USING PREFERENCE/UTILITY CURVES IN THE CREATION OF A COMPUTER PROGRAM FOR DECISION EVALUATION DISPLAY ANALYSIS

by

Pamela F. Rice

Major paper submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING

in

Systems Engineering

APPROVED:

W.J. Fabrycky, Chairman

S.F. Midkiff

B.S. Blanchard, Jr.

December 1991
Blacksburg, Virginia

USING PREFERENCE/UTILITY CURVES IN
THE CREATION OF A COMPUTER PROGRAM
FOR DECISION EVALUATION DISPLAY
ANALYSIS

Acknowledgements

I would like to thank my husband, Ken, for his help and support during the computer program development and report writing. He also wrote the software routines in the Infinity Library, which were invaluable in the development of the UDED computer program.

USING PREFERENCE/UTILITY CURVES IN
THE CREATION OF A COMPUTER PROGRAM
FOR DECISION EVALUATION DISPLAY
ANALYSIS

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

Table of Contents (continued)

USING PREFERENCE/UTILITY CURVES IN
THE CREATION OF A COMPUTER PROGRAM
FOR DECISION EVALUATION DISPLAY
ANALYSIS

List of Figures

USING PREFERENCE/UTILITY CURVES IN THE CREATION OF A COMPUTER PROGRAM FOR DECISION EVALUATION DISPLAY ANALYSIS

SECTION 1.0 - INTRODUCTION

SECTION 1.1 - BACKGROUND/MOTIVATION

During the course of a life-cycle cost (LCC) analysis, many decisions must be made. The different phases of the program must be broken out e.g., acquisition, production, operations, maintenance, and disposal/retirement, then the appropriate cost and multifactor parameters must be defined. Cost parameters include research and development, production and construction, operation and support, and retirement/disposal. Multifactor parameters include mean time to repair, speed, weight, and many others. One way for the analyst to obtain an understanding of the project's or program's non-cost performance against its life-cycle cost is to use the decision evaluation display (DED). The DED (Figure 1) uses life-cycle cost as the x-axis and the multifactor criteria as the y-axes. Each alternative has a life-cycle cost line drawn with its multifactor criteria noted. The DED allows the analyst insight into the performance or effectiveness indicators (usually non-cost parameters) of various alternatives against their LCC.[1]

After working with life-cycle cost analysis and decision evaluation displays (DEDs) in both ENGR 5004, Systems Engineering and ISE 5434, Economic Evaluation of Industrial Projects, I became frustrated with the lack of a formal method for evaluating mutually exclusive alternatives which were close in both their life-cycle costs and in their effectiveness measures (performance indicators). Selecting the final alternative almost always fell back to the "gut feel" method. While taking ISE 5144, Performance and Productivity Measurement and Evaluation, I became acquainted with a method for dealing with multifactor criteria which seemed to supply the means for analytically dealing with the multifactor criteria of the DED. This method was called the multicriteria

[1]Blanchard, Benjamin S., and Wolter J. Fabrycky, Systems Engineering and Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1990.

performance/productivity measurement technique (MCP/PMT). Thus, I chose as the basis for my project and report to write a program in "C" which allows the analyst to enter the basic data for a standard decision evaluation display and then produce a utility curve based DED. Hopefully, this program will help other students avoid the same type of frustration I encountered.

SECTION 1.2 - REPORT DESCRIPTION

This report is divided into three parts: the body, a program user's guide (Appendix A), and a programmer's guide (Appendix B). The body describes the background and motivation behind choosing this particular project, the basic theoretical concepts involved, a general program description, advantages and disadvantages of using the program, suggestions for possible enhancements to the program, conclusions, and a reference list. The user's guide provides a detailed description of the program's user interface, the calculations performed, the program's capabilities, and the available report formats. An example problem is also included. The programmer's guide which is provided to help individuals who may wish to modify the computer program describes the "C" program, the basic programming philosophy, provides a data dictionary, and contains the source code listings for the program.

SECTION 2.0 - THEORETICAL CONCEPTS

SECTION 2.1 - LIFE-CYCLE COST

Life-cycle cost is a concept which over the past 10-15 years has gained attention and respect in both the Department of Defense and commercial marketplaces. Life-cycle cost tries to identify the total cost of a project or product from its inception through usage, repair, and finally disposal. The purpose of evaluating LCC during design is to minimize the total project/product cost by identifying potential problems in the design phase where they are less costly to fix.

There are three important concepts used in life-cycle cost analysis: time value of money, earning power of money, and purchasing power of money. The time value of money describes the relationship between interest rate and time. Money can earn interest at a certain rate through its investment for a given period of time; therefore, a dollar

received at a future date is not worth as much as a current dollar. Money also has earning power. A dollar invested today will have a greater worth than a dollar a given number of years from now. Since money has a value which differs with time, equal dollar amounts have different values at different points in time. Money also has purchasing power which is dependent on the inflation rate and varies with time. In times of high inflation, money has less purchasing power.[2]

To account for the variations in the worth or cost of a dollar, an extensive set of equations has been developed to allow an analyst to determine the equivalency of different sums of money at different times with differing interest rates. In most cases, a project or product being evaluated will have numerous money flows with various interest rates; reducing them to a common basis is critical in order to compare various alternatives.

There are three factors which relate to the equivalence of sums of money: the sum amounts, the time when the sums occur, and the interest rate. Since interest formulas take into account time and the interest rate, they are useful conversion factors for calculating the equivalence of various sums occurring at different points in time.[3]

SECTION 2.2 - THE DECISION EVALUATION DISPLAY

SECTION 2.2.1 - SETTING UP THE DISPLAY

To set up the DED, the analyst must have identified several optimized mutually exclusive alternatives with their associated equivalent LCCs and multifactor (performance indicator) values.

A mutually exclusive alternative is one in which the acceptance of it will preclude the acceptance of any other alternative. These types of alternatives generally exist when there are a number of alternatives which will fulfill a given need. In addition, when

---

[2] Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

[3] Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

reviewing the list of alternatives, the analyst must identify those alternatives which are contingent upon the acceptance of other alternatives. In that case, those contingent alternatives must be taken out of consideration until or if the alternative upon which they are contingent is selected. [4]

The second step in setting up the DED requires the analyst to calculate the equivalent LCC for each alternative based on its respective money flow. The analyst must then determine if any of the equivalent LCCs exceed a given monetary limitation (if there is one). If any of them do, they must be removed from consideration.

The third step is to identify the desired performance indicators and their values for each of the alternatives. In many cases, a minimum or maximum level for each performance indicator will be defined. These levels usually refer back to some identifiable need or requirement. In the case of derived requirements, the analyst will need to employ some type of iterative method to determine the sensitivity of each of the alternatives for the given parameter.

Once the analyst has identified the final set of mutually exclusive alternatives, calculated their equivalent LCCs, and determined their performance indicator values and any hard requirement levels, a DED can be created to assist the analyst in making a final selection.

SECTION 2.2.2 - USING THE DED

The DED (Figure 1) uses as its x-axis an equivalent life-cycle cost. The analyst can use present equivalent worth/cost, annual equivalent worth/cost, or future equivalent worth/cost. However, one of them must be selected and used for all of the mutually exclusive alternatives to be compared. The y-axes are the effectiveness or performance measures (indicators) which the analyst has selected. These performance indicators are often non-monetary, but design to cost or investment cost could also be used.

---

[4] Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

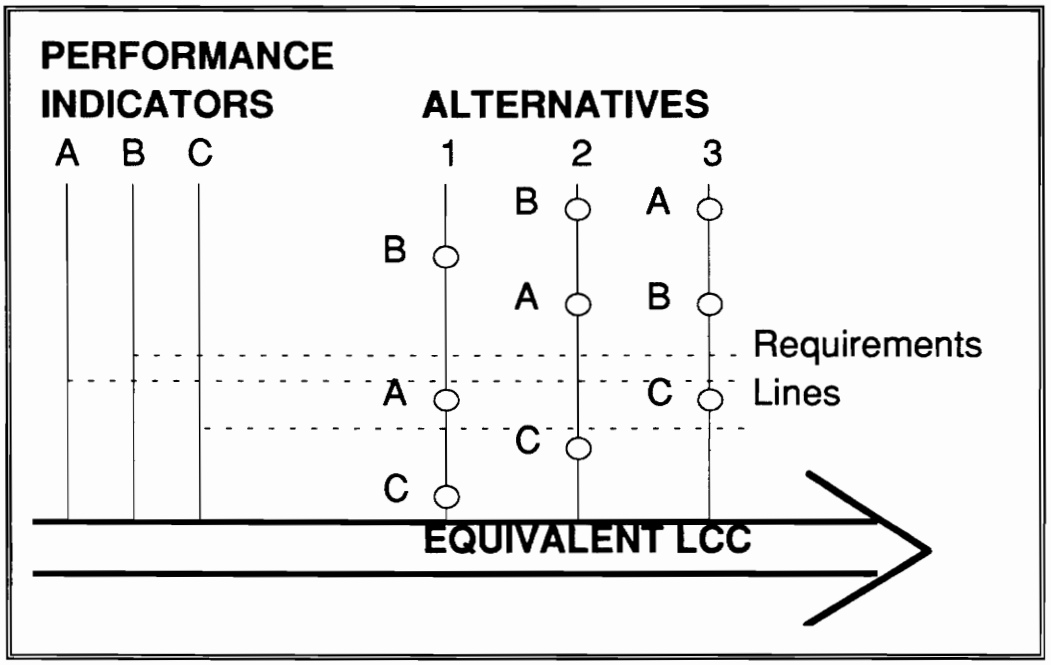

Figure 1. A Standard Decision Evaluation Display[5]

[5]Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

In the DED, the values for the various alternatives are plotted for the performance indicators along a vertical line which matches the equivalent LCC for that alternative. In he best case scenario, one alternative would provide the most desirable values for the performance indicators with the lowest equivalent LCC. This seldom occurs. Usually, the number of alternatives can be reduced to just a few, but then the analyst must make a "gut feel" decision on which one of the remaining is the "best." In addition, there is no way to formally weight the performance indicators to help the analyst determine which alternative provides the best multifactor performance.

SECTION 2.3 - THE UTILITY DECISION EVALUATION DISPLAY

SECTION 2.3.1 - UDED DESCRIPTION

A utility decision evaluation display (UDED) maps the multiple y-axes (performance indicators) of the DED onto a single utility scale axis, thus allowing indicators with different measurement units to be evaluated on a common denominator basis. This allows the analyst to make a more meaningful comparison between two indicators which could have very different measurement units. Figure 2 shows an example of a generic utility DED.

SECTION 2.3.2 - MULTICRITERIA PERFORMANCE/PRODUCTIVITY MEASUREMENT TECHNIQUE (MCP/PMT)

The multicriteria performance/productivity measurement technique is normally used as an integral component of the normative productivity measurement methodology (NPMM). NPMM is a tool used in designing, developing, and implementing productivity/performance measurement systems in a participative manner. The MCP/PMT is used to aggregate unlike productivity measures using a utility scale for each productivity/performance indicator. The utility scale provides a common denominator for the multifactor criteria. Figure 3 shows an example of a MCP/PMT worksheet.

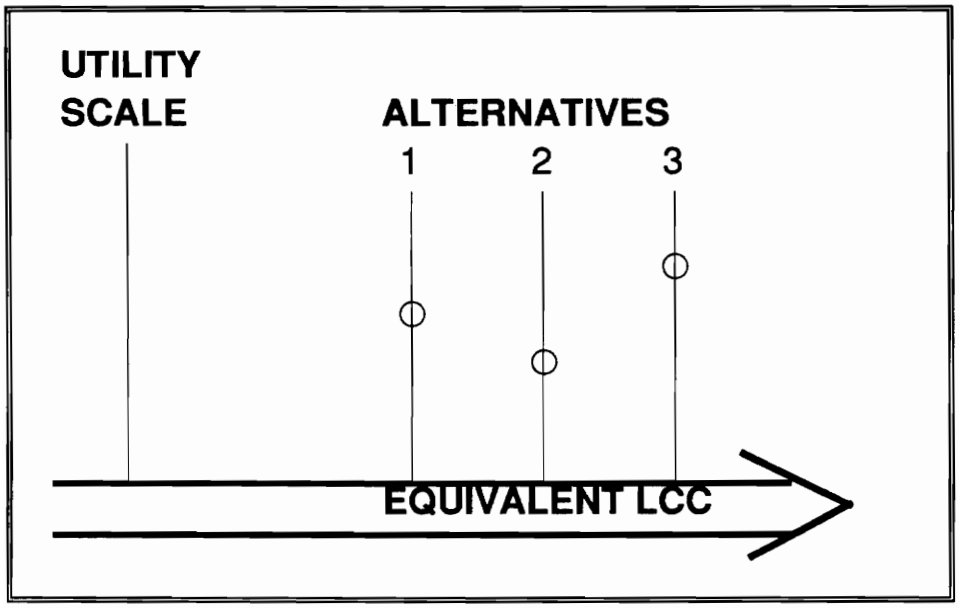

Figure 2. A Utility Decision Evaluation Display

Effectiveness	Efficiency	Quality	Productivity	Performance Criteria
Accom-plishments	as a %	% defects	O/I Index	Measures, Ratios, Indices, Objectives
5	90%	0.025	0.85	Act. Perf. this Period
5	100%	0	1.3	10
				9
		0.01	1.2	8
3	90%			7
		0.02	1.15	6
	80%			5
2			1.1	4
	75%		1	3
1		0.03		2
			0.95	1
0	60%	0.05	0.9	0
10	7	4	0	← Equivalent Score
30	5	30	10	← Subjective Weightings
300	35	120	0	Sum to get Performance Index= 455

Performance Scores

X

=

Figure 3. An Example of a MCP/PMT Worksheet[6]

[6] Sink, D. Scott, Productivity Management: Planning, Measurement and Evaluation, Control and Improvement. New York: John Wiley & Sons, Inc., 1985, p. 202.

The values for 0, 5, and 10 on the utility scale have special meaning. The level 0 represents the lowest level of productivity or performance possible for a specific productivity/performance indicator. The level 5 represents an acceptable level of productivity or performance, and the level 10 represents the best possible level (excellence). The actual utility curves are then determined based on the desired values for 0, 5, and 10 and on a group consensus regarding the proper curve shape.

A list of productivity/performance indicators is generally identified through the use of the nominal group technique (NGT) which is a participative, round robin, brain storming technique. The list of productivity/performance indicators must then be ranked based on their importance. The most important indicator is ranked as number one (1).

A more precise measurement of the importance of an indicator is its rating. The rating value can range from 100 for the most important indicators to 1 for the least important. If 0 were used, the indicator would not be considered at all. A productivity/performance indicator weighting is then calculated based on the given indicator's rank divided by the total for all of the productivity/performance indicator rankings.[7]

Productivity or performance for a specified period is calculated by entering the values for the productivity/performance indicators for that period, determining the utility value for each, multiplying by the productivity/performance indicator weighting values to obtain a weighted value for each indicator, and then totaling the values. The totalled value can then be compared on an aggregate scale to determine the overall productivity or performance for the period.

SECTION 2.3.3 - TAILORING MCP/PMT

The same basic process described above was used in the development of the utility decision evaluation display computer program with the following exceptions. For the UDED, the emphasis is on performance rather than on productivity, thus for the remainder of this paper, the phrase performance indicator will be used. The curve fit type is not controllable by the analyst. A second or first order least squares curve fit is implemented by the computer program to determine the equation for each of the perfor-

[7] Sink, D. Scott, Productivity Management: Planning, Measurement and Evaluation, Control and Improvement. New York: John Wiley & Sons, Inc., 1985.

mance indicator utility curves based on the values entered for the benchmarks of 0, 5, and 10 on the utility scale. The utility values for the performance indicators are then calculated for each of the alternatives from these equations.

SECTION 3.0 - THE UTILITY DECISION EVALUATION DISPLAY PROGRAM

SECTION 3.1 - GENERAL DESCRIPTION

The utility decision evaluation display computer program is written in the "C" language. It is made up of six basic modules: DEDMAIN, PERF, ALT, CALC, REPORT, and SETUP. Module names will appear in capital letters throughout this paper; structure names will be in italics.

SECTION 3.2 - MODULE DESCRIPTIONS

DEDMAIN is the main module from which all of the other modules are accessed. It sets up the main menu, and loads any existing data into PERF and ALT.

PERF is typically the first module into which data would be entered. This module allows the analyst to add the performance indicator name, rank, rating, and benchmark values (0, 5, 10) which are based on the tailoring of the MCP/PMT process. The analyst can also edit, delete, or save the performance indicator data.

ALT is the module where the data for the mutually exclusive alternatives is entered. The analyst can add the alternative name, life-cycle cost, and the performance indicator values for the given alternative. The analyst can also edit, delete, or save the alternatives data.

CALC is the module where all of the calculations or conversions are performed. The analyst has the option of allowing alternatives which do not meet all of the multi-criteria requirements (utility curve value 5) to be processed through the utility conversion to obtain a weighted score or to have those alternatives removed from consideration. The CALC module calculates the weighting for each performance indicator, converts the

values for each alternative's performance criteria into utility values, calculates a weighted score for each performance indicator for each alternative, and finally calculates the weighted index for each alternative. The module saves the calculation data automatically.

REPORT is the module where the analyst can view on screen or print out the data entered or the results of the calculations. The report types include: performance indicators, alternatives, alternatives' utility values, and weighted score report.

The final module is SETUP. SETUP allows the analyst to enter the file name to be used in the other modules. It also allows the user to define the type of LCC which will be entered e.g., present equivalent worth (PEW) or annual equivalent worth (AEW).

There are numerous cross-checks between modules to verify data integrity. For example, if a performance indicator is deleted after an alternative's data has been entered, that same indicator values will be deleted out of the alternative's data.

For a more detailed description of actual program usage, please refer to Appendix A, the User's Guide. For more information regarding the program code or data definitions, please refer to Appendix B, the Programmer's Guide.

SECTION 3.3 - ADVANTAGES OF THE UDED COMPUTER PROGRAM

The advantages of using the utility DED program are: simplification of selection, deterministic process, automatic evaluation of requirements criteria, and built in sensitivity or "what if" capability. Making a selection of an alternative from a standard DED is usually not a simple process of picking the alternative with the highest performance for the lowest cost. Since the multifactor criteria are reduced to a single utility value in the UDED program, the analyst can at a glance evaluate cost versus performance.

