNormNotValid;
    int     fCentered;
    unsigned char   chNW, chNE, chSE, chSW, chNS, chEW;
} MENU;

extern MENU mnuAtrib;

/* Public menu, output, and input functions and macros */

/* Structure and maximum length for menu items */
#define MAXITEM 80
```

```
typedef struct _ITEM
{
    int     iHilite;
    char    achItem[MAXITEM];
} ITEM;

int MenuHori( int row, int col, ITEM aItem[], int iCur );
int MenuVerti( int row, int col, ITEM aItem[], int iCur );
void Box( int row, int col, int rowLast, int colLast );
void ClearBox( int row, int col, int rowLast, int colLast );
unsigned GetKey( int fWait );
void _outchar( char out );
int SelectOneFileMenu( int row, int col, int nofiles);
void Itemize2( int row, int col, int fCur, char *itm);
```

MENU.C

```
/*****************************************************************************
MODULE EXTERNAL AVAILABLE FUNCTION(s):

  _outchar
  Box
  ClearBox
  GetKey
  Itemize2
  MenuHori
  MenuVerti
  SelectOneFileMenu

MODULE INTERNAL FUNCTION(s):

  Itemize
*****************************************************************************/
#include <string.h>
#include <stddef.h>
#include <ctype.h>
#include <graph.h>
#include <bios.h>

#include <conio.h>

#include <math.h>
#include <stdio.h>
#include <process.h>
#include <errno.h>

#include "menu.h"
#include "favmod.h"
```

```
/* Prototype for internal function */
static void Itemize( int row, int col, int fCur, ITEM itm, int cBlank);

/* Default menu attribute. The default works for color or B&W. You can
 * override the default value by defining your own MENU variable and
 * assigning it to mnuAtrib, or you can modify specific fields at
 * run time. For example, you could use a different attribute for color
 * than for black and white.
 */
MENU mnuAtrib =
{
    _TBLACK, _TBLACK, _TWHITE, _TBRIGHTWHITE, _TBRIGHTWHITE,
    _TWHITE, _TWHITE, _TBLACK, _TWHITE, _TBLACK,
    TRUE,
    '+', '+', '+', '+', ' ', ' '
};


/* MenuHori - Puts a horizontal menu on screen and reads menu input from keyboard. When a
 * highlighted hot key or ENTER is pressed, returns the index of the
 * selected menu item.
 *
 * Params: row and col - If "fCentered" attribute of "mnuAtrib" is true,
 *           center row and column of menu; otherwise top left of menu
 *         aItem - array of structure containing the text of each item
 *           and the index of the highlighted hot key
 *         iCur - index of the current selection--pass 0 for first item,
 *           or maintain a static value
 *
 * Return: The index of the selected item
 *
 * Uses:   mnuAtrib
 */
int MenuHori( int row, int col, ITEM aItem[], int iCur )
{
    int cItem, cchItem = 2; /* Counts of items and chars per item       */
    int i, iPrev;           /* Indexes - temporary and previous         */
    int acchItem[MAXITEM];  /* Array of counts of character in items    */
    char *pchT;             /* Temporary character pointer              */
    char achHilite[36];     /* Array for highlight characters           */
    unsigned uKey;          /* Unsigned key code                        */
    long bgColor;           /* Screen color, position, and cursor       */
    short fgColor;
    struct rccoord rc;
    unsigned fCursor;

    /* Save screen information. */
    fCursor = _displaycursor( _GCURSOROFF );
    bgColor = _getbkcolor();
    fgColor = _gettextcolor();
    rc = _gettextposition();

    /* Count items, find longest, and put count of each in array. Also,
     * put the highlighted character from each in a string.
     */
    for( cItem = 0; aItem[cItem].achItem[0]; cItem++ )
    {
       acchItem[cItem] = strlen( aItem[cItem].achItem );
       cchItem = (acchItem[cItem] > cchItem) ? acchItem[cItem] : cchItem;
       i = aItem[cItem].iHilite;
       achHilite[cItem] = aItem[cItem].achItem[i];
    }
    cchItem += 2;
    achHilite[cItem] = 0;          /* Null-terminate and lowercase string  */
    strlwr( achHilite );
```

```
    /* Adjust if centered, and draw menu box. */
    if( mnuAtrib.fCentered )
    {
       row -= cItem / 2;
       col -= cchItem / 2;
    }