The basic DED does not allow the analyst to formally rank the importance of the performance indicators nor provide a weighting factor. Thus, the analyst must make a more subjective assessment of the cost of an alternative versus its perceived performance. For the UDED, in cases where the weighted indexes are close or where there is a large difference in two alternatives e.g., low cost and minimal performance versus high cost

and high performance, the individual weighted scores for the alternatives can be compared to obtain a more precise understanding of the driving factors. This allows the analyst to make a more informed decision.

The analyst must often tradeoff cost against non-cost criteria such as mean time between failures (MTBF), availability, speed, weight, or mean time to repair (MTTR). This tradeoff analysis may involve reevaluating program baseline and derived requirement limits, the perceived importance of a multifactor criterion, or the cost limitations to be imposed. The UDED program allows the analyst to perform sensitivity analyses on the limits of the requirements and the ranking/rating of the performance indicators simply by changing the data and re-running the calculations.

SECTION 3.4 - DISADVANTAGES OF THE UDED COMPUTER PROGRAM

The disadvantages of using the Utility DED program are: some loss of insight into individual performance indicator affects, loss of curve fit tailoring based on past history, and defining performance indicator benchmark endpoints may be difficult (0, 10). While the utility/preference scale simplifies the final analysis of the DED, it also has a tendency to mask the individual traits/impacts of the performance indicators. This is especially true if the analyst chooses a moderate ranking for a performance indicator since the weighting factor will further cloud that indicator's impact on the final weighted score for each alternative.

One of the biggest program limitations is in the area of curve fitting. In some cases, historical data may be available which the analyst would want to use in generating the utility curve; however, the UDED program will not allow that data to be entered. The analyst is basically constrained to straight lines or curves with a single inflection point i.e., second order (x^2) curves.

Initially, defining performance indicator benchmark endpoints seems to be straightforward. Answering what is the worst level that is possible, and what is the best level that can be achieved would seem to be easy. However, only an analyst with a strong understanding of the individual multifactor criteria can make those judgments

realistically. Fortunately, since there is a computer program involved, tweaking the best and worst values does not require extensive time.

SECTION 3.5 - ENHANCEMENTS

Several enhancements to the basic UDED program are possible which would expand its usefulness. The area with the greatest potential for improvement is the curve fitting routine. Currently, at most, only a second order curve fit is allowed. Expanding the curve fitting routine to include sine waves, exponentials, logarithms, and third or higher order curves or allowing for piecewise curve fitting would provide the analyst additional "what if" capabilities and could better reflect any historical data which might be available.

Another place where enhancements could be beneficial would be in providing graphical forms of the DED and UDED as well as showing the utility scale curve fits for each performance indicator. Increasing the number of performance indicators and alternatives allowed would also be advantageous when using the program for a large project.

Allowing data to be imported or exported into/out of the program could allow the user more flexibility. By allowing for the export of data, another program could be used to provide a graph of the DED or UDED.

To enhance the "what if" capability of the UDED computer program, the ability to change the filename and keep past data would be invaluable. The analyst would not have to reenter data or as a minimum exit to DOS to copy the file to another name in order to reuse it.

SECTION 3.6 - CONCLUSION

The utility decision evaluation display (UDED) program has several limitations both conceptually and in implementation; however, the ability to perform "what if" evaluations and to reduce the multifactor criteria to a single value is a useful and useable tool. The limitation in the curve fitting routine, the lack of graphical representations, and the limited number of performance indicators and alternatives allowed can be overcome by

enhancing the basic program. The conceptual limitations such as losing insight into how a specific performance indicator affects the outcome must be recognized by the analyst. The analyst can in turn perform sensitivity analyses to gain a firmer understanding of the affect of a specific indicator, but this requires the analyst to have a broad understanding of the interactions between the rating and the benchmarks selected.

This computer program like any other is only as good as the data entered and the human analysis of the results provided. It enhances and, in some cases, simplifies the decision making process which must be undertaken, but it cannot think nor evaluate data the way a human would.

SECTION 4.0 - REFERENCES

[1] Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

[2] Blanchard, Benjamin S., and Wolter J. Fabrycky, Systems Engineering and Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1990.

[3] Sink, D. Scott, Productivity Management: Planning, Measurement and Evaluation, Control and Improvement. New York: John Wiley & Sons, Inc., 1985.

[4] Kernighan, Brian W., and Dennis M. Ritchie, The C Programming Language. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1988.

[5] Lafore, Robert, Turbo C Programming for the IBM. Indianapolis, Indiana: Howard W. Sams & Company, 1987.

[6] Radford, K. J., Modern Managerial Decision Making. Reston, Virginia: Reston Publishing Company, Inc., 1981.

APPENDIX A

UTILITY DECISION EVALUATION DISPLAY COMPUTER PROGRAM

USER'S GUIDE

APPENDIX A
UTILITY DECISION EVALUATION DISPLAY COMPUTER PROGRAM USER'S GUIDE

SECTION 1.0 - INTRODUCTION

The purpose of this user's guide is to provide a basic understanding of how to use the utility decision evaluation display (UDED) program and to describe some of its limitations.

SECTION 1.1 - BACKGROUND

The decision evaluation display (DED - Figure A1) is a useful method for comparing mutually exclusive alternatives based on their life-cycle cost (LCC) and other multifactor criteria. The multifactor criteria can include mean time between failure (MTBF), speed, weight, design to cost (DC), or any other parameter which provides needed information about an alternative.[5]

There are several problems in using the standard DED: the inability to weight criteria based on their importance and the non-deterministic process used in evaluating the display. This program allows the user to weight the various multifactor criteria based on their importance (ranking). It then uses a utility curve to translate the multifactor criteria with differing units, e.g. pounds, miles per hour, seconds into a non-dimensional value found on the utility or preference scale. These utility values are then weighted according to the user's specifications and a total indicator is calculated for each alternative. Thus, a single two-dimensional graph can be formed with LCC on the x-axis and the total indicator (weighted utility value) on the y-axis (Figure A2).

The final choice of alternative must in some cases still be a subjective one based on the evaluator's "gut feel." A judgement will need to be made on how much increased "performance" is worth in increased LCC. However, the set of alternatives which must be evaluated in this manner is reduced.

---

[5] Blanchard, Benjamin S., and Wolter J. Fabrycky, Systems Engineering and Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1981.

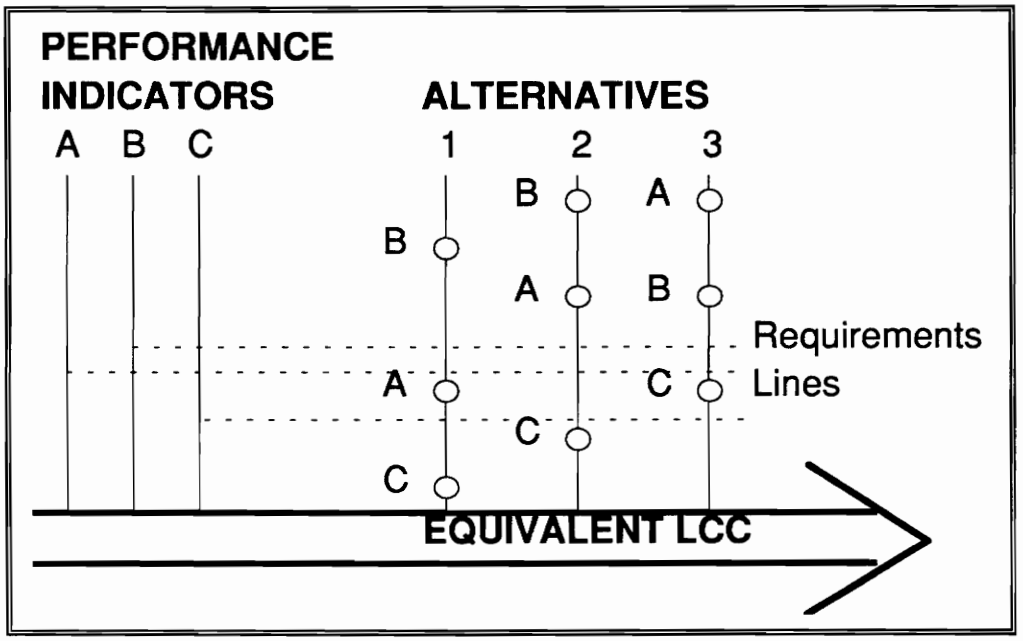

Figure A1. Decision Evaluation Display[1]

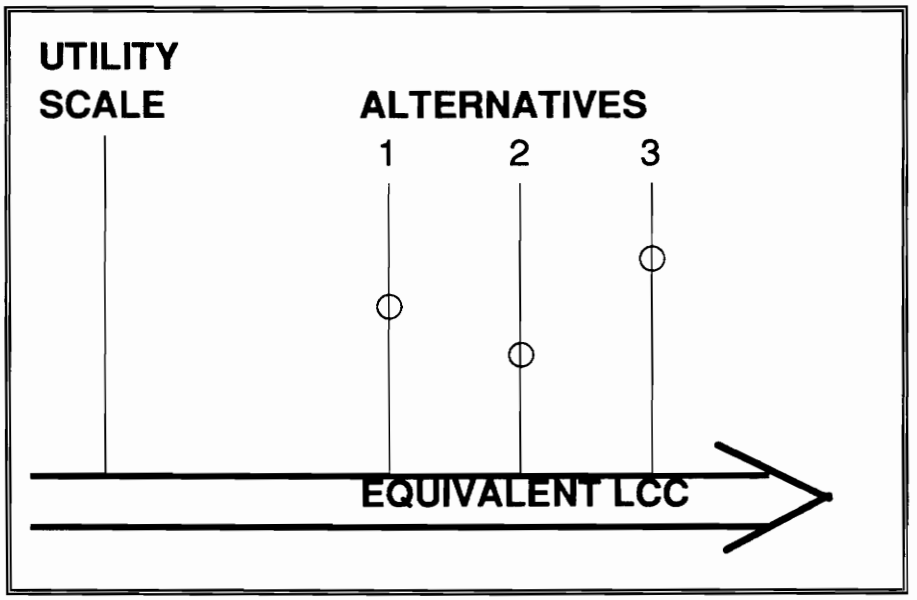

Figure A2. Utility Decision Evaluation Display

[1]Fabrycky, Wolter J., and Benjamin S. Blanchard, Life-cycle Cost and Economic Analysis. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1991.

SECTION 1.2 - EQUIPMENT REQUIREMENTS/LOADING THE PROGRAM

In order to use the utility decision evaluation display (UDED) program, you will need as a minimum an IBM PC compatible computer with 512k of memory and one 360k disk drive. A monochrome monitor can be used although a color monitor is desirable.

The UDED program can be executed simply by typing in UDED at the DOS prompt and hitting return.

SECTION 2.0 - PROGRAM CONFIGURATION

The UDED program is setup in six (6) basic modules. They are the main menu, multifactor (performance) criteria, mutually exclusive alternatives, calculations, reports, and program setup.

SECTION 2.1 - USER INTERFACES

SECTION 2.1.1 - MAIN MENU

The main menu which is shown in Figure A3 allows the user to select one of the remaining five modules, quit the program, or get help. Selections can be made by pressing the appropriate key (1-6) or by using the up or down arrow keys to reach the desired option and pressing return.

The data drive and directory are displayed for the user's information in the lower right hand corner.

SECTION 2.1.2 - POP UP MENU

The basic pop up menu shown in Figure A4 is common to the multifactor (performance) criteria and the alternatives modules. The pop up menu is called by pressing F2. The user then uses the up- or down- arrow keys to reach the desired selection and then presses return. All four of the possible selections (ADD DATA, EDIT DATA, DELETE DATA, SAVE CHANGES) are context sensitive for the particular module. In order to leave the pop up menu, the user presses ESC.

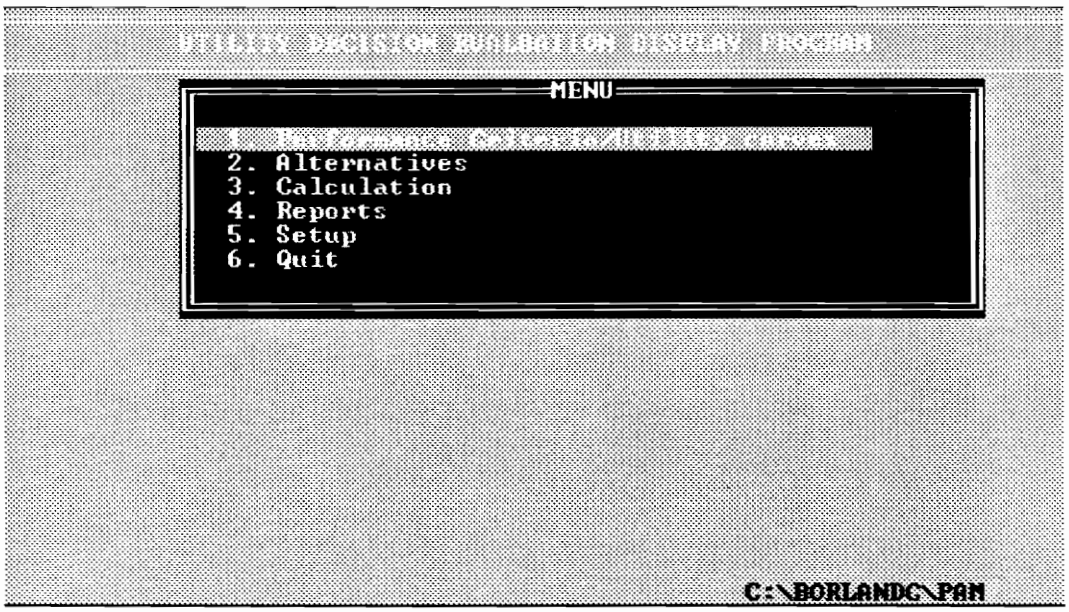

Figure A3. UDED Main Menu

Figure A4. Pop Up Menu

ADD DATA is used for basic data entry. Once selected, a blank data entry screen is brought up on the right-hand side of the screen.

EDIT DATA is used to modify data which has previously been entered. Select the edit option in the pop up menu by using the arrow keys and pressing return. A scroll bar is place over the last possible data selection; use the arrow keys to reach the desired data to edit and press return. A painted data entry screen is brought up.

DELETE DATA is used to remove data. Select the delete option in the pop up menu by using the arrow keys and pressing return. A scroll bar is placed over the last data selection; use the arrow keys to reach the desired data to delete and press return. A window will pop up to verify if the user wants to delete the selected item. If the user presses return, the item will be deleted; if they press ESC, the item is left unmodified.

SAVE CHANGES is used to save data from memory into a file. Select the save option in the pop up menu by using the arrow keys and pressing return. A window will pop up to inform the user that the file is being saved. The user must save all data prior to exiting the program; otherwise, it will be lost.

SECTION 3.0 - PROGRAM MODULES

SECTION 3.1 - PERFORMANCE INDICATOR MODULE

The performance indicator (multifactor criteria) module is used to enter the performance indicators, rank, rating, and their associated utility curve benchmark values at 0, 5, and 10. This should be the first module into which data is entered.

If the user has not selected a file name under the setup module, a message will instruct the user to exit and to enter the desired filename under the setup module. The user may then return to the performance indicator module and begin entering data.

The first step is to press F2 to bring up the pop up menu. To enter data, select ADD DATA. A data entry screen will appear to the right where the data is to be entered (Figure A5). The user enters the performance indicator name (which may not exceed 9 characters). The rank and rating values are associated with one another. In general, the higher the rank of

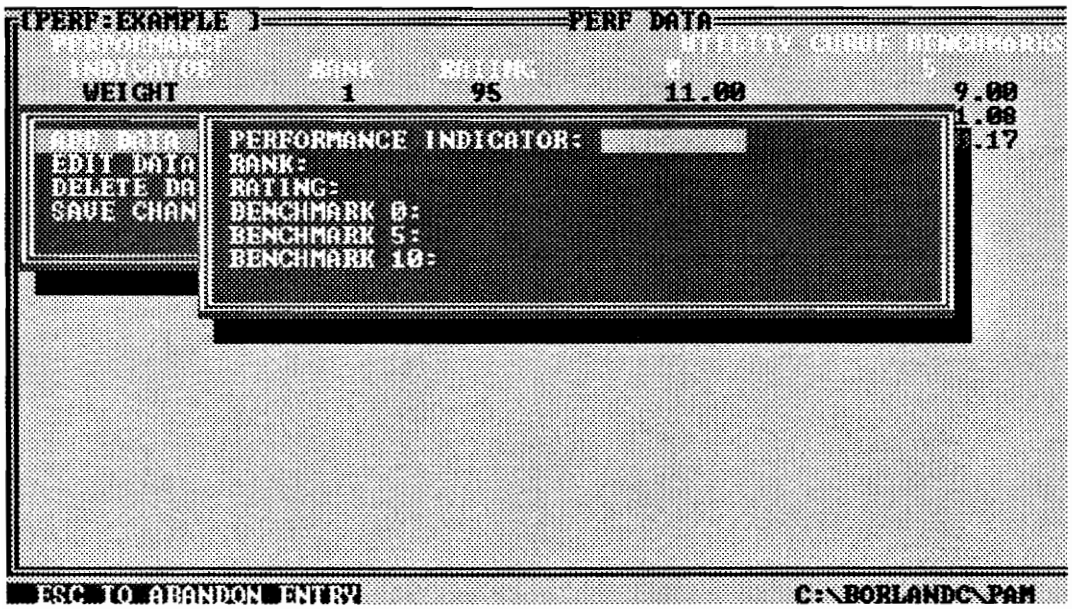

Figure A5. Performance Indicator Data Entry Screen

a performance indicator, the higher its rating will be. The values for rank can range from 1 (most important) to 12 (for the least important - e.g., if there are only three indicators then the least important would be assigned the rank of 3). The rating values range from 100 to 1 (note if 0 is entered the indicator is ignored). A rating of 100 is the highest possible value and that indicator will receive the highest weighting.

Values are then entered for the utility benchmarks. The value entered for benchmark utility value 0 should correspond to the least acceptable value expected for a performance indicator. The value entered for benchmark utility value 10 should correspond to the most excellent value achievable for a performance indicator. The value entered for benchmark utility value 5 should be the minimum acceptable performance indicator value; this can also be used as a hard requirement holder.