    /* Put items on menu. */
    for( i = 0; i < cItem; i++ )
    {
       if( i == iCur )
           Itemize( row, col+(cchItem*i), TRUE, aItem[i], cchItem - acchItem[i] );
       else
           Itemize( row, col+(cchItem*i), FALSE, aItem[i], cchItem - acchItem[i] );
    }

    while( TRUE)
    {
       /* Wait until a uKey is pressed, then evaluate it. */
       uKey = GetKey( WAIT );
       switch( uKey )
       {
           case LEFTKEY:                              /* Right key             */
              iPrev = iCur;
              iCur = (iCur > 0) ? (--iCur % cItem) : cItem - 1;
              break;
           case RIGHTKEY:                             /* Left key */
              iPrev = iCur;
              iCur = (iCur < cItem) ? (++iCur % cItem) : 0;
              break;
           /*case ESC:
              return ESC;*/
           default:
              if( uKey > 256 )             /* Ignore unknown function key  */
                  continue;
              pchT = strchr( achHilite, (char)tolower( uKey ) );
              if( pchT != NULL )           /* If in highlight string,      */
                  iCur = pchT - achHilite;/*   evaluate and fall through  */
              else
                  continue;                /* Ignore unknown ASCII key     */
           case CR:
              _setbkcolor( bgColor );
              _settextcolor( fgColor );
              _settextposition( rc.row, rc.col );
              _displaycursor( fCursor );
              return( iCur);
       }
       /* Redisplay current and previous. */
       Itemize( row, col+(cchItem*iCur),
              TRUE, aItem[iCur], cchItem - acchItem[iCur] );
       Itemize( row, col+(cchItem*iPrev),
              FALSE, aItem[iPrev], cchItem - acchItem[iPrev] );
    }
} /* MenuHori() */
```

```
/* MenuVerti- Puts a vertical menu on screen and reads menu input from keyboard. When a
 * highlighted hot key or ENTER is pressed, returns the index of the
 * selected menu item.
 *
 * Params: row and col - If "fCentered" attribute of "mnuAtrib" is true,
 *           center row and column of menu; otherwise top left of menu
 *         aItem - array of structure containing the text of each item
 *           and the index of the highlighted hot key
 *         iCur - index of the current selection--pass 0 for first item,
 *           or maintain a static value
 *
 * Return: The index of the selected item
 *
 * Uses:   mnuAtrib
 */
int MenuVerti( int row, int col, ITEM aItem[], int iCur )
{
int cItem, cchItem = 2; /* Counts of items and chars per item        */
int i, iPrev;           /* Indexes - temporary and previous          */
int acchItem[MAXITEM];  /* Array of counts of character in items     */
char *pchT;             /* Temporary character pointer               */
char achHilite[36];     /* Array for highlight characters            */
unsigned uKey;          /* Unsigned key code                         */
long bgColor;           /* Screen color, position, and cursor        */
short fgColor;
struct rccoord rc;
unsigned fCursor;

    /* Save screen information. */

    fCursor = _displaycursor( _GCURSOROFF );
    bgColor = _getbkcolor();
    fgColor = _gettextcolor();
    rc = _gettextposition();

    /* Count items, find longest, and put count of each in array. Also,
     * put the highlighted character from each in a string.
     */
    for( cItem = 0; aItem[cItem].achItem[0]; cItem++ )
    {
       acchItem[cItem] = strlen( aItem[cItem].achItem );
       cchItem = (acchItem[cItem] > cchItem) ? acchItem[cItem] : cchItem;
       i = aItem[cItem].iHilite;
       achHilite[cItem] = aItem[cItem].achItem[i];
    }
    cchItem += 2;
    achHilite[cItem] = 0;          /* Null-terminate and lowercase string  */
    strlwr( achHilite );

    /* Adjust if centered, and draw menu box. */
    if( mnuAtrib.fCentered )
    {
       row -= cItem / 2;
       col -= cchItem / 2;
    }

    /* Put items on menu. */
    for( i = 0; i < cItem; i++ )
    {
       if( i == iCur )
           Itemize( row + i, col, TRUE, aItem[i], cchItem - acchItem[i]);
       else
           Itemize( row + i, col, FALSE, aItem[i], cchItem - acchItem[i]);
    }
```

```
    while( TRUE)
    {
       /* Wait until a uKey is pressed, then evaluate it. */
       uKey = GetKey( WAIT);
       switch( uKey )
       {
           case UPKEY:                          /* Up key       */
              iPrev = iCur;
              iCur = (iCur > 0) ? (--iCur % cItem) : cItem - 1;
              break;
           case DOWNKEY:                        /* Down key     */
              iPrev = iCur;
              iCur = (iCur < cItem) ? (++iCur % cItem) : 0;
              break;
           case ESC:
              return( ESC);
           default:
              if( uKey > 256 )              /* Ignore unknown function key  */
                  continue;
              pchT = strchr( achHilite, (char)tolower( uKey ) );
              if( pchT != NULL )            /* If in highlight string,      */
                  iCur = pchT - achHilite;/*   evaluate and fall through  */
              else
                  continue;                 /* Ignore unknown ASCII key     */
           case CR:
              _setbkcolor( bgColor );
              _settextcolor( fgColor );
              _settextposition( rc.row, rc.col );
              _displaycursor( fCursor );
              return( iCur);
       }
       /* Redisplay current and previous. */
       Itemize( row + iCur, col,
              TRUE, aItem[iCur], cchItem - acchItem[iCur] );
       Itemize( row + iPrev, col,
              FALSE, aItem[iPrev], cchItem - acchItem[iPrev] );
    }
} /* MenuVerti */


/* ClearBox - Draws a blank box with no borders, filling interior with blanks of the border
color.
 *
 * Params: row and col - upper left of box
 *         rowLast and colLast - height and width
 *
 * Return: None
 *
 * Uses:   mnuAtrib
 */
void ClearBox( int row, int col, int rowLast, int colLast )
{
    int i;
    char achT[ 81];                 /* Temporary array of characters */

    /* Set color and position. */
    _settextposition( row, col );
    _settextcolor( mnuAtrib.fgBorder );
    _setbkcolor( mnuAtrib.bgBorder );

    /* Draw box top. */
    achT[0] = ' ';
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = ' ';
    achT[colLast + 2] = 0;

    _outtext( achT );
```

```
    /* Draw box sides and center. */
    achT[0] = ' ';
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = ' ';
    achT[colLast + 2] = 0;
    for( i = 1; i <= rowLast; ++i )
    {
       _settextposition( row + i, col );
       _outtext( achT );
    }

    /* Draw box bottom. */
    _settextposition( row + rowLast + 1, col );
    achT[0] = ' ';
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = ' ';
    achT[colLast + 2] = 0;
    _outtext( achT );

} /* ClearBox() */

/* Box - Draw menu box, filling interior with blanks of the border color.
 *
 * Params: row and col - upper left of box
 *         rowLast and colLast - height and width
 *
 * Return: None
 *
 * Uses:   mnuAtrib
 */
void Box( int row, int col, int rowLast, int colLast )
{
    int i;
    char achT[MAXITEM + 2];          /* Temporary array of characters */

    /* Set color and position. */
    _settextposition( row, col );
    _settextcolor( mnuAtrib.fgBorder );
    _setbkcolor( mnuAtrib.bgBorder );

    /* Draw box top. */
    achT[0] = mnuAtrib.chNW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chNE;
    achT[colLast + 2] = 0;

    _outtext( achT );

    /* Draw box sides and center. */
    achT[0] = mnuAtrib.chNS;
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = mnuAtrib.chNS;
    achT[colLast + 2] = 0;
    for( i = 1; i <= rowLast; ++i )
    {
       _settextposition( row + i, col );
       _outtext( achT );
    }
    /* Draw box bottom. */
    _settextposition( row + rowLast + 1, col );
    achT[0] = mnuAtrib.chSW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chSE;
    achT[colLast + 2] = 0;
    _outtext( achT );
}
```

```
/* Itemize - Display one selection (item) of a menu. This function
 * is normally only used internally by MenuHori and MenuVerti.
 *
 * Params: row and col - top left of menu
 *         fCur - flag set if item is current selection
 *         itm - structure containing item text and index of highlight
 *         cBlank - count of blanks to fill
 *
 * Return: none
 *
 * Uses:   mnuAtrib
 */
void Itemize( int row, int col, int fCur, ITEM itm, int cBlank)
{
    int i;
    char achT[MAXITEM];             /* Temporary array of characters */