The benchmark values entered will be used by the program to calculate a first or second order least squares curve fit. The utility benchmark value entered for 5 may also be used during calculations to weed out those alternatives which do not meet the minimum acceptable requirements.

A maximum of twelve (12) performance indicators may be entered.

Several error checking and data integrity routines are included in the program. If performance indicators are entered after mutually exclusive alternatives have been entered, a message will come up to remind the user to update the alternatives file. If the performance indicators are edited after entering mutually exclusive alternatives, a check will be made to verify that the values entered for the alternatives still fall within the limits of the performance indicator utility benchmark values. If they do not, a message will come up asking if the user wishes to continue with the modified data. If the user deletes a performance indicator after entering alternatives, the alternatives data will be modified after the user agrees.

The pop up menu is used to add, edit, delete, or save the data. Data should be saved prior to exiting from the main program or it will be lost. In order to get out of the pop up menu, press ESC. To return to the main menu, press ESC.

SECTION 3.2 - ALTERNATIVES MODULE

After entering the performance indicators, the next step is to enter the mutually exclusive alternatives with their associated life-cycle cost and performance indicator values. The file name will have already been selected based on the choice entered under the setup module.

Press F2 to call up the pop up menu and select ADD DATA. A data entry screen will come up to the right and a second screen with the maximum and minimum values allowed for the performance indicators will come up in the bottom left-hand portion of the screen (Figure A6). Enter the alternative name, life-cycle cost, and the values for each of the performance indicators within the maximum and minimum allowable values.

The values entered for the life-cycle cost should be of the same type (i.e., LCC types may not be mixed). For example, all should be present equivalent worth (PEW) or annual equivalent worth (AEW). The LCC type is entered under the setup module.

A maximum of 15 alternatives may be entered.

The program will verify that the value entered for a particular performance indicator falls with in the maximum and minimum values for the benchmarks. If it does not, a warning will appear noting the performance indicator and the allowable value range. The data entry screen is redrawn for the user to edit the appropriate value.

The pop up menu is used to add, edit, delete, or save the data. Data should be saved prior to exiting from the main program or it will be lost. In order to get out of the pop up menu, press ESC. Then, to return to the main menu, press ESC again.

SECTION 3.3 - CALCULATIONS MODULE

Once the performance indicators and the alternatives data have been entered, the calculations routine can be run (note: it must be run in order to view or to print the reports). There are two different calculation algorithms available. The first, which is called regular calculation, converts the performance indicator values entered for each alternative into utility values, calculates the weighted scores, and the weighted index. The second calculation routine, which is called requirement line calculation, allows alternatives which do not meet

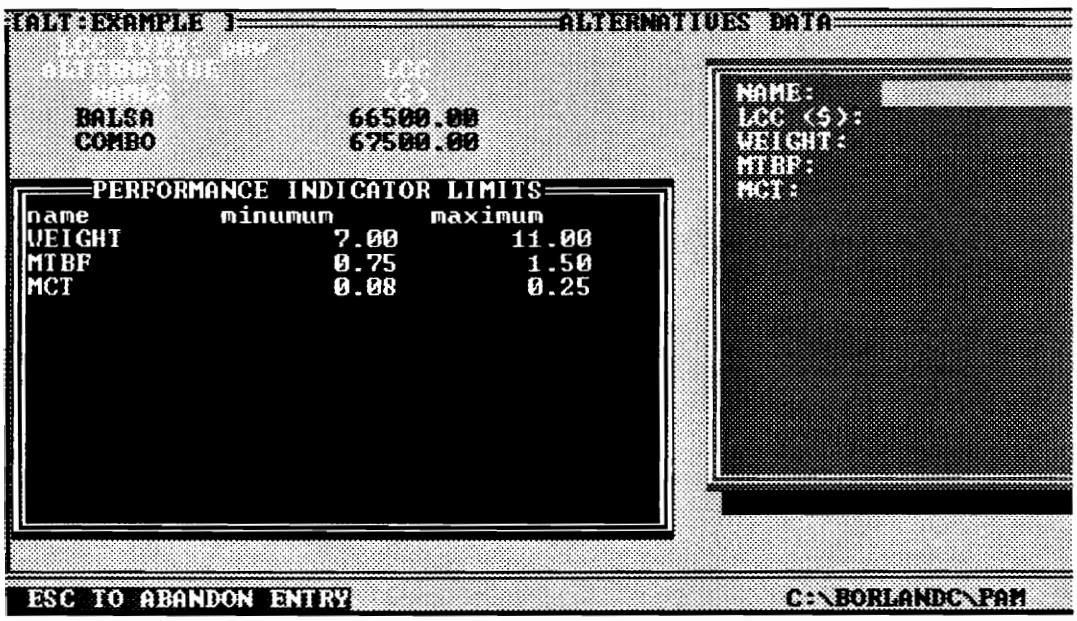

Figure A6. Alternatives Data Entry Screen

the minimum hard requirements (i.e. those levels defined by the performance indicator benchmark value of 5) to be removed from consideration automatically by setting the weighted score and weighted index to zero. The other parameters are calculated as in the regular calculation routine. Figure A7 shows the calculations screen.

In this module, the performance indicators, the alternatives, and the calculations data are saved automatically.

In order to get out of the calculations pop up menu and to return to the main menu, press ESC. After running one of the calculation routines, control is automatically returned to the main menu.

The user can then use the report module to analyze the results.

SECTION 3.4 - REPORT MODULE

The report module provides the user with the capability of displaying the desired report to the screen or printing it out (Figure A8). The report formats available are:

1. Performance Indicator - displays the performance indicator name, rank, rating, benchmark values, and performance indicator weight (Figure A9).
2. Alternatives - displays the life-cycle cost type, alternative number, name, lcc, and performance indicator values (Figure A10).
3. Alternatives utility - displays the life-cycle cost type, alternative number, name, lcc, and utility values for the performance indicators (Figure A11).
4. Weighted scores - displays the life-cycle cost type, alternative number, name, lcc, performance indicator weighted score, and the weighted index (Figure A12).

If required, additional screens of data can be viewed by using the page down key. F3 is used to toggle between printing to the screen and the printer.

Press ESC to return to the report menu from a report screen display. Go to selection 6 to return to the main menu.

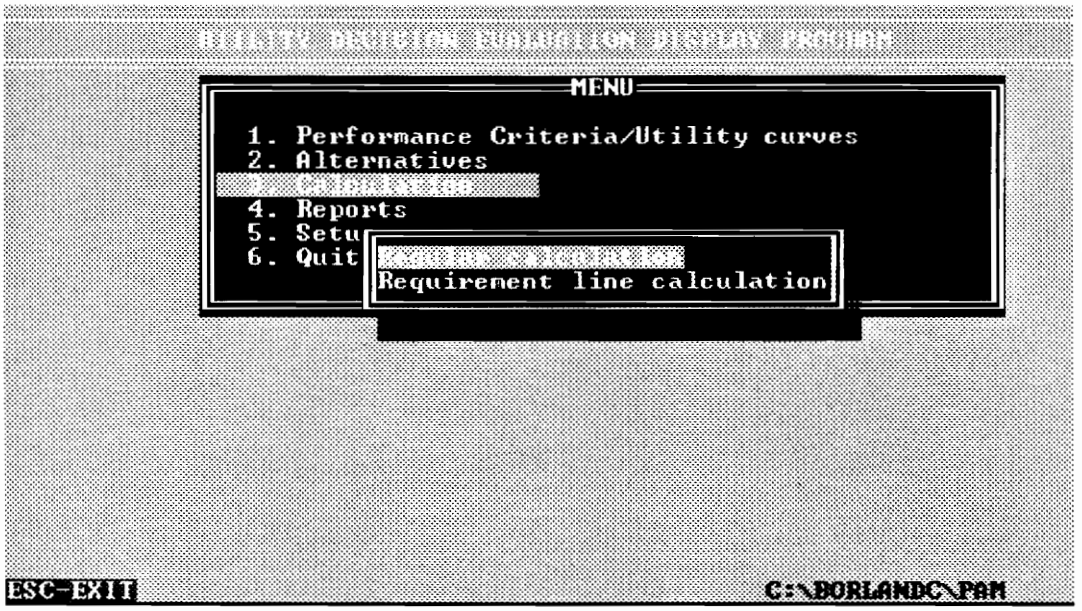

Figure A7. Calculations Screen

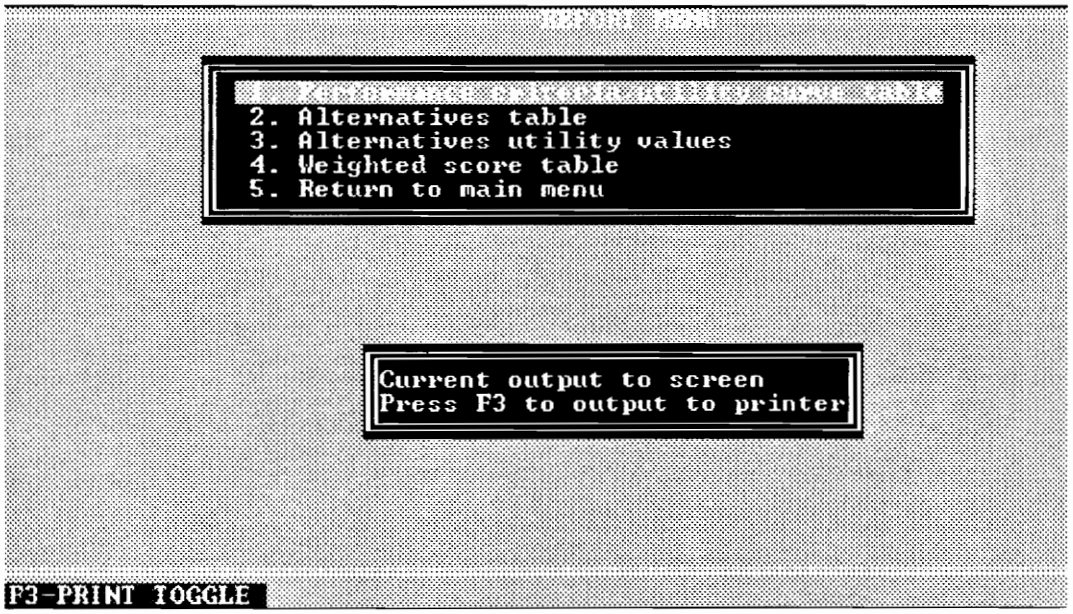

Figure A8. Report Screen

PERFORMANCE INDICATORS

PERFORMANCE INDICATOR	RANK	RATING	UTILITY CURVE BENCHMARKS 0	5	10	WEIGHT
WEIGHT	1	95	11.00	9.00	7.00	36.54
MTBF	2	85	0.75	1.08	1.50	32.69
MCT	3	80	0.25	0.17	0.08	30.77

Figure A9. Performance Indicator Report

ALTERNATIVES DATA

LCC TYPE: pew

ALTERNATIVE NAME	LCC ($)	PERFORMANCE INDICATOR VALUES WEIGHT	MTBF	MCT
BALSA	66500.00	8.50	1.07	0.16
COMBO	67500.00	8.00	1.10	0.15

Figure A10. Alternatives Report

ALTERNATIVES UTILITY DATA

LCC TYPE: pew

ALT NUM	ALTERNATIVE NAME	LCC ($)	PERFORMANCE INDICATOR UTILITY VALUES WEIGHT	MTBF	MCT
1	BALSA	66500.00	6.25	4.87	5.45
2	COMBO	67500.00	7.50	5.26	6.06

Figure A11. Alternatives' Utility Value Report

ALTERNATIVES WEIGHTED SCORE DATA

LCC TYPE: pew

ALT NUM	ALTERNATIVE NAME	LCC ($)	WEIGHTED SCORES WEIGHT	MTBF	MCT	WEIGHTED INDEX
1	BALSA	66500.00	228.37	159.09	167.83	555.29
2	COMBO	67500.00	274.04	172.12	186.48	632.64

Figure A12. Weighted Scores Report

SECTION 3.5 - SETUP MODULE

The setup module allows the user to enter two data items: the desired filename and the life-cycle cost type (Figure A13). The file name is used by the other modules in order to save data to files. The life-cycle cost type is printed out as a part of the reports information. All LCCs must be of the same type in order for the comparison on the UDED (or DED) to be valid. LCC types would include present equivalent worth (PEW) or annual equivalent worth (AEW).

SECTION 4.0 - EXAMPLE PROBLEM USING MODEL AIRPLANES

SECTION 4.1 - PROBLEM STATEMENT

Joe wants to build seven Royal Zero one-fifth scale radio controlled airplane models for a movie studio. He has two building techniques to choose from: all balsa wood or a combination of balsa wood and foam. Joe has three non-cost parameters which he wants to include in his decision process: model weight, Mean Time Between Failure (MTBF), and Mean Corrective Maintenance Time ($\overline{M}_{ct}$).

Since the movie studio wants an acrobatic model, the maximum weight allowed is 9 lbs. They also want a model which has a low failure rate and which requires a fairly low amount of maintenance. Therefore, the minimum MTBF is 1.08 hours (65 minutes); the maximum $\overline{M}_{ct}$ is 0.167 hours (10 minutes).

Based on similar type models, the one-fifth scale all balsa wood Royal Zero model has the following multifactor criteria values: weight 8.5 lbs, MTBF 1.08 hours (65 minutes), $\overline{M}_{ct}$ 0.16 hours (9.6 minutes). The combination model has the following multifactor criteria values: weight 8.0 lbs, MTBF 1.25 hours (75 minutes), $\overline{M}_{ct}$ 0.15 hours (9 minutes).

The Life Cycle Cost (LCC) Present Equivalent Worth (PEW) is $66,500 for seven all balsa wood models and $67,500 for seven combination models.

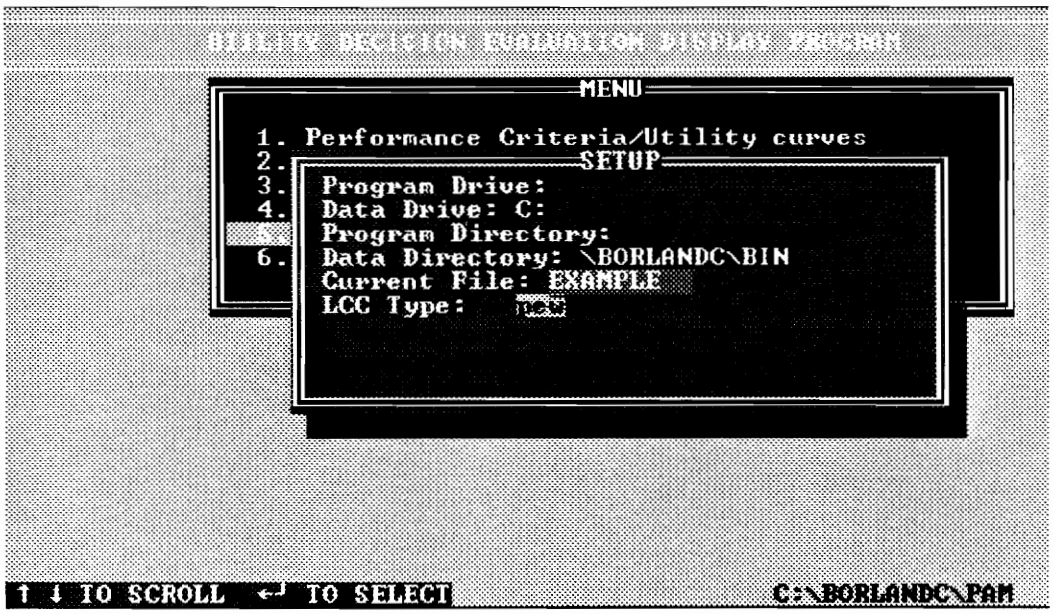

Figure A13. Setup Menu

SECTION 4.2 - USING THE UDED PROGRAM

SECTION 4.2.1 - SETUP

The first step is to go to the Setup module and enter a file name for the example; the text will use EXAMPLE. Then enter the LCC type of PEW. This will return you to the main menu.

SECTION 4.2.2 - PERFORMANCE INDICATORS

The next step is to select item 1, Performance Indicators, from the menu. Once the Performance Indicator screen comes up, select F2 for the pop up menu. Select ADD DATA from the pop up menu and enter WEIGHT for the Performance Indicator name. For the rank, enter 1. For the rating, enter 95. The entered rank and rating will make WEIGHT the performance indicator with the most importance. The maximum weight desired is 9 lbs based on the problem statement. Therefore, the benchmark value for utility value of 5 should be 9. Selecting the values for 0 and 10 is somewhat harder. In theory the best possible value (benchmark utility value of 10) would be 0; however, in the real world, a more reasonable value would be 7 lbs. The worst possible value (benchmark utility value of 0) could possibly be a ton, but again a more realistic value would be 11 lbs. Therefore, enter 11 for the 0 benchmark value, 9 for the 5 benchmark value, and 7 for the 10 benchmark value. After the <CR> is pressed for the 10 benchmark value, control is returned to the pop up menu.

Select ADD DATA again. Enter MTBF for the Performance Indicator name in the data entry window. Enter 2 for the rank, and 85 for the rating. MTBF is the second most important performance indicator. The minimum MTBF required in the problem statement is 1.08 hours (65 minutes); therefore, 1.08 should be entered as the benchmark utility value for 5. Selecting the appropriate values for the benchmarks of 0 and 10 requires a similar thought process as that used for the weight. The best MTBF would be infinity (i.e., never failing), a more realistic value for a model airplane is 1.5 hours (90 minutes). The worst MTBF would be 0 (i.e., always failing), again a more likely value for this example is 0.75 hours (45 minutes). Enter 0.75 for benchmark value 0, 1.08 for benchmark value 5, and 1.5 for benchmark value 10. After the <CR> is pressed for the 10 benchmark value, control is returned to the pop up menu.

Select ADD DATA again. Enter MCT[2] for the Performance Indicator name in the data entry window. Enter 3 for the rank and 80 for the rating. The maximum allowable M_{ct} is 0.167 hours (10 minutes) based on the problem statement. Thus, 0.167 is the utility benchmark value for 5. Using a similar thought process as above, the best MCT (benchmark value 10) is 0.083 (5 minutes). The worst MCT (benchmark value 0) is 0.25 (15 minutes). Enter 0.25 for benchmark 0, 0.167 for benchmark 5, and 0.083 for benchmark 10. After the <CR> is pressed for the benchmark 10 value, control is returned to the pop up menu.