    /* Set text position and color. */
    _settextposition( row, col );
    if ( fCur) {
       _settextcolor( mnuAtrib.fgSelect );
       _setbkcolor( mnuAtrib.bgSelect );
    }
    else {
     _settextcolor( mnuAtrib.fgNormal );
     _setbkcolor( mnuAtrib.bgNormal );
    }

    /* Display item and fill blanks. */
    strcat( strcpy( achT, " " ), itm.achItem );
    _outtext( achT );
    memset( achT, ' ', cBlank-- );
    achT[cBlank] = 0;
    _outtext( achT );

    /* Set position and color of highlight character, then display it. */
    i = itm.iHilite;
    _settextposition( row, col + i + 1 );
    if( fCur)
    {
       _settextcolor( mnuAtrib.fgSelHilite );
       _setbkcolor( mnuAtrib.bgSelHilite );
    }
    else
    {
       _settextcolor( mnuAtrib.fgNormHilite );
       _setbkcolor( mnuAtrib.bgNormHilite );
    }
    _outchar( itm.achItem[i] );
}
```

```
/* GetKey - Gets a key from the keyboard. This routine distinguishes
 * between ASCII keys and function or control keys with different shift
 * states. It also accepts a flag to return immediately if no key is
 * available.
 *
 * Params: fWait - Code to indicate how to handle keyboard buffer:
 *   NO_WAIT     Return 0 if no key in buffer, else return key
 *   WAIT        Return first key if available, else wait for key
 *   CLEAR_WAIT  Throw away any key in buffer and wait for new key
 *
 * Return: One of the following:
 *
 *   Keytype                                  High Byte    Low Byte
 *   -------                                  ---------    --------
 *   No key available (only with NO_WAIT)        0            0
 *   ASCII value                                 0         ASCII code
 *   Unshifted function or keypad                1         scan code
 *   Shifted function or keypad                  2         scan code
 *   CTRL function or keypad                     3         scan code
 *   ALT function or keypad                      4         scan code
 *
 * Note:   getkey cannot return codes for keys not recognized by BIOS
 *         int 16, such as the CTRL-UP or the 5 key on the numeric keypad.
 */
unsigned GetKey( int fWait)
{
unsigned uKey=0;

    /* If CLEAR_WAIT, drain the keyboard buffer. */

    switch( fWait){
       case WAIT:
           while( !kbhit( ));
           uKey = getch();
           break;
       case NO_WAIT:
           if ( kbhit())
              uKey = getch();
           else
              uKey = 0;
           break;
       case CLEAR_WAIT:
           getch();
           while( !kbhit() );
           uKey = getch();
           break;
    }
    if ( kbhit( ))
       uKey = getch() + 0x0100;

    return( uKey);

} /* GeyKey */
```

```
/* _outchar - Display a character. This is the character equivalent of
 * _outtext. It is affected by _settextposition, _settextcolor, and
 * _setbkcolor. It should not be used in loops. Build strings and then
 * _outtext to show multiple characters.
 *
 * Params: out - character to be displayed
 *
 * Return: none
 */
void _outchar( char out )
{
    static char achT[2] = " ";      /* Temporary array of characters */

    achT[0] = out;
    _outtext( achT );
} /* _outchar */


/*****************************************************************************
name: Itemize2( int row, int col, int fCur, char *itm)

goal: To display (hilighted when FCUR=TRUE, or not) the menu item ITM
      at coordinate (ROW, COL).

input: row:    row to display menu item
       col:    column to display the menu item
       fCur:   TRUE:  item is hilighted
             FALSE: item is not hilighted
       itm:    item to display

output: none
*****************************************************************************/
void Itemize2( int row, int col, int fCur, char *itm)
{
char achT[MAXITEM];             /* Temporary array of characters */

    /* Set text position and color. */

    _settextposition( row, col );
    if( fCur ) {
       _settextcolor( mnuAtrib.fgSelect );
       _setbkcolor( mnuAtrib.bgSelect );
    }
    else {
       _settextcolor( mnuAtrib.fgNormal );
       _setbkcolor( mnuAtrib.bgNormal );
    }