At this point, you have entered all of the performance indicators for the example. If you want to correct a mistake, select EDIT DATA from the pop up menu, scroll to the appropriate indicator, and press <CR>. A painted data entry screen will come up for that performance indicator. Use the up- and down-arrow keys to reach the data to edit, then type in the new value. Press <ESC> to return to the pop up menu.

If the data is correct, use the down arrow key to reach SAVE CHANGES in the pop up menu and press <CR>. The data entered has now been saved to a file called EXAMPLE.PRF.

Press <ESC> twice to return to the main menu.

SECTION 4.2.3 - ALTERNATIVES

Press 2, Alternatives, or scroll down to it and press <CR> to enter the alternatives menu.

Once the alternatives screen has come up, press F2 to bring up the pop up menu. To enter the alternatives data, select ADD DATA. The data entry screen will appear to the right and the performance indicator limits screen will appear to the lower left.

Enter BALSA as the alternative name, then enter 66500 for the LCC (note: do not use "$" or "," when entering cost data). Enter the following values for WEIGHT, MTBF, MCT, respectively, 8.5, 1.08, 0.16. After the <CR> is pressed for the last performance indicator value, control is returned to the pop up menu.

---

[2]The computer does not recognize $\overline{M}_{ct}$, thus MCT is used.

Select ADD DATA. Enter COMBO as the alternative name, then enter 67500 for the LCC cost in the data entry window. Enter 8 for WEIGHT, 1.1 for MTBF, and 0.15 for MCT. After the <CR> is pressed for the last performance indicator value, control is returned to the pop up menu.

If any of the values entered, fall outside of the limits entered for the performance indicators, a warning message comes up and the painted data entry screen is drawn. Use the arrow keys to reach the incorrect value and type in the correct one. Press <ESC> to return to the pop up menu.

To verify the values entered, select EDIT DATA in the pop up menu, then select the desired alternative. A painted data entry screen will come up on the right. If the data is correct as typed, press <ESC> to return to the pop up menu; otherwise, use the arrow keys to reach the incorrect value and type over it. Press <ESC> to return to the pop up menu. The other alternative's data can be verified in the same manner.

Once the data is correct, select SAVE CHANGES in the pop up menu. The alternatives data will then be saved to a file called EXAMPLE.ALT.

Press <ESC> twice to return to the main menu.

SECTION 4.2.4 - CALCULATIONS

Press 3 or scroll to calculations and press return to bring up the calculations menu. There are two calculation algorithms to choose from: regular or requirement line. Select regular and press <CR>. The performance indicator weights, alternatives utility values, weighted scores, and weighted index will be calculated and saved automatically. Control is returned to the main menu.

SECTION 4.2.5 - REPORTS

Select the report module by pressing 4 or by scrolling to report and pressing return to bring up the report menu. F3 can be used to toggle between printing to the screen or to the printer; it defaults to the screen. Select any of the reports to view; pressing <ESC> will return control to the report menu.

Select the weighted score report. Press 1 to provide a report ordered by the weighted index. The COMBO alternative has the higher weighted index, but it also has the higher LCC (Figure A14).

A part of the human analysis process is to determine whether the added performance is worth the extra cost. The program cannot provide that type of information. However, the program can be used to determine the sensitivity of the various multifactor criteria based on the rating values and benchmark values chosen.

SECTION 4.2.6 - MODIFICATIONS

Go back to the main menu. Select alternatives (2). Press F2 and select EDIT DATA. Select BALSA. Change the value for MTBF from 1.08 to 1.07 which is below the benchmark 5 value. Press <ESC> to return to the pop up menu. Select SAVE CHANGES. Press <ESC> twice to return to the main menu.

Select calculations again. This time use the requirement line calculation. Enter the report module, and select the weighted score report. Press 1 for the sorted report. This time the only alternative available is COMBO. Since the MTBF for BALSA does not meet the benchmark 5 requirement (1.07 versus 1.08), its weighted score is zero (0) and the weighted index for BALSA is set equal to zero (0) (Figure A15).

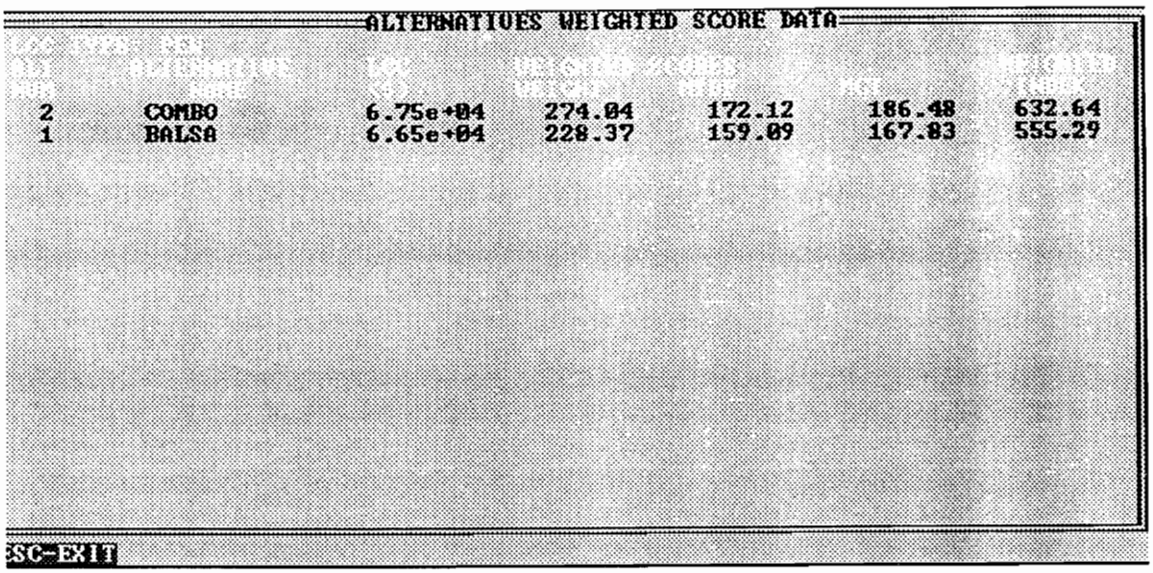

Figure A14. Example - Ordered Weighted Scores Screen: Regular Calculation

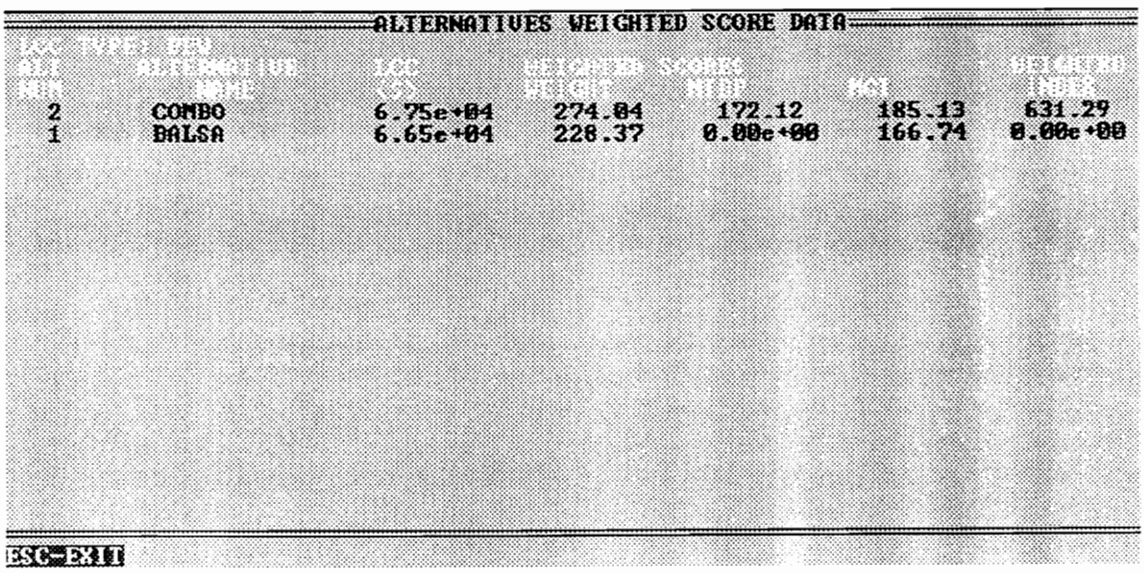

Figure A15. Example - Ordered Weighted Scores Screen: Requirements Line Calculation

APPENDIX B

UTILITY DECISION EVALUATION DISPLAY COMPUTER PROGRAM

PROGRAMMER'S GUIDE

APPENDIX B
UTILITY DECISION EVALUATION DISPLAY
COMPUTER PROGRAM
PROGRAMMER'S GUIDE

SECTION 1.0 - GENERAL INFORMATION

SECTION 1.1 - INTRODUCTION

The purpose of this programmer's guide is to provide the necessary information to allow the UDED computer program to be maintained and/or modified. Basic information regarding the program layout, compiler settings, data dictionaries, variable definitions, and module and subroutine descriptions are provided. Module names are capitalized, and structure names are italicized throughout the text.

SECTION 1.2 - PROGRAM LAYOUT

The program is divided into six modules. DEDMAIN is the main program and controls access to the other five subprograms (PERF, ALT, CALC, REPORT, SETUP). There are four data structures (*perf, alt, calc, info*) which are used in the program. Section 2.0 describes which structure variables are used in the various program modules. The infinity library routines were written by Ken Rice to support a program he was writing. They provide an easy means of getting user input, displaying messages, and cursor control. A commented listing of those routines can be found in Section 4.0.

SECTION 1.3 - COMPILER DESCRIPTION AND SETTINGS

The Borland C++ compiler version 2.0 was used to compile the "C" source code. The source code and complier adhere to ANSI "C" standards. Standard DOS code was generated using floating point emulation and the 80286 instruction set. The code was optimized for size and was compiled using the medium memory model with a standard stack frame.

SECTION 2.0 - STRUCTURE DESCRIPTIONS

SECTION 2.1 - STRUCTURE *PERF*

SECTION 2.1.1 - DATA DICTIONARY

Variable name	Type	Module
a0	float	CALC
a1	float	CALC
a2	float	CALC
cperf_0	character 15	PERF
cperf_5	character 15	PERF
cperf_10	character 15	PERF
perf_0	float	PERF, CALC
perf_5	float	PERF, CALC
perf_10	float	PERF, CALC
perf_name	character 10	PERF
perf_num	unsigned integer	CALC
perf_rank	unsigned integer	PERF, CALC
perf_rate	unsigned integer	PERF, CALC
perf_wt	float	PERF, CALC
*next	pointer	DEDMAIN, PERF, ALT, CALC, REPORT
*prior	pointer	DEDMAIN, PERF, ALT, CALC, REPORT

SECTION 2.1.2 - DEFINITIONS

perf_name	The performance criteria name is entered by the user. Limit of 9 characters.
perf_rank	The performance criteria rank or priority is assigned by the user based on the relative importance of each of the criteria where 1 is the most important criteria. Unsigned Integer values only.

perf_rate	The performance indicator rating is a more specific application of the ranking of each of the criteria. The user assigns point values based on the perceived importance of each performance criteria. Typically, the highest priority or priorities receive 100 points. The lowest priority can receive as few as 1 point (note the use of 0 will totally remove the criterion from consideration). Unsigned Integer values only.
perf_0/ cperf_0[15]	The user enters the desired benchmark value for the utility curve value equal to 0. This should relate to the lowest level anticipated for the performance criteria. Single precision decimal/character 15.
perf_5/ cperf_5[15]	The user enters the desired benchmark value for the utility curve value equal to 5. This should relate to the acceptable level for the performance criteria or to a hard requirement. Single precision decimal/character 15.
perf_10/ cperf_10[15]	The user enters the desired benchmark value for the utility curve value equal to 10. This should relate to the excellent level for the performance criteria. Single precision decimal/character 15.
perf_wt	This variable is equal to the division of each of the perf_rate values by perf_rate_tot which is then multiplied by 100 to obtain the weighted values for each performance criteria. Single precision decimal.
perf_num	The program assigns a number to each performance criteria. Unsigned integer only.
a0	This variable is the constant for x^2 in the least squares curve fit. Single precision decimal.
a1	This variable is the constant for x in the least squares curve fit. Single precision decimal.
a2	This variable is the constant (y-intercept) in the least squares curve fit. Single precision decimal.
*next	This is a pointer to the next set of data.
*prior	This is a pointer to the prior set of data.

SECTION 2.2 - STRUCTURE *ALT*

SECTION 2.2.1 - DATA DICTIONARY

Variable name	Type	Module
alt_lcc	float	ALT, REPORT
alt_lcc_type	character 4	ALT, REPORT
alt_name	character 15	ALT
alt_perf_crit	float array [15]	ALT, REPORT
calt_lcc	character 15	ALT
calt_perf_crit	character array [15][15]	ALT
*next	pointer	DEDMAIN, PERF, ALT, CALC, REPORT
*prior	pointer	DEDMAIN, PERF, ALT, CALC, REPORT

SECTION 2.2.2 - DEFINITIONS

alt_name	The user enters the name of the alternative. Limit of 14 characters.
alt_lcc	The user enters the LCC for each alternative. Single precision decimal.
alt_perf_crit[15]/ calt_perf_crit[15][15]	The user enters the value for all performance indicators for a given alternative. Single precision decimal.
*next	This is a pointer to the next set of data.
*prior	This is a pointer to the prior set of data.

SECTION 2.3 - STRUCTURE *CALC*

SECTION 2.3.1 - DATA DICTIONARY

Variable name	Type	Module
alt_num	unsigned integer	CALC, REPORT
alt_util_val	float array [15]	CALC, REPORT
alt_wt_score	float array [15]	CALC, REPORT
alt_wt_ind	float array [15]	CALC, REPORT
altname	character 15	REPORT
altlcc	float	REPORT
*next	pointer	DEDMAIN, PERF, ALT, CALC, REPORT
*prior	pointer	DEDMAIN, PERF, ALT, CALC, REPORT

SECTION 2.3.2 - DEFINITIONS

alt_num	This variable is calculated by the program based on the total number of alternatives entered. Unsigned integer values only.
alt_util_val[15]	Alternative utility values for all performance indicators for each alternative. It calculated by using a least squares curve fit routine based on the benchmark values entered. Single precision decimal.
alt_wt_score[15]	Alternative weighted score which is based on multiplying alt_util_val[i] by perf_wt. Single precision decimal.
alt_wt_ind	This variable is calculated by summing all of the values in alt_wt_score[i] for a given alternative. Single precision decimal.
altname	This variable is copied from the alt structure variable alt_name to allow the weighted scores report sorting to be done easier. Character 15.

altlcc	This variable is copied from the alt structure variable alt_lcc to allow the weighted scores report sorting to be done easier.
*next	This is a pointer to the next set of data.
*prior	This is a pointer to the prior set of data.

SECTION 2.4 - STRUCTURE *INFO*

SECTION 2.4.1 - DATA DICTIONARY

Variable name	Type	Module
directory	character 64	DEDMAIN, PERF, ALT
drive	character 2	DEDMAIN, PERF, ALT
name	character 9	DEDMAIN, PERF, ALT, CALC
alt_lcc_type	character 4	DEDMAIN, ALT, REPORT

SECTION 2.4.2 - DEFINITIONS

directory	This variable holds the information on program directory. Limit of 63 characters.
drive	This variable holds the information on program drive. Limit of 1 character (e.g., A, B, C).
name	The user enters the filename to be used. Limit of 8 characters.
alt_lcc_type	The user enters the type of LCC to be entered for each alternative: PEW, AEW, FEW. Limit of 3 characters.

SECTION 3.0 - MODULE DESCRIPTIONS

SECTION 3.1 - DEDMAIN MODULE

Allows the user to select which of the other modules to enter from the main menu or to exit the program.

SECTION 3.1.1 - SUBROUTINES

1. **menu_select**
   - prototype: int menu_select(void)
   - description: paints main menu, returns menu selection
2. **get_disk_info**
   - prototype: void get_disk_info(void)
   - description: gets current directory and disk information
3. **display_dir**
   - prototype: void display_dir(void)
   - description: displays current directory on screen
4. **draw_screen**
   - prototype: void draw_screen(void)
   - description: draws initial screen
5. **load_perf_data**
   - prototype: struct perf *load_perf_data(char *, struct perf *)
   - description: loads any existing perf data from file to memory
6. **error_check**
   - prototype: void error_check(void)
   - description: menu error checking routine
7. **load_alt_data**
   - prototype: struct alt *load_alt_data(char *, struct alt *)
   - description: loads any existing alt data from file to memory
8. **perf_cnt**
   - prototype: int perf_cnt(struct perf *)
   - description: counts the number of performance indicators

SECTION 3.1.2 - EXTERNAL SUBROUTINES

1. struct info*read_setup(struct info *) - resides in setup

SECTION 3.1.3 - EXTERNAL VARIABLES

None

SECTION 3.1.4 - EXTERNAL MODULES

1. perf - performance indicator module
2. alt - alternatives module
3. calc - calculations module
4. report - report module
5. setup - setup module

SECTION 3.1.5 - GLOBAL VARIABLES

1. pcount - integer, total number of performance indicators
2. dfree - structure, holds disk information
3. perf - structure, holds performance indicators information
4. alt - structure, holds alternatives information
5. calc - structure, holds calculations information
6. dir - character 64, holds directory information
7. *messages - character pointer, holds messages to go in the bottom left hand corner of the screen which are context sensitive

SECTION 3.2 - PERF MODULE

Allows the user to add, edit, delete, save performance indicator data or return to main menu.

SECTION 3.2.1 - SUBROUTINES

1. **scroll_perf**

prototype	struct perf *scroll_perf(struct perf *)
description	allows the user to scroll through the performance indicators in order to select one to edit or delete

2. **delete_perf**
   - prototype: struct perf *delete_perf(struct perf *)
   - description: allows the user to delete a performance indicator, cleans up pointers
3. **disp_perf**
   - prototype: void disp_perf(struct perf *, int)
   - description: used by scroll_perf routine to display performance indicator data
4. **save_perf_data**
   - prototype: void save_perf_data(struct perf *, struct info *)
   - description: saves performance indicator data from memory to a file
5. **pop_perf_menu**
   - prototype: void pop_perf_menu(void)
   - description: prints the pop up menu (add, edit, delete, save) menu on the screen
6. **perf_menu**
   - prototype: int perf_menu(void)
   - description: allows user to select add, edit, delete, or save
7. **display_perf_data**
   - prototype: struct perf *display_perf_data(struct perf *)
   - description: displays performance indicator data on the screen
8. **add_perf_data**
   - prototype: struct perf *add_perf_data(struct perf *, struct perf *)
   - description: resets pointers if performance indicators are added
9. **get_perf_data**
   - prototype: struct perf *get_perf_data(struct perf *, int)
   - description: allows the user to add or edit performance indicator data, cleans up pointers
10. **draw_perf_screen**
    - prototype: void draw_perf_screen(void)
    - description: draws initial performance indicator screen

11. **alt_add**

prototype	int alt_add(int)
description	if alternatives data has been entered, a warning is issued that the alternatives data must also be updated. It also returns the revised value for pcount.

12. **alt_edit**

prototype	int alt_edit(void)
description	if alternatives data has been entered it issues a warning to the user that the alternatives data may also require modification.

13. **alt_del**

prototype	int alt_del(struct perf *)
description	if alternatives data has been entered, it issues a warning to the user that deleting the selected performance indicator will automatically cause it to be deleted for the alternatives data. It also returns the revised value for pcount.

SECTION 3.2.2 - EXTERNAL SUBROUTINES

1. display_dir - resides in dedmain
2. error_check - resides in dedmain

SECTION 3.2.3 - EXTERNAL VARIABLES

1. pcount - defined in dedmain, total number of performance indicators
2. *messages - defined in dedmain, messages for the lower lefthand corner of the screen which are context sensitive
3. struct perf *begin - defined in dedmain, structure perf pointer
4. struct perf *end - defined in dedmain, struct perf pointer
5. struct alt *start - defined in dedmain, struct alt pointer
6. struct alt *last - defined in dedmain, struct alt pointer

SECTION 3.2.4 - EXTERNAL MODULES

None

SECTION 3.2.5 - GLOBAL VARIABLES

1. *menu[4] - static character pointer. Holds the pop up menu.
2. m, n - integer counters for the pop up menu.

SECTION 3.3 - ALT MODULE

Allows the user to add, edit, delete, save alternatives data or return to main menu.

SECTION 3.3.1 - SUBROUTINES

1. **scroll_alt**

   prototype — struct alt *scroll_alt(struct alt *)

   description — allows the user to scroll through the alternatives in order to select one to edit or delete

2. **delete_alt**

   prototype — struct alt *delete_alt(struct alt *)

   description — allows the user to delete an alternative, cleans up pointers

3. **disp_alt**

   prototype — void disp_alt(struct alt *, int)

   description — used by scroll_alt routine to display alternatives data

4. **save_alt_data**

   prototype — void save_alt_data(struct alt *, struct info *)

   description — saves alternatives data from memory to a file

5. **pop_alt_menu**

   prototype — void pop_alt_menu(void)

   description — prints the pop up menu (add, edit, delete, save) on the screen

6. **alt_menu**

prototype	int alt_menu(void)
description	allows user to select add, edit, delete, or save

7. **display_alt_data**

prototype	struct alt *display_alt_data(struct alt *)
description	displays alternatives data on the screen

8. **add_alt_data**

prototype	struct alt *add_alt_data(struct alt *, struct alt *)
description	resets pointers if alternatives are added

9. **get_alt_data**

prototype	struct alt *get_alt_data(struct alt *, int)
description	allows the user to add or edit alternatives data, cleans up pointers

10. **draw_alt_screen**

prototype	void draw_alt_screen(void)
description	draws initial alternatives screen

11. **maxmin_check**

prototype	void maxmin_check(struct alt *, struct perf *)
description	verifies that the values entered for the performance indicators for each alternative fall between the minimum and maximum benchmark values for that indicator.

12. **perf_maxmin**

prototype	void perf_maxmin(void)
description	displays performance indicator limits for alternatives data entry

13. **alt_warn**

prototype	int alt_warn(void)
description	sets a flag to warn the user if they try to enter more than 15 alternatives

SECTION 3.3.2 - EXTERNAL SUBROUTINES

1. display_dir - resides in dedmain
2. error_check - resides in dedmain

SECTION 3.3.3 - EXTERNAL VARIABLES

1. pcount - defined in dedmain, total number of performance indicators
2. *messages - defined in dedmain, messages for the lower lefthand corner of the screen which are context sensitive
3. struct perf *begin - defined in dedmain, structure perf pointer
4. struct perf *end - defined in dedmain, struct perf pointer
5. struct alt *start - defined in dedmain, struct alt pointer
6. struct alt *last - defined in dedmain, struct alt pointer

SECTION 3.3.4 - EXTERNAL MODULES

None

SECTION 3.3.5 - GLOBAL VARIABLES

1. *menu[4] - static character pointer. Holds the pop up menu.
2. mg, ng - integer counters for the pop up menu.
3. eflag - warning. eflag is set to 0 as a default. If user tries to enter more than 15 alternatives, eflag is set to 1.

SECTION 3.4 - CALC MODULE

Allows the user to perform regular calculations or those based on a requirement or return to main menu.

SECTION 3.4.1 - SUBROUTINES

1. **pop_calc_menu**

prototype	void pop_calc_menu(void)
description	prints the pop up menu (regular or requirements based calculation) on the screen

2. **calc_menu**

prototype	int calc_menu(void)
description	allows user to select one of the two calculation routines or escape

3. **limit_ck**

prototype	void limit_ck(void)
description	verifies that the performance indicator values entered for each alternative fall within the maximum and minimum benchmark values. If they do not, the user is allowed to edit the data.

4. **draw_calc_screen**

prototype	void draw_calc_screen(void)
description	draws calculations screen if limit_ck finds a problem with the data

5. **reg_calc**

prototype	void reg_calc(void)
description	calculates the utility values for each alternative's performance indicators, performance indicator number, alternative number, performance indicator weight, alternative weighted scores, and alternative weighted index.

6. **mod_calc**

prototype	void mod_calc(void)
description	Automatically throws out those alternatives which do not meet the minimum requirement on utility curve benchmark 5. It then calculates the utility values for each of the remaining alternative's performance indicators, performance indicator number, alternative number, performance indicator weight, alternative weighted scores, and alternative weighted index.

7. **add_calc_data**

prototype	struct calc *add_calc_data(struct calc *, struct calc *)
description	sets pointers for the calc structure

8. **save_calc_data**

prototype	void save_calc_data(struct calc *, struct info *)
description	saves calculations data from memory to a file

SECTION 3.4.2 - EXTERNAL SUBROUTINES

1. save_perf_data - resides in module perf
2. save_alt_data - resides in module alt
3. get_alt_data - resides in module alt

SECTION 3.4.3 - EXTERNAL VARIABLES

1. pcount - defined in module dedmain, total number of performance indicators
2. *messages - defined in module dedmain, messages for the lower lefthand corner of the screen which are context sensitive
3. struct perf *begin - defined in module dedmain, structure perf pointer
4. struct perf *end - defined in module dedmain, struct perf pointer
5. struct alt *start - defined in module dedmain, struct alt pointer
6. struct alt *last - defined in module dedmain, struct alt pointer
7. struct calc *c_begin - defined in module dedmain, structure calc pointer
8. struct calc *c_end - defined in module dedmain, struct calc pointer

SECTION 3.4.4 - EXTERNAL MODULES

None

SECTION 3.4.5 - GLOBAL VARIABLES

1. *menu[2] - static character pointer. Holds the pop up menu.
2. mc, nc - integer counters for the pop up menu.

SECTION 3.5 - REPORT MODULE

Allows the user to display reports on the screen, print reports, or return to main menu.

SECTION 3.5.1 - SUBROUTINES

1. **draw_report_screen**
   - prototype: void draw_report_screen(void)
   - description: draws report screen with the report menu
2. **select_report**
   - prototype: int select_report(void)
   - description: allows the user to select the desired report
3. **perf_report1**
   - prototype: void perf_report1(void)
   - description: allows the user to print the performance indicator report
4. **perf_report2**
   - prototype: void perf_report2(void)
   - description: allows the user to display on the screen the performance indicator report
5. **alt_report1**
   - prototype: void alt_report1(void)
   - description: allows the user to print the alternatives report
6. **alt_report2**
   - prototype: void alt_report2(void)
   - description: allows the user to display on the screen the alternatives report
7. **alt_util_report1**
   - prototype: void alt_util_report1(void)
   - description: allows the user to print the alternatives with utility data report
8. **alt_util_report2**
   - prototype: void alt_util_report2(void)
   - description: allows the user to display on the screen the alternatives with utility data report

9. **weight_score_report1**
   - prototype: void weight_score_report1(void)
   - description: allows the user to print the alternatives weighted scores and weighted index report

10. **weight_score_report2**
   - prototype: void weight_score_report2(void)
   - description: allows the user to display on the screen the alternatives weighted scores and weighted index report

11. **screen_check**
   - prototype: int screen_check(void)
   - description: counts the number of screens required to display the alternatives report

12. **screen_check1**
   - prototype: int screen_check1(void)
   - description: counts the number of screens required to display the alternative utility and weighted scores reports

13. **page_check**
   - prototype: int page_check(void)
   - description: counts the number of pages required to print the alternatives report

14. **page_check1**
   - prototype: int page_check1(void)
   - description: counts the number of pages required to print the alternative utility and weighted scores reports

SECTION 3.5.2 - EXTERNAL SUBROUTINES

None

SECTION 3.5.3 - EXTERNAL VARIABLES

1. pcount - defined in module dedmain, total number of performance indicators
2. *messages - defined in module dedmain, messages for the lower lefthand corner of the screen which are context sensitive
3. struct perf *begin - defined in module dedmain, structure perf pointer
4. struct perf *end - defined in module dedmain, struct perf pointer

5. struct alt *start - defined in module dedmain, struct alt pointer
6. struct alt *last - defined in module dedmain, struct alt pointer
7. struct calc *c_begin - defined in module dedmain, structure calc pointer
8. struct calc *c_end - defined in module dedmain, struct calc pointer

SECTION 3.5.4 - EXTERNAL MODULES

None

SECTION 3.5.5 - GLOBAL VARIABLES

1. pflag - printer flag. pflag default is 0. When user selects printer, pflag is set to 1.

SECTION 3.6 - SETUP MODULE

Displays directory and disk information. Allows the user to enter the file name and the life-cycle cost (LCC) type or return to main menu.

SECTION 3.6.1 - SUBROUTINES

1. **read_setup**

prototype	struct info*read_setup(struct info *)
description	reads setup.dat file if one exists

2. **save_setup**

prototype	struct info*save_setup(struct info *)
description	saves the current information into the setup.dat file

SECTION 3.6.2 - EXTERNAL SUBROUTINES

None

SECTION 3.6.3 - EXTERNAL VARIABLES

1. *messages - defined in module dedmain, messages for the lower lefthand corner of the screen which are context sensitive

SECTION 3.6.4 - EXTERNAL MODULES

None

SECTION 3.6.5 - GLOBAL VARIABLES

None

SECTION 4.0 - INFINITY LIBRARY ROUTINES

SECTION 4.1 - INFINITY HEADER FILE

```
/* custom header file for infinity library routines */
/* Ken Rice 22 Jan 1989 - 23 Oct 1989 */

/* function prototypes */

void getinput(int,int,int,char *,int,int);
void border(int,int,int,int,int);
void box(int,int,int,int,int);
void colors(int,int);
void cursoff(void);
void curson(void);
void cursins(void);
void message(char *);
void open_win(int,int,int,int,int,int,int);
void close_win(int,int,int,int,int);
```

SECTION 4.2 BORDER.C

```
/* border(color, left,top, right, bottom) */
/* border a border around the specified window */
/* differs from box */

void border(lcolor,a,b,c,d)
{
        register int i,j;
        int left,right,top,bottom;

        /* convert to window coordinates */
        left=1;
        top=1;
        right=c-a;
        bottom=d-b+1;

        /* set color */
        textcolor(lcolor);

        /* draw the bottom of the box */
        gotoxy(left,bottom-1); putch('\xC8');
        for (i=0; i<(right-left); i++) putch('\xCD'); putch('\xBC');
        gotoxy(1,1);
        insline();


        /* draw top of the box */
        gotoxy(left,top);putch('\xC9');
        for (i=0; i<(right-left); i++) cprintf("\xCD"); putch('\xBB');

   /* draw sides of the box */
        gotoxy(left,top+1);
        for (i=0; i<(bottom-top-1); i++)
   {
                gotoxy(left,top+1+i); putch('\xBA');
                gotoxy(left+(right-left+1),top+1+i); putch('\xBA');
   }

}
```

SECTION 4.3 BOX.C

```
#include<conio.h>

/* box(upper left row,upper left column ,width,height) */
/* draws a box at the specified location of the specified size */

void box(int lcolor,int uplr,int uplc,int width,int height)
{
        register int i;

        /* set colors */
        textcolor(lcolor);
```

```
        /* draw top of the box */
        gotoxy(uplc,uplr);putch('\xC9');
        for (i=0; i<width; i++) cprintf("\xCD"); putch('\xBB');

        /* draw sides of the box */
        gotoxy(uplc,uplr+1);
        for (i=0; i<height-1; i++)
        {
                gotoxy(uplc,uplr+1+i); putch('\xBA');
                gotoxy(uplc+width+1,uplr+1+i); putch('\xBA');
        }


        /* draw the bottom of the box */
        gotoxy(uplc,uplr+height); putch('\xC8');
        for (i=0; i<width; i++) putch('\xCD'); putch('\xBC');

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

SECTION 4.4 COLORS.C

```
/* colors.c ... sets text and background colors */
/* Ken Rice 1988 */

void colors(background,text)
{
        textcolor(text);
        textbackground(background);
@
```

SECTION 4.5 CURSOR.C

```
/* cursor.c ... cursor control routines */
/* Ken Rice 1988 */

#include <conio.h>

void cursoff(void)                /* turns the cursor off */
{
_setcursortype(_NOCURSOR);

}

void curson(void)                 /* turns the cursor on */
{
_setcursortype(_NORMALCURSOR);

}


void cursins(void)                 /* sets a insert cursor */
{
_setcursortype(_SOLIDCURSOR);

}
```

SECTION 4.6 GETINPUT.C

```
/* get_input ... gets input in confined box */
/* returns entry */
/* Ken Rice 1988 */

#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "infinity.h"


int get_input(col,row,length,variable,background,foreground)
{
        struct text_info old;             /* old is current window info */
```

```
        int i, pos=1, status=1;
        char p[2];
        char data[80];
        char temp[79];
        struct flag {                   /* define a bit field to hold flags */
                unsigned done   : 1;
                unsigned cursor : 1;
                unsigned insflg : 1;
        } flag;
        flag.done=0;                    /* initialize flags */
        flag.cursor=0;
        flag.insflg=0;
        gettextinfo(&old);                      /* get data on current window */
        row--;                          /* row has to be one less */
        strcpy(p," ");
        strcpy(data,(char *)variable);
        window(old.winleft+col,                 /* open window for data entry */
                        old.wintop+row,
                        old.winleft+length+col-1,
                        old.wintop+row);

        textcolor(foreground);                  /* set colors in window */
        textbackground(background);
        clrscr();

        window(old.winleft+col,
                        old.wintop+row,
                        old.winleft+length+col-1,
                        old.wintop+row+1);

        curson();                               /* turn on cursor */


        /* display and string setup here */
        gotoxy(1,1);
        cprintf("%s",data);
        gotoxy(1,1);

        while(!flag.done) {
                *p=getch();
                flag.cursor=0;

                if(iscntrl(*p)) {       /* if it is a control character */
                        switch(*p) {
                                case 27:                /* escape */
                                        status=27;
                                        flag.done++;
                                break;

                                case 13:                        /* return */
                                        status=1;
                                        flag.done++;
                                break;

                                case 8:                                         /* backspace */
                                        if(strlen(data)>0) {
                                                gotoxy(1,1);
                                                for(i=1;i<=strlen(data);i++) cprintf("%s","
");

                                                if(strlen(data)+1==pos)
                                                        strcpy(&data[strlen(data)-1],"\0");
                                                else
                                                        strcpy(&data[pos-2],&data[pos-1]);
                                                gotoxy(1,1);
                                                cprintf("%s",data);
                                                gotoxy(--pos,1);
                                        }
                                break;


                                case 127:               /* ctrl backspace - clear line */
                                        gotoxy(1,1);
                                        for(i=1;i<=strlen(data);i++) cprintf("%s"," ");
                                        gotoxy(1,1);
                                        strcpy(data,"");
                                        pos=1;
                                break;
                        }
```

```
        }

        if(!*p) {       /* if it is an extended character */
                *p=getch();
                flag.cursor=1;
                switch(*p) {
                        case 72:                    /* cursor up */
                                status=-1;
                                flag.done++;
                                pos=1;
                        break;

                        case 80:                /* cursor down */
                                status=+1;
                                flag.done++;
                                pos=1;
                        break;

                        case 75:            /* cursor left */
                                if(--pos<1) pos=1;
                                gotoxy(pos,1);
                        break;

                        case 77:            /* cursor right */
                                if(++pos>length) pos=length;
                                gotoxy(pos,1);
                        break;

                        case 79:            /* end */
                                gotoxy(pos=strlen(data),1);
                        break;

                        case 82:            /* insert mode*/
                                flag.insflg=flag.insflg ? 0:1;
                                flag.insflg ? cursins():curson();
                        break;

                        case 83:            /* delete */
                                if(strlen(data)>0) {
                                        gotoxy(1,1);
                                        for(i=1;i<=strlen(data);i++) cprintf("%s","
");
                                        strcpy(&data[pos-1],&data[pos]);
                                        gotoxy(1,1);
                                        cprintf("%s",data);
                                        gotoxy(pos,1);
                                }
                        break;