    /* Display item and fill blanks. */
    if ( itm[ 0] == (char )NULL) strcpy( itm, "           ");
    strcat( strcpy( achT, " " ), itm );
    strcat( achT, " " );
    _outtext( achT );

} /* Itemize2() */
```

```
/*****************************************************************************
name: int SelectOneFileMenu( int row,  int col, int nofiles, char fnames[][13])

goal: Displays a menu at coordinates (ROW, COL) aimed at selecting one file
      in the list FNAMES composed of NOFILES files.

input: row:              starting row of the menu
       col:              starting column of the menu
       nofiles:          the number of filenames contained in FNAMES
       GLOBAL fnames:    the filenames stored as ASCII in a list

return: iCur: the coordinate of the selected file in FNAMES or
              the ESC code indicating no selection
*****************************************************************************/
int SelectOneFileMenu( int row,  int col, int nofiles)
{
int      iPrev=0,iCur=0;/* counter tracking current and previous cursor pos */
int      i;             /* utility counter */
int      noitem;        /* maximum number of items displayed at one time */
int      starti;        /* first top menu itme to be displayed */
int      prevstarti=0;  /* previous top menu item to be displayed */
unsigned uKey;          /* Unsigned key code                      */
long     bgColor;       /* Screen color, position, and cursor   */
short    fgColor;
struct   rccoord rc;
unsigned fCursor;


    /* Save screen information. */
    fCursor = _displaycursor( _GCURSOROFF );
    bgColor = _getbkcolor();
    fgColor = _gettextcolor();
    rc = _gettextposition();


    /* determine the maximum number of item to displayed */
    if ( nofiles > ( 23 - row)) noitem  = 23-row;
    else                        noitem  = nofiles;

    Box( row++, col++, noitem, strlen( fnames[1])+2);

    /* Put items on menu. */
    Itemize2( row, col, TRUE, fnames[ 0]);
    for( i = 1; i < noitem; i++ )
       Itemize2( row + i, col, FALSE, fnames[i]);

    /* displays  "MORE..." to indicate more files for selection */
    _settextcolor( mnuAtrib.fgBorder);
    _setbkcolor( mnuAtrib.bgBorder);
    if ( noitem != nofiles) {
       _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
       _outtext( "more...");
    }
    else {
       _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
       _outtext( "       ");
    }
```

```
    /* while a selection or ESC key do ... */
    while( TRUE)
    {
       /* Wait until a uKey is pressed, then evaluate it. */
       uKey = GetKey( WAIT );
       switch( uKey )
       {
           case UPKEY:                /* Up key       */
              iPrev = iCur;
              iCur = (iCur > 0) ? (--iCur % nofiles) : nofiles - 1;
              break;
           case DOWNKEY:                    /* Down key     */
              iPrev = iCur;
              iCur = (iCur < nofiles) ? (++iCur % nofiles) : 0;
              break;
           case ESC:
              return( ESC);
           case CR:
              _setbkcolor( bgColor );
              _settextcolor( fgColor );
              _settextposition( rc.row, rc.col );
              _displaycursor( fCursor );
              return( iCur);
           default:
              iPrev = iCur;
       }
       /* Redisplay current and previous menu item. */

       starti = noitem * ( int)floor( iCur/ noitem);

       if ( prevstarti == starti)
           /* de-hilights the appropriate selection */
           Itemize2( row + iPrev -( int)floor( iPrev / noitem) * noitem,
                   col, FALSE, fnames[ iPrev]);
       else {
           prevstarti = starti;

           /* displays "MORE..." to indicate more files for selection */
           _settextcolor( mnuAtrib.fgBorder);
           _setbkcolor( mnuAtrib.bgBorder);
           if ( ( nofiles-starti) < noitem) {
              _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
              _outtext( "       ");
           }
           else {
              _settextposition( row + noitem, col+ strlen( fnames[ 0])/2);
              _outtext( "more...");
           }

           /* Put items on menu. */
           for( i = 0; i < noitem; i++ ) {
              Itemize2( row + i, col, FALSE, fnames[ starti + i]);
           }
       }
       /* hilights the appropriate selection */
       Itemize2( row + iCur -( int)floor( iCur / noitem) * noitem,
               col, TRUE, fnames[ iCur]);

    } /* end while */

} /* SelectOneFileMenu() */
```