                        case 59:            /* help */
                                open_win(0,18,8,62,18,CYAN,BLUE);
                                cursoff();
                                gotoxy(18,1); cprintf("EDIT HELP");
                                gotoxy(13,11); cprintf("Press any key to exit");
                                textcolor(WHITE);
                                gotoxy(3,3); cprintf("CTRL+BACKSPACE");
                                gotoxy(3,4); cprintf("DELETE");
                                gotoxy(3,5); cprintf("BACKSPACE");
                                gotoxy(3,6); cprintf("ESC");
                                gotoxy(3,7); cprintf("INSERT");
                                gotoxy(3,8); cprintf("RETURN");
                                gotoxy(3,9); cprintf("END");
                                textcolor(DARKGRAY);
                                gotoxy(18,3); cprintf("Clears line");
                                gotoxy(10,4); cprintf("Deletes character under
cursor");
                                gotoxy(13,5); cprintf("Deletes character left of
cursor");
                                gotoxy(7,6); cprintf("Aborts entry");
                                gotoxy(10,7); cprintf("Toggles insert mode");
                                gotoxy(10,8); cprintf("Accepts entry");
                                gotoxy(7,9); cprintf("Moves cursor to end of
entry");
                                getch();
                                close_win(0,18,8,62,18);
```

```
                                        window(old.winleft+col,          /* reset window
*/
                                                old.wintop+row,
                                                old.winleft+length+col-1,
                                                old.wintop+row+1);
                                        curson();
                                        textcolor(foreground);          /* set colors in
window */
                                        textbackground(background);
                                break;
                        }

                }

                if(isascii(*p) && !iscntrl(*p) && !flag.cursor) {  /* characters */
                        gotoxy(1,1);
                        if(pos>strlen(data)) strcat(data,p);      /* if appending */
                        else {
                                strcpy(temp,data);
                                strcpy(&data[pos-1],p);
                                strcat(&data[pos],&temp[pos-flag.insflg]);
                        }
                        cprintf("%s",data);
                        pos++;
                        gotoxy(pos,1);
                        if(strlen(data)==length) {
                                status=1;
                                flag.done++;
                                pos=1;
                        }
                        flag.cursor=0;
                }

        }

        cursoff();                          /* cleanup from routine */
        window(old.winleft+col,             /* reset window */
                        old.wintop+row,
                        old.winleft+length+col-1,
                        old.wintop+row);
        textattr(old.attribute);            /* reset to original color */
        clrscr();
        window(old.winleft+col,
                        old.wintop+row,
                        old.winleft+length+col-1,
                        old.wintop+row+1);
        gotoxy(1,1); cprintf("%s",data);
        window(old.winleft,                  /* reset window on exit */
                        old.wintop,
                        old.winright,
                        old.winbottom);
        strcpy((char *)variable,data);       /* copy answer into variable */
        return status;                       /* return the status */
}

                /* cursor up ... -1 */

                /* cursor down . +1 */

                /* escape ..... 0  */

                /* return ..... +1 */
```

SECTION 4.7 MESSAGE.C

```
/* message.c ... prints a red and white message at bottom of screen */
/* Ken Rice 1988 */

#include<conio.h>
#include<string.h>

char itoa(int,char *, int);

void message(char *message)
{
        int length;
        char dummy[17];
        char format[17];
```

```
        textcolor(7);
        textbackground(7);          /* set text and background to LIGHT GRAY */
        gotoxy(1,25);
        cprintf("%-39s"," ");       /* clear message area */
        textcolor(15);
        textbackground(4);          /* set text to WHITE, background to RED */
        gotoxy(1,25);
        length=strlen(message);     /* get the length of the message */
        itoa(length,dummy,2);       /* convert to ascii */
        strcpy(format,"%-.");       /* load format string */
        strcat(format,dummy);       /* concatenate message to format */
        strcat(format,"s");         /* add the end of the format specifier */
        cprintf(format,message);    /* print the message */
}
```

SECTION 4.8 WINDOWS.C

```
/* Infinity Software window routines                    */
/* Ken Rice October 23,1989                             */
/* Prototypes in infinity.h                             */

/* open_win(window #, upper left row,
                                                        upper left colum,
                                                        bottom right row,
                                                        bottom right colum,
                                                        background color,
                                                        border color)

        close_win(same as open win less color information */


#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void open_win(int,int,int,int,int,int,int);
void close_win(int,int,int,int,int);
void border(int,int,int,int,int);

char *window_name[4];

void open_win(int win_no,int x1,int y1,int x2,int y2,
                                int background,int edge)
{

        char *temp_win;
        int width,height;
        register int j;

        width=x2-x1;
        height=y2-y1;

        window_name[win_no]=(char *)malloc((width+12)*(height+12)*sizeof(char));
        temp_win=(char *)malloc((width+12)*(height+12)*sizeof(char));

        gettext(x1,y1,x2+1,y2+1,window_name[win_no]); /* for window and shadow */

        gettext(x1+1,y1+1,x2+1,y2+3,temp_win);        /* for shadow */
        for(j=1;j<((width+1)*(height+1)*2+1);j=j+2)
                *(temp_win+j)=DARKGRAY+(BLACK<<4);
        puttext(x1+1,y1+1,x2+1,y2+3,temp_win);

        window(x1,y1,x2,y2);          /* open normal window */
        textbackground(background);
        clrscr();
        border(edge,x1,y1,x2,y2);
        free(temp_win);

}


void close_win(int win_no,int x1,int y1,int x2,int y2)
{

        puttext(x1,y1,x2+1,y2+1,window_name[win_no]);
        free(window_name[win_no]);
}
```

SECTION 5.0 - UDED COMPUTER PROGRAM LISTINGS

SECTION 5.1 - HEADER FILE FOR UDED

```
/* Header file for UDED */
/* Pam Rice  14 November 1991 */

/* function prototypes */

void getinput(int,int,int,char *,int,int);      /* gets user input */
void border(int,int,int,int,int);               /* defines a border */
void box(int,int,int,int,int);                  /* defines a window box */
void colors(int,int);                           /* defines background & text colors */
void cursoff(void);                             /* turns cursor off */
void curson(void);                              /* turns cursor on */
void cursins(void);                             /* cursor insert mode */
void message(char *);                           /* defines messages */
void open_win(int,int,int,int,int,int,int);     /* opens a window and saves to memory */
void close_win(int,int,int,int,int);            /* closes window and returns previous
info */


struct perf                          /* structure for performance */
       {                                 /* criteria and utility curve data */
       char perf_name [10];              /* performance indicator name */
       unsigned int perf_rank;           /* ranking */
       unsigned int perf_rate;           /* rating */
       float perf_0;                     /* utility benchmark 0 */
       float perf_5;                     /* utility benchmark 5 */
       float perf_10;                    /* utility benchmark 10 */
       char cperf_0 [15];                /* utility benchmark 0 */
       char cperf_5 [15];                /* utility benchmark 5 */
       char cperf_10 [15];               /* utility benchmark 10 */
       float perf_wt;                    /* perf ind weight */
       float a0;                         /* x**2 constant */
       float a1;                         /* x constant */
       float a2;                         /* constant, y-intercept */
       unsigned int perf_num;            /* performance indicator number */
       struct perf *next;                /* pointer to next item in struct perf */
       struct perf *prior;               /* pointer to previous item in struct perf */
       };



struct alt                        /* structure for mutually exclusive alternatives */
       {
       char alt_name [15];              /* alternative name */
       float alt_lcc;                   /* alternative lcc */
       char calt_lcc [15];              /* alternative lcc */
       float alt_perf_crit[12];         /* perf ind for each alt */
       char calt_perf_crit[12][15];     /* perf ind for each alt */
       struct alt *next;                /* pointer to next item in struct alt */
       struct alt *prior;               /* pointer to previous item in struct alt */
       };


struct calc                     /* structure for the calculated values */
       {
       unsigned int alt_num;            /* alternative number */
       float alt_util_val[12];          /* perf ind utility value for each alt  */
       float alt_wt_score[15];          /* perf ind weighted score for each alt */
       float alt_wt_ind;                /* weighted index for each alt */
       char altname [15];               /* alternative name */
       float altlcc;                    /* alternative lcc */
       struct calc *next;               /* pointer to next item in struct calc */
       struct calc *prior;              /* pointer to previous item in struct calc */
       };

struct info                     /* holds data on system set up */
       {
       char name [9];                   /* file name */
       char drive [2];                  /* drive being used */
       char directory [64];             /* directory being used */
```

```
        char alt_lcc_type [4];          /* alternative lcc type */
        };
```

SECTION 5.2 -DEDMAIN.C

```
/* DEDMAIN.C ... This is the main program for the Utility Decision */
/* Evaluation Display program */
/* Pam Rice   14 November 1991 */

#include "ded.h"          /* ded structure header file */
#include <stdio.h>                  /* standard input/output header file */
#include <dos.h>                    /* dos header file */
#include <stdlib.h>                 /* standard library header file */
#include <conio.h>                  /* color input/output header file */
#include <dir.h>                    /* directory header file */
#include <string.h>                 /* string manipulation header file */
#include <alloc.h>                  /* far definition header file */
#include <process.h>                /* process control header file */

#define mono (int far *)0xb0000000L  /* monochrome display */
#define cga (int far *)0xb8000000L   /* EGA, CGA or VGA display */


/* declare function prototypes */

void draw_screen(void);             /* draw main menu */
void get_disk_info(void);           /* get directory info */
void display_dir(void);             /* print directory info */
void report(struct info *);         /* report module */
void error_check(void);             /* basic error-checking routine */

int perf_cnt(struct perf *);        /* counter for # of perf indicators */
int menu_select(void);              /* menu selection routine */

struct info *perf(struct info *);   /* perf module */
struct info *alt(struct info *);    /* alt module */
struct info *calc(struct info *);   /* calc module */
struct info *setup(struct info *);          /* setup module */
struct info *read_setup(struct info *);     /* loads prior setup */

struct perf *load_perf_data(char *, struct perf *);     /* loads prior perf data */
struct alt *load_alt_data(char *, struct alt *);        /* loads prior alt data */

/* declare global variables */
int pcount;                         /* holds number of performance indicators */

struct dfree disk_table;            /* holds disk info */

struct perf perfind;                /* struct perf */
struct perf *begin;                 /* struct perf pointer */
struct perf *end;                   /* struct perf pointer */

struct alt altind;                  /* struct alt */
struct alt *start;                  /* struct alt pointer */
struct alt *last;                   /* struct alt pointer */

struct calc calcind;                /* struct calc */
struct calc *c_begin;               /* struct calc pointer */
struct calc *c_end;                 /* struct calc pointer */

char dir[64];                       /* holds directory name */

/* char *messages[]={                    messages for bottom of screen */
/*      "  F1-HELP  F2-MENU  ESC-EXIT ",                                  */
/*      "  F1-HELP  ",                                                    */
/*      "  F1-HELP  \x18 \x19 TO SCROLL  \x1B\xD9 TO SELECT",             */
/*      "  F1-HELP  ESC TO ABANDON ENTRY",                                */
/*      "  F1-HELP  ESC-EXIT",                                            */
/*      "  F1-HELP  ESC-EXIT  PGDN-MORE  PGUP-PREVIOUS",                 */
/*      "  F1-HELP  F3-PRINT TOGGLE                  "                   */

char *messages[]={                  /* messages for bottom of screen */
        "  F2-MENU  ESC-EXIT ",
        "  F1-HELP  ",
        "  \x18 \x19 TO SCROLL  \x1B\xD9 TO SELECT",
        "  ESC TO ABANDON ENTRY",
        "  ESC-EXIT",
        "  ESC-EXIT  PGDN-MORE  PGUP-PREVIOUS",
        "  F3-PRINT TOGGLE "
```

```
};


/************************************************/


int main(void)            /* dedmain */
{

        struct info *defaults;  /* struct info pointer */
        char curdir[64];        /* directory info */
        char curdrv[2];         /* drive info */

        start=last=NULL;                    /* initialize pointers for alt */
        begin=end=NULL;                     /* initialize pointers for perf */
        c_begin=c_end=NULL;           /* initialize pointers for calc */

        getcurdir(0,curdir);          /* initialize default info */
        curdrv[0]='A'+getdisk();
        curdrv[1]=0;
        defaults=(struct info*)malloc(sizeof(struct info));  /* define size of struct
info */
        strcpy(defaults->name,"NONE");
        strcpy(defaults->drive,curdrv);
        strcpy(defaults->directory,curdir);
        read_setup(defaults);   /* load defaults if any exist */
        cursoff();
        end=load_perf_data(defaults->name,begin);           /* load perf data */
        last=load_alt_data(defaults->name,start);           /* load alt data */
        pcount=perf_cnt(begin);                             /* count perf ind */
        get_disk_info();                                 /* get disk info */

        for(;;) {
                draw_screen();                  /* draw defaults screen */
                display_dir();                  /* draw directory info */

                switch(menu_select()) {         /* main menu selection */

                        case 1:                         /* perf module */
                                perf(defaults);
                                break;

                        case 2:                         /* alt module */
                                alt(defaults);
                                break;

                        case 3:                         /* calc module */
                                calc(defaults);
                                break;

                        case 4:                         /* report module */
                                report(defaults);
                                break;

                        case 5:
                                setup(defaults);                            /* change setup
*/
                                read_setup(defaults);                       /* read changes
*/
                                begin=end=NULL;
                                start=last=NULL;
                                end=load_perf_data(defaults->name,begin);    /* load perf
data */
                                last=load_alt_data(defaults->name,start);    /* load alt
data */
                                pcount=perf_cnt(begin);                       /* count perf
ind */
                                display_dir();                              /* display
directory info */
                                break;

                        case 6:                         /* exit */
                                curson();
                                exit(0);

                        default:                      /* error check */
```

```
                                error_check();
                                window(15,4,64,13);
                        }
                }
}                                       /* end of main progran */

/*********************************************************************/

menu_select(void)                       /* select a menu operation */
{
        int m=4,n=0;
        char move;

        static char *menu[6]={
                "  1. Performance Criteria/Utility curves  ",
                "  2. Alternatives  ",
                "  3. Calculation    ",
                "  4. Reports     ",
                "  5. Setup      ",
                "  6. Quit       ",
        };


        window(15,4,64,13);             /* open window for menu */
        colors(BLUE,YELLOW);
        clrscr();
        border(WHITE,15,4,64,13);
        gotoxy(24,1);
        cprintf("MENU");
        textcolor(YELLOW);
        for(n=1;n<6;n++) {
                gotoxy(2,m++); cprintf("%s",menu[n]);   /* print menu */
        }
        colors(CYAN,WHITE);               /* hilite the first item */
        gotoxy(2,3);
        cprintf("%s",menu[0]);
        m=3;n=0;                        /* reset counters to first item */


        while((move=getch()) !=13)      /* check for return */
        {

                if( move==0)                /* check for extended code */
                {
                        switch(getch())
                        {
                                case 80:                                /* down cursor */
                                        colors(BLUE,YELLOW);
                                        gotoxy(2,m); cprintf("%s",menu[n++]);
                                        if(n>5) {m=2;n=0;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,++m); cprintf("%s",menu[n]);
                                        break;

                                case 72:                                 /* up cursor */
                                        colors(BLUE,YELLOW);
                                        gotoxy(2,m); cprintf("%s",menu[n--]);
                                        if(n<0) {m=9;n=5;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,--m); cprintf("%s",menu[n]);
                                        break;

                                case 59:
                                        break;

                                default:
                                        error_check();
                                        window(15,4,64,13);

                        }

                }
                if(move>48&&move<55) {          /* if number is pressed */
                        window(1,1,80,25);          /* reset window to full screen */
                        return(move-48);            /* return number */
                }

        }
```

```
        window(1,1,80,25);              /* reset window to full screen */

        return(++n);
}                                   /* end of menu select */

/*************************************************************************/

void get_disk_info()
{
        getcwd(dir,64);                     /* get current directory */
        getdfree(0,&disk_table);            /* get disk info */
}

/*************************************************************************/

void display_dir()
{
        colors(LIGHTGRAY,BLACK);
        gotoxy(50,25);
        cprintf("%s",dir);                          /* print current directory */
}


/*************************************************************************/
void draw_screen(void) {            /* routine to draw initial screen */
        textbackground(LIGHTGRAY);
        clrscr();
        box(WHITE,1,1,78,2);
        gotoxy(15,2); cprintf("UTILITY DECISION EVALUATION DISPLAY PROGRAM");
        colors(LIGHTGRAY,RED);
        gotoxy(70,2);
        cprintf("Ver. 1.0");
/*      gotoxy(2,76); message(messages[1]);     */
}

/*******************************************************/
struct perf *load_perf_data(char *rfile,struct perf *info)
{                           /* routine to load perf data */
        FILE *fp;
        struct perf *temp;
        char file[14];
        temp=NULL;
        strcpy(file,rfile);
        strcat(file,".PRF");
        if(!strcmp(file,"NONE.PRF")) return NULL;
        if((fp=fopen(file,"rb"))==NULL)  {
                printf("Cannot open file\n");
                return NULL;
        }
        begin=(struct perf *)malloc(sizeof(struct perf));
        info=begin;
        while(!feof(fp))  {
                if((fread(info,sizeof(struct perf),1,fp)!=1)) break;
                info->next=(struct perf*)malloc(sizeof(struct perf));
                if(!info->next)  {
                        printf("Out of memory\n");
                        return NULL;
                }
                info->prior=temp;
                temp=info;
                info=info->next;
        }
        temp->next=NULL;                /* last entry */
        end=temp;
        begin->prior=NULL;
        fclose(fp);
        return end;
}                                               /* end of load perf data */

/*************************************************************/
void error_check(void)              /* menu error checking routine */
{
        open_win(1,40,15,75,20,BLUE,YELLOW);
        textcolor(WHITE);
        gotoxy(3,2); cprintf("Enter only selections listed");
        gotoxy(3,4); cprintf("  Press RETURN to continue");
        if(getch()==13) {                   /* if return is pressed */
                close_win(1,40,15,75,20);
```

```
        }
        window(1,1,80,25);
        return;
}                                                   /* end of error checking routine */

/*********************************************************************/
struct alt *load_alt_data(char *rfile,struct alt *info)
{                                   /* load alt data */
        FILE *fp;
        struct alt *temp;
        char file[14];
        temp=NULL;
        strcpy(file,rfile);
        strcat(file,".ALT");
        if(!strcmp(file,"NONE.ALT")) return NULL;
        if((fp=fopen(file,"rb"))==NULL)  {
                printf("Cannot open file\n");
                return NULL;
        }
        start=(struct alt *)malloc(sizeof(struct alt));
        info=start;
        while(!feof(fp))  {
                if((fread(info,sizeof(struct alt),1,fp)!=1)) break;
                info->next=(struct alt*)malloc(sizeof(struct alt));
                if(!info->next)  {
                        printf("Out of memory\n");
                        return NULL;
                }
                info->prior=temp;
                temp=info;
                info=info->next;
        }
        temp->next=NULL;                /* last entry */
        last=temp;
        start->prior=NULL;
        fclose(fp);
        return last;
}                                           /* end of load alt dat */

/*********************************************************************/
int perf_cnt(struct perf *pcnt)             /* counts # of perf indicators */
{
        int pcount=0;
        struct perf*temp;
        temp=pcnt;
        while(temp)  {
                pcount+=1;
                temp=temp->next;
                }
        pcount--;
        return pcount;
}                                           /* end of perf_cnt */

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

SECTION 5.3 - PERF.C

```
/* PERF.C ... This is the performance indicator and */
/* utility curve module for the Utility Decision */
/* Evaluation Display program */
/* Pam Rice   13 October 1991 */

#include "ded.h"                        /* ded header file */
#include <stdio.h>                      /* standard i/o header file */
#include <dos.h>                        /* dos header file */
#include <stdlib.h>                     /* standard library header file */
#include <conio.h>                      /* color i/o header file */
#include <dir.h>                        /* directory header file */
#include <string.h>                     /* string control header file */
#include <math.h>                       /* math header file */

#define MAXPERF 12                       /* maximum number of perf ind */

/* declare function prototypes */

void draw_perf_screen(void);            /* draw initial screen */
void display_dir(void);                 /* display directory info */
void pop_perf_menu(void);               /* pop up menu */
void save_perf_data(struct perf *,struct info *);    /* save perf data */
```

```
void disp_perf(struct perf *,int);     /* display perf data */
void error_check(void);                /* error check routine */

char itoa(int,char *, int);            /* integer to ascii conversion */

int alt_add(int);                      /* alternatives check */
int alt_edit(void);                    /* alternatives check */
int alt_del(struct perf *);            /* alternatives check */
int perf_menu(void);                   /* perf ind menu */
int get_input(int,int,int,char *,int,int);     /* get user input */


struct perf *add_perf_data(struct perf *,struct perf *);  /* add data */
struct perf *scroll_perf(struct perf *);                  /* scroll thru data */
struct perf *delete_perf(struct perf *);                  /* delete data */
struct perf *get_perf_data(struct perf *,int);            /* get perf data input */
struct perf *display_perf_data(struct perf *);            /* display perf data */

/* define external relationships */
extern int pcount;                                        /* perf ind count */
extern char *messages[];                                  /* messages */
extern struct perf *begin;                                /* struct perf pointer */
extern struct perf *end;                                  /* struct perf pointer */
extern struct alt *start;                                 /* struct alt pointer */
extern struct alt *last;                                  /* struct alt pointer */


/* define global variables */
static char *menu[4]={                                    /* pop up menu */
       " ADD DATA ",
       " EDIT DATA ",
       " DELETE DATA ",
       " SAVE CHANGES "
};

int m,n;                                                  /* counters for menu */


/*********************************************************************/

struct info *perf(struct info *defaults)        /* perf module */
       {
              int k;
              int check;
              char input;
              struct perf *temp;            /* struct perf pointer */
              struct perf *selection;       /* struct perf pointer */
              draw_perf_screen();                        /* draw initial screen */
              display_dir();                                    /* display directory
info */
              if(!strcmp(defaults->name,"NONE"))   /* is file picked? */
              {
                     gotoxy(27,10);
                     colors(RED,WHITE);
                     cprintf("  NO FILE SPECIFIED    ");
                     gotoxy(27,11);
                     cprintf(" ESC and go to Setup ");
              }
              else  {                          /* if yes then display */
                     gotoxy(2,1);
              textcolor(BLACK);
              cprintf("[PERF:%-8s]",defaults->name);
                     display_perf_data(begin);                  /* display any
existing data */
              }
              while((input=getch()) !=27)          /* while not escape */
              {
              if(input==0)                                                /* check for
extended code */
              {
              switch(getch())
                     {
                     case 59:
       /* F1 Help key */
                            gotoxy(10,10);
                            printf("HELP");
                     break;
```

```
                    case 60:
        /* F2 menu key */
                            open_win(3,2,5,17,11,MAGENTA,WHITE);
                            pop_perf_menu();
        /* pop up perf */
                            while ((k=perf_menu()) !=5 && k!=0)
                                    {
                /* while not exit, loop */
                                    switch(k)
                                            {
                                            case 1:                                 /* add
*/
                                                    check=alt_add(pcount);          /*
warning of alt mods */
                                                    if(check!=999)  {
                                                            pcount=check;           /*
reset pcount */
                                                            temp=(struct perf
*)malloc(sizeof(struct perf));
                                                            if(get_perf_data(temp,1))
{               /* if not escape */
        begin=add_perf_data(temp,begin);    /* reset pointers */
        close_win(3,2,5,17,11);
        display_perf_data(begin);           /* display data */
        open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                                    pop_perf_menu();
/* pop up menu */
                                                            }
                                                    }
                                                    break;

                                            case 2:             /* edit */
                                                    check=alt_edit();
                /* warning of alt mods */
                                                    if(check!=999)  {
                                                            close_win(3,2,5,17,11);
        selection=scroll_perf(begin); /* pick data to edit */
                                                            if(selection)
                                                            get_perf_data(selection,0);
                                                            display_perf_data(begin);
/* display data */
        open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                            pop_perf_menu();
/* pop up menu */
                                                    }
                                                    break;

                                            case 3:             /* delete */
                                                    close_win(3,2,5,17,11);
                                                    selection=scroll_perf(begin);
                                                    check=alt_del(selection);
                /* alt mods */
                                                    if(check!=999)  {
                                                            pcount=check;
/* reset pcount */
                                                            if(selection)
        begin=delete_perf(selection); /* reset pointers */
                                                            display_perf_data(begin);
/* display data */
        open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                            pop_perf_menu();
/* pop up menu */
                                                    }
                                                    break;

                                            case 4:             /* save */
                                                    save_perf_data(begin,defaults);
                                                    close_win(3,2,5,17,11);
                                                    display_perf_data(begin);
```

```
                                                open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                pop_perf_menu();
                                                break;

                                        default:
        /* error check */
                                                error_check();
                                        }
                                }

                        close_win(3,2,5,17,11);       /* close options window */
                        window(1,1,80,25);
                        message(messages[0]);
                        break;

                default:
                        error_check();
                }
        }
        }
        window(1,1,80,25);
        return(0);
}                                       /* end of function perf */

/*************************************************************************/
struct perf *scroll_perf(struct perf *ptr)       /* scroll thru data */
{
        int i,count=0,temp,rec=0,rec_count;
        char input;
        struct perf *perf_array[MAXPERF];

        window(1,1,80,25);                      /* set up screen          */
        message(messages[2]);
        window(2,4,79,23);
        colors(LIGHTGRAY,BLACK);
        clrscr();

        while(ptr) {                            /* fill array of pointers */
                perf_array[count++]=ptr;
                ptr=ptr->next;                       /* and count them         */
        }
        rec_count=count;
        if(count>18) {                          /* if more than 1 screen  */
                rec=count-18;                        /* take last 18           */
                count=18;                            /* limit display          */
        }
        for(i=1;i<count;i++)
                disp_perf(perf_array[rec++],i);      /* display initial data   */

        colors(CYAN,BLACK);                     /* hilite the last record */
        gotoxy(1,count); cprintf("%77s"," ");
        disp_perf(perf_array[rec],count);


        /* If an extended character is picked it will start off with 00. */
        /* The following code throws out this 00 and gets the key that   */
        /* is picked.  If ESC (27) or Enter (13) is picked it takes      */
        /* a different course of action.                                 */

        while((input=getch())==0 && input!=27 && input!=13) {
                switch(getch()) {
                        case 72:                             /* cursor up              */
                                colors(LIGHTGRAY,BLACK);     /* unlite last entry      */
                                disp_perf(perf_array[rec],count);
                                if(count==1) {               /* top of screen          */
                                        rec--;
                                        if(rec<0) rec=0;
                                        else insline();
                                }
                                else
                                        disp_perf(perf_array[rec--],count--);
                                colors(CYAN,BLACK);
                                disp_perf(perf_array[rec],count);
                        break;

                        case 80:
        /* cursor down */
                                if(rec==rec_count-1) break;
```

```
                colors(LIGHTGRAY,BLACK);        /* unlite last entry      */
                disp_perf(perf_array[rec],count);
                if(count==18) {         /* bottom of screen */
                        rec++;
                        if(rec>rec_count-1) rec=rec_count-1;
                        else {
                                gotoxy(1,1);
                                delline();
                        }
                }
                else
                        disp_perf(perf_array[rec++],count++);
                colors(CYAN,BLACK);
                disp_perf(perf_array[rec],count);
        break;

        case 79:
/* end of list */
                if((rec_count-1)>18) {
                        rec=rec_count-18;
                        count=18;
                }
                else {
                        rec=0;
                        count=rec_count;
                }
                colors(LIGHTGRAY,BLACK);
                for(i=1;i<count;i++)
                        disp_perf(perf_array[rec++],i);
                colors(CYAN,BLACK);
                gotoxy(1,count); cprintf("%77s"," ");
                disp_perf(perf_array[rec],count);
        break;

        case 71:
/* top of list */
                rec=0;
                count=1;
                colors(LIGHTGRAY,BLACK);
                temp=rec_count>18?19:rec_count+1;
                for(i=1;i<temp;i++)
                        disp_perf(perf_array[i-1],i);
                colors(CYAN,BLACK);
                gotoxy(1,count); cprintf("%77s"," ");
                disp_perf(perf_array[rec],count);
        break;

        case 73:
/* page up */
                if(rec_count<19) break;
                rec-=36;
                if(rec<0) rec=0;
                colors(LIGHTGRAY,BLACK);
                for(i=1;i<19;i++)
                        disp_perf(perf_array[rec++],i);
                colors(CYAN,BLACK);
                gotoxy(1,count); cprintf("%77s"," ");
                rec+=count-19;
                disp_perf(perf_array[rec],count);
        break;

        case 81:
/* page down */
                if(rec_count<19) break;
                rec+=36;
                if(rec>rec_count-1) rec=rec_count-1;
                rec=rec-17;
                colors(LIGHTGRAY,BLACK);
                for(i=1;i<19;i++)
                        disp_perf(perf_array[rec++],i);
                colors(CYAN,BLACK);
                gotoxy(1,count); cprintf("%77s"," ");
                rec+=count-19;
                disp_perf(perf_array[rec],count);
        break;
```

```
                case 59:
            /* help */
                        open_win(0,18,8,62,17,CYAN,BLUE);
                        gotoxy(18,1); cprintf("SCROLL HELP");
                        gotoxy(13,10); cprintf("Press any key to exit");
                        textcolor(WHITE);
                        gotoxy(3,3); cprintf("Cursor Up");
                        gotoxy(3,4); cprintf("Cursor Down");
                        gotoxy(3,5); cprintf("Page Up");
                        gotoxy(3,6); cprintf("Page Down");
                        gotoxy(3,7); cprintf("HOME");
                        gotoxy(3,8); cprintf("END");
                        textcolor(DARKGRAY);
                        gotoxy(13,3); cprintf("Moves up one record");
                        gotoxy(15,4); cprintf("Moves down one record");
                        gotoxy(11,5); cprintf("Moves up one screen");
                        gotoxy(13,6); cprintf("Moves down one screen");
                        gotoxy(8,7);  cprintf("Moves to beginning of all
records");
                        gotoxy(7,8);  cprintf("Moves to end of all records");
                        getch();
                        close_win(0,18,8,62,17);
                        window(2,4,79,23);
                    break;

            }
        }
        if(input==13) return(perf_array[rec]);      /* if CR return pointer */
        return(0);                                  /* else return */
}

/*************************************************************/

struct perf *delete_perf(struct perf * ptr)    /* delete */
{
        struct perf *temp;
        open_win(1,25,10,54,15,BLUE,YELLOW);
        textcolor(WHITE);
        gotoxy(3,2); cprintf("Press RETURN to DELETE");
        gotoxy(3,5); cprintf("ESC to ABORT");
        textcolor(YELLOW);
        gotoxy(3,3); cprintf("%s",ptr->perf_name);
        if(getch()==13) {          /* if return is pressed */

                close_win(1,25,10,54,14);
                if(!ptr->next && !ptr->prior) {  /* only one record */
                        begin=end=NULL;
                        return(begin);
                }

                if(!ptr->prior) {                /* first record */
                        begin=ptr->next;
                        ptr->next->prior=NULL;
                        return(begin);
                }

                if(!ptr->next) {                 /* last record */
                        end=ptr->prior;
                        ptr->prior->next=NULL;
                        return(begin);
                }


                temp=ptr->prior;                 /* middle record */
                temp->next=ptr->next;
                ptr->next->prior=temp;
                return(begin);

        }
        pcount++;
        close_win(1,25,10,54,15);
        window(1,1,80,25);
        return(begin);                      /* return new pointer */
}                                   /* end of delete */

/*********************************************************************/
void disp_perf(struct perf *ptr, int i)    /* display data for scrolling */
{
```

```
        gotoxy(1,i);  cprintf("%77s"," ");
        gotoxy(5,i);  cprintf("%s",ptr->perf_name);
        gotoxy(20,i); cprintf("%2u",ptr->perf_rank);
        gotoxy(28,i); cprintf("%3u",ptr->perf_rate);
        if((fabs(ptr->perf_0)<.01) || (fabs(ptr->perf_0)>999999.99)) {
                gotoxy(36,i); cprintf("%10.2e",ptr->perf_0);}
        else {
                gotoxy(36,i); cprintf("%10.2f",ptr->perf_0);}
        if((fabs(ptr->perf_5)<.01) || (fabs(ptr->perf_5)>999999.99)) {
                gotoxy(53,i); cprintf("%10.2e",ptr->perf_5);}
        else {
                gotoxy(53,i); cprintf("%10.2f",ptr->perf_5);}
        if((fabs(ptr->perf_10)<.01) || (fabs(ptr->perf_10)>999999.99)){
                gotoxy(67,i); cprintf("%10.2e",ptr->perf_10);}
        else {
                gotoxy(67,i); cprintf("%10.2f",ptr->perf_10);}

        return;
}                                               /* end of display */
/*********************************************************************/
void save_perf_data(struct perf *ptr,struct info *defaults)
{                                               /* save perf data */
        FILE *fp;
        char *perf_file;
        struct perf *info;
        info=ptr;
        perf_file=(char *)malloc(14);
        strcpy(perf_file,defaults->name);
        strcat(perf_file,".PRF");
        window(19,8,46,10);
        textbackground(BLACK);
        clrscr();
        window(18,7,45,9);
        textbackground(MAGENTA);
        clrscr();
        border(WHITE,18,7,45,9);
        textcolor(YELLOW);
        gotoxy(3,2); cprintf("SAVING FILE:");
        colors(CYAN,BLACK);
        gotoxy(15,2); cprintf(" %s",perf_file);
        if((fp=fopen(perf_file,"wb"))==NULL) {
                cprintf("Cannot open file\n");
                exit(1);
        }
        while(info) {
                fwrite(info,sizeof(struct perf),1,fp);
                info=info->next;
        }
        fclose(fp);
        return;
}                                                   /* end of perf data */


/*********************************************************************/
void pop_perf_menu(void)                    /* pops up perf menu */
{
        int i;
        window(1,1,80,25);
        message(messages[4]);
        window(2,5,17,11);
        colors(MAGENTA,YELLOW);

        for(i=0;i<4;i++){
                gotoxy(2,i+2);
                cprintf("%s",menu[i]);              /*print menu*/
        }
        colors(CYAN,WHITE);
        gotoxy(2,2);
        cprintf("%s",menu[0]);
        m=2;n=0;                                    /* hilite first entry */
}                                                 /* end of pop up menu */

/*********************************************************************/

int perf_menu(void)                       /* gets performance indicators input */
{
        int move;
```

```
        while((move=getch()) !=13)          /* check for return */
        {
                if( move==0)                        /* check for extended code */
                {
                        switch(getch())
                        {
                                case 80:                                /* down cursor */
                                        colors(MAGENTA,YELLOW);
                                        gotoxy(2,m++); cprintf("%s",menu[n++]);
                                        if(n>3) {m=2;n=0;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,m); cprintf("%s",menu[n]);
                                        break;

                                case 72:                                /* up cursor */
                                        colors(MAGENTA,YELLOW);
                                        gotoxy(2,m--); cprintf("%s",menu[n--]);
                                        if(n<0) {m=5;n=3;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,m); cprintf("%s",menu[n]);
                                        break;
                        }
                }
                if (move==27) return(0);                /* escape */

        }
        return(n+1);                                    /* return selection */
}

/*************************************************************************/
struct perf *display_perf_data(struct perf *ptr) /* display perf data */

{
        int i=1;
        colors(LIGHTGRAY,BLACK);
        window(2,4,79,23);
        clrscr();
        while(ptr && i<19) {                /* print data */

                gotoxy(5,i);  cprintf("%s",ptr->perf_name);
                gotoxy(20,i); cprintf("%2u",ptr->perf_rank);
                gotoxy(28,i); cprintf("%3u",ptr->perf_rate);
                if((fabs(ptr->perf_0)<.01) || (fabs(ptr->perf_0)>999999.99)) {
                        gotoxy(36,i); cprintf("%10.2e",ptr->perf_0);}
                else {
                        gotoxy(36,i); cprintf("%10.2f",ptr->perf_0);}
                if((fabs(ptr->perf_5)<.01) || (fabs(ptr->perf_5)>999999.99)) {
                        gotoxy(53,i); cprintf("%10.2e",ptr->perf_5);}
                else {
                        gotoxy(53,i); cprintf("%10.2f",ptr->perf_5);}
                if((fabs(ptr->perf_10)<.01) || (fabs(ptr->perf_10)>999999.99)){
                        gotoxy(67,i++); cprintf("%10.2e",ptr->perf_10);}
                else {
                        gotoxy(67,i++); cprintf("%10.2f",ptr->perf_10);}

                ptr=ptr->next;                  /* move to next pointer value */
        }

        window(1,1,80,25);
        return 0;
}

/*************************************************************************/
struct perf *add_perf_data(struct perf *ptr, struct perf *top)
{                                       /* reset pointers after add */

        if(end==NULL) {         /* list is empty, this is the first */
                end=ptr;
                ptr->prior=NULL;
                ptr->next=NULL;
                return(ptr);
        }                               /* else add data onto end of list */

        end->next=ptr;
        ptr->prior=end;         /* prior is previous end of file  */
        end=ptr;                /* new end is ptr */
        ptr->next=NULL;
        return(top);
```

```
}                                       /* end of function add */

/*************************************************************************/
struct perf *get_perf_data(struct perf *ptr,int flag)
{

        /* user data entry */
      unsigned int j=1,k=1;
      char temp[5];
      window(1,1,80,25);
      message(messages[3]);
      open_win(1,13,5,59,13,MAGENTA,WHITE);    /* print titles */
      gotoxy(3,2);  cprintf("PERFORMANCE INDICATOR:");
      gotoxy(3,3);  cprintf("RANK:");
      gotoxy(3,4);  cprintf("RATING:");
      gotoxy(3,5);  cprintf("BENCHMARK 0:");
      gotoxy(3,6);  cprintf("BENCHMARK 5:");
      gotoxy(3,7);  cprintf("BENCHMARK 10:");

      if(flag) {                              /* not in edit mode          */
            strcpy(ptr->perf_name," ");       /* initialize variables      */
            ptr->perf_rank=0;
            ptr->perf_rate=0;
            ptr->perf_0=0.0;
            ptr->perf_5=0.0;
            ptr->perf_10=0.0;
            strcpy(ptr->cperf_0," ");
            strcpy(ptr->cperf_5," ");
            strcpy(ptr->cperf_10," ");
            ptr->perf_wt=0.0;
            ptr->perf_num=0;
      }
      else {                                  /* print data to edit        */
            textcolor(WHITE);
            gotoxy(26,2); cprintf("%-10s",ptr->perf_name);
            gotoxy(10,3);  cprintf("%u",ptr->perf_rank);
            gotoxy(11,4);  cprintf("%u",ptr->perf_rate);
            gotoxy(16,5); cprintf("%-15s",ptr->cperf_0);
            gotoxy(16,6); cprintf("%-15s",ptr->cperf_5);
            gotoxy(17,7); cprintf("%-15s",ptr->cperf_10);
      }
      textcolor(WHITE);

      while(j && j!=27) {                     /* keep going unless j=0 or ESC */
            switch(k) {
                  case 1:                             /* perf name */
                        j=get_input(25,2,9,ptr->perf_name,CYAN,WHITE);
                        k=(j<0) ? 1:2;
                        break;

                  case 2:                             /* perf rank */
                        itoa(ptr->perf_rank,temp,10);
                        j=get_input(9,3,3,temp,CYAN,WHITE);
                        ptr->perf_rank=atoi(temp);
                        strcpy(temp,0);
                        k+=j;
                        break;

                  case 3:                             /* perf rating */
                        itoa(ptr->perf_rate,temp,10);
                        j=get_input(10,4,4,temp,CYAN,WHITE);
                        ptr->perf_rate=atoi(temp);
                        strcpy(temp,0);
                        k+=j;
                        break;

                  case 4:                             /* benchmark 0 */
                        j=get_input(15,5,15,ptr->cperf_0,CYAN,WHITE);
                        ptr->perf_0=atof(ptr->cperf_0);
                        k+=j;
                        break;

                  case 5:                             /* benchmark 5 */
                        j=get_input(15,6,15,ptr->cperf_5,CYAN,WHITE);
                        ptr->perf_5=atof(ptr->cperf_5);
                        k+=j;
                        break;
```

```
                        case 6:                             /* benchmark 10 */
                                j=get_input(16,7,15,ptr->cperf_10,CYAN,WHITE);
                                ptr->perf_10=atof(ptr->cperf_10);
                                j=0;
                                break;


                }

        }
        close_win(1,13,5,59,13);
        window(1,1,80,25);
        message(messages[0]);
        window(2,5,17,11);
        if(j==27) {
         --pcount;
         return(0);}
        else return(ptr);                   /* return pointer */
}                                       /* end of function get perf data */

/*************************************************************************/

        /* draw default perf screen */
void draw_perf_screen(void)
{

        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(2,1); cprintf("[PERF: NONE]");
        gotoxy(36,1);
        cprintf("PERF DATA");
        textcolor(WHITE);

        gotoxy(4,2); cprintf("PERFORMANCE");    /* print titles */
        gotoxy(5,3); cprintf("INDICATOR");
        gotoxy(20,3); cprintf("RANK");
        gotoxy(28,3); cprintf("RATING");
        gotoxy(43,2); cprintf("UTILITY CURVE BENCHMARKS");
        gotoxy(42,3); cprintf("0");
        gotoxy(58,3); cprintf("5");
        gotoxy(72,3); cprintf("10");

        colors(RED,WHITE);
        window(1,1,80,25);
        message(messages[0]);

}                                       /* end of draw perf screen */

/*************************************************************/
int alt_add(check)                      /* alternatives check for add */
{
        struct alt *ptr;                    /* if no alt data, increment pcount */
        if(last==NULL)  {                   /* no message generated */
                ++check;
                if(check==12)  {
                        close_win(3,2,5,17,11);
                        window(1,1,80,25);
                        check=999;
                        open_win(3,14,10,44,16,RED,WHITE);
                        gotoxy(3,2);  cprintf("Maximum number of         ");
                        gotoxy(3,3);  cprintf("Performance Indicators    ");
                        gotoxy(3,4);  cprintf("have already been entered.");
                        gotoxy(3,5);  cprintf("Press RETURN to continue. ");
                        if(getch()==13) {
                                close_win(3,14,10,44,16);
                                window(1,1,80,25);
                                open_win(3,2,5,17,11,MAGENTA,YELLOW);
                                pop_perf_menu();
                                return(check);}
                }
                return(check);
        }
        ++check;
        if(check==12)  {
                close_win(3,2,5,17,11);
```

```
                window(1,1,80,25);
                check=999;
                open_win(3,14,10,44,16,RED,WHITE);
                gotoxy(3,2);  cprintf("Maximum number of         ");
                gotoxy(3,3);  cprintf("Performance Indicators    ");
                gotoxy(3,4);  cprintf("have already been entered.");
                gotoxy(3,5);  cprintf("Press RETURN to continue. ");
                if(getch()==13) {
                        close_win(3,14,10,44,16);
                        window(1,1,80,25);
                        open_win(3,2,5,17,11,MAGENTA,YELLOW);
                        pop_perf_menu();
                        return(check);}
        }
        --check;
        open_win(1,30,10,68,17,BLUE,YELLOW);    /* else */
        textcolor(WHITE);
        gotoxy(3,2); cprintf("Adding performance indicators");
        gotoxy(3,3); cprintf("will require adding data to the");
        gotoxy(3,4); cprintf("alternatives (.ALT) file.");
        gotoxy(3,6); cprintf("Press RETURN to continue");
        gotoxy(3,7); cprintf("Press ESC to abort");
        if(getch()==13)  {                    /* if return is pressed */
                check++;
                ptr=start;
                while(ptr) {            /* allow for another perf ind value to be added
*/
                        strcpy(ptr->calt_perf_crit[check]," ");
                        ptr->alt_perf_crit[check]=0.0;
                        ptr=ptr->next;
                }
                close_win(1,30,10,68,17);
                window(1,1,80,25);
                window(2,5,17,11);
                return check;
                }
        check=999;              /* escape flag */
        close_win(1,30,10,68,17);
        window(1,1,80,25);
        window(2,5,17,11);
        return check;
}
/*************************************************************/
int alt_edit(void)            /* alternatives check for edit */
{

        if(last==NULL)  {          /* if no alt data, return */
                return 0;}
        open_win(1,30,10,65,17,BLUE,YELLOW);  /* else */
        textcolor(WHITE);
        gotoxy(3,2); cprintf("Editing performance indicators");
        gotoxy(3,3); cprintf("may require modifications to the");
        gotoxy(3,4); cprintf("alternatives (.ALT) file.");
        gotoxy(3,6); cprintf("Press RETURN to continue");
        gotoxy(3,7); cprintf("Press ESC to abort");
        if(getch()==13)  {                    /* if return is pressed */
                close_win(1,30,10,65,17);
                window(1,1,80,25);
                window(2,5,17,11);
                return 0;
                }
        close_win(1,30,10,65,17);
        window(1,1,80,25);
        window(2,5,17,11);
        return 999;                   /* escape flag */
}                                   /* end of function alt_edit */
/*************************************************************/
int alt_del(struct perf *selection)       /* alternatives check for edit */
{

        int i=0,j=0,k=0,l=0,m=0,check=pcount;
        struct perf *temp1,*temp2;
        struct alt *temp3;
        temp1=selection;
        temp2=begin;
        temp3=start;
        if(last==NULL)  {                     /* if no alt data, reset pcount */
                return --check;}
```

```
        open_win(1,30,10,68,17,BLUE,YELLOW);    /* else */
        textcolor(WHITE);                       /* message and mod alt data & pcount */
        gotoxy(3,2); cprintf("Deleting performance indicators");
        gotoxy(3,3); cprintf("will automatically change the data");
        gotoxy(3,4); cprintf("in the alternatives (.ALT) file.");
        gotoxy(3,6); cprintf("Press RETURN to continue");
        gotoxy(3,7); cprintf("Press ESC to abort");
        if(getch()==13)  {                      /* if return is pressed */
                close_win(1,30,10,68,17);
                window(1,1,80,25);
                window(2,5,17,11);
                while (temp2!=temp1)  {
                        i++;
                        temp2=temp2->next;
                }
                m=i;
                j=i;                            /* mod alt data */
                k=i+1;
                l=i+1;
                while(temp3 && i<12)  {
                        while(i!=check)  {
                                temp3->alt_perf_crit[i]=temp3->alt_perf_crit[k];
                                k++;
                                i++;
                        }
                        while(j!=check)  {
                                strcpy(temp3->calt_perf_crit[j],temp3->calt_perf_crit[l]);
                                l++;
                                j++;
                        }
                        temp3=temp3->next;
                        i=m;
                        j=m;
                        k=m+1;
                        l=m+1;
                }
                return --check;
                }
        check=999;                      /* escape flag */
        close_win(1,30,10,68,17);
        window(1,1,80,25);
        window(2,5,17,11);
        return check;
}                               /* end of function alt_del */
```

SECTION 5.4 - ALT.C

```
/* ALT.C ... This is the alternatives file */
#include "ded.h"                        /* ded header file */
#include <stdio.h>                      /* standard i/o header file */
#include <dos.h>                        /* dos header file */
#include <stdlib.h>                     /* standard library header file */
#include <conio.h>                      /* color i/o header file */
#include <dir.h>                        /* directory header file */
#include <string.h>                     /* string header file */
#include <math.h>                                               /* math header file */

#define MAXALT 15                       /* maximum number of alternatives */

/* declare function prototypes */

void draw_alt_screen(struct info *); /* draw screen defaults */
void display_dir(void);                 /* draw directory info */
void pop_alt_menu(void);                /* pop up menu */
void save_alt_data(struct alt *,struct info *);  /* save alt dat */
void disp_alt(struct alt *,int);        /* display alt data for scroll */
void error_check(void);                 /* error check */
void maxmin_check(struct alt *,struct perf *);   /* check perf tolerances */
void perf_maxmin(void);                                         /* displays perf ind max
and min for data entry */

char itoa(int,char *, int);             /* integer to ascii conversion */

int alt_menu(void);                     /* alt menu */
int get_input(int,int,int,char *,int,int);   /* get user input */
int alt_warn(void);                                             /* warns user if too many
alts entered */
```

```
struct alt *add_alt_data(struct alt *,struct alt *);    /* add data */
struct alt *scroll_alt(struct alt *);                   /* edit data */
struct alt *delete_alt(struct alt *);                   /* delete data */
struct alt *get_alt_data(struct alt *,int);             /* get user input */
struct alt *display_alt_data(struct alt *);             /* display alt data */

/* define external relationships */
extern int pcount;                                      /* perf count */
extern char *messages[];                                /* messages */
extern struct alt *start;                               /* struct alt pointer */
extern struct alt *last;                                /* struct alt pointer */
extern struct perf *begin;                              /* struct perf pointer */
extern struct perf *end;                                /* struct perf pointer */

/* define global variables */
static char *menu[4]={                                  /* pop up menu */
        " ADD DATA ",
        " EDIT DATA ",
        " DELETE DATA ",
        " SAVE CHANGES "
};

int mg,ng,eflag=0;                                      /* counters, flag */


/*********************************************************************/

struct info *alt(struct info *defaults)       /* module alt */
        {
                int k,aflag=0;
                char input;
                struct alt *temp;
                struct alt *selection;
                draw_alt_screen(defaults);                          /* draw initial
screen */
                display_dir();                                      /* display directory
info */
                if(!strcmp(defaults->name,"NONE"))    /* is file picked? */
                {
                        gotoxy(27,10);
                        colors(RED,WHITE);
                        cprintf("  NO FILE SPECIFIED    ");
                        gotoxy(27,11);
                        cprintf(" ESC and go to Setup ");
                }
                else  {                             /* if yes then display */
                        gotoxy(2,1);
                textcolor(BLACK);
                cprintf("[ALT:%-8s]",defaults->name);
                        display_alt_data(start);                    /* display any
existing data */
                }
                while((input=getch()) !=27)           /* while not escape */
                {
                if(input==0)                                                /* check for
extended code */
                {
                switch(getch())
                        {
                        case 59:
        /* F1 Help key */
                                gotoxy(10,10);
                                printf("HELP");
                        break;
                        case 60:
        /* F2 menu key */
                                open_win(3,2,5,17,11,MAGENTA,WHITE);
                                pop_alt_menu();
        /* pop up alt */
                                while ((k=alt_menu()) !=5 && k!=0)
                                        {
                /* while not exit, loop */
                                        switch(k)
                                                {
```

```
                                        case 1:                                 /* add
*/
                                                temp=(struct alt
*)malloc(sizeof(struct alt));
                                                aflag=alt_warn();
                                                if(!aflag)  {
                                                        close_win(3,2,5,17,11);
                                                        window(1,1,80,25);
                                                        perf_maxmin();
                                                        if(get_alt_data(temp,1))  {
                /* if not escape */
                                                                if(!eflag){
        maxmin_check(temp,begin);                               /* check for valid
data */
        start=add_alt_data(temp,start);                                 }

        display_alt_data(start);

        open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                                pop_alt_menu();
                                                        }
                                                }
                                                else  {
                                                        close_win(3,2,5,17,11);
                                                        window(1,1,80,25);

        open_win(3,14,10,44,16,RED,WHITE);
                                                        gotoxy(3,2);
cprintf("Maximum number of          ");
                                                        gotoxy(3,3);
cprintf("Alternatives have          ");
                                                        gotoxy(3,4);
cprintf("already been entered.      ");
                                                        gotoxy(3,5);  cprintf("Press
RETURN to continue. ");
                                                        if(getch()==13) {

        close_win(3,14,10,44,16);
                                                                window(1,1,80,25);

        open_win(3,2,5,17,11,MAGENTA,YELLOW);
                                                                pop_alt_menu();}
                                                }
                                        break;

                                        case 2:                 /* edit */
                                                temp=(struct alt
*)malloc(sizeof(struct alt));
                                                selection=scroll_alt(start);
                                                close_win(3,2,5,17,11);
                                                window(1,1,80,25);
                                                perf_maxmin();
                                                if(selection)
                                                temp=get_alt_data(selection,0);
                                                if(!eflag)
maxmin_check(temp,begin);                       /* check for valid data */
                                                display_alt_data(start);
                                                open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                pop_alt_menu();
                                                break;

                                        case 3:                 /* delete */
                                                close_win(3,2,5,17,11);
                                                selection=scroll_alt(start);    /*
select */
                                                if(selection)
                                                start=delete_alt(selection);    /*
reset pointers */
                                                display_alt_data(start);
                                                open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                pop_alt_menu();
                                                break;

                                        case 4:                 /* save */
```

```
                                                save_alt_data(start,defaults);
                                                close_win(3,2,5,17,11);
                                                display_alt_data(start);
                                                open_win(3,2,5,17,11,MAGENTA,WHITE);
                                                pop_alt_menu();
                                                break;

                                        default:
        /* error check */
                                                error_check();
                                        }
                                }

                        close_win(3,2,5,17,11);       /* close options window */
                        window(1,1,80,25);
                        message(messages[0]);
                        break;

                default:
                        error_check();
                }
        }
        }
        window(1,1,80,25);
        return(0);
}                                       /* end of function alt */

/*************************************************************************/
struct alt *scroll_alt(struct alt *ptr)    /* scroll thru data */
{
        int i,count=0,temp,rec=0,rec_count;
        char input;
        struct alt *alt_array[MAXALT];

        window(1,1,80,25);                      /* set up screen          */
        message(messages[2]);
        window(2,5,79,23);
        colors(LIGHTGRAY,BLACK);
        clrscr();

        while(ptr) {                            /* fill array of pointers */
                alt_array[count++]=ptr;
                ptr=ptr->next;                       /* and count them         */
        }
        rec_count=count;
        if(count>18) {                          /* if more than 1 screen  */
                rec=count-18;                        /* take last 18           */
                count=18;                            /* limit display          */
        }
        for(i=1;i<count;i++)
                disp_alt(alt_array[rec++],i);     /* display initial data   */

        colors(CYAN,BLACK);                     /* hilite the last record */
        gotoxy(1,count); cprintf("%77s"," ");
        disp_alt(alt_array[rec],count);


        /* If an extended character is picked it will start off with 00. */
        /* The following code throws out this 00 and gets the key that   */
        /* is picked.  If ESC (27) or Enter (13) is picked it takes      */
        /* a different course of action.                                 */

        while((input=getch())==0 && input!=27 && input!=13) {
                switch(getch()) {
                        case 72:                              /* cursor up              */
                                colors(LIGHTGRAY,BLACK);      /* unlite last entry      */
                                disp_alt(alt_array[rec],count);
                                if(count==1) {                /* top of screen          */
                                        rec--;
                                        if(rec<0) rec=0;
                                        else insline();
                                }
                                else
                                        disp_alt(alt_array[rec--],count--);
                                colors(CYAN,BLACK);
                                disp_alt(alt_array[rec],count);
                        break;
```

```
                case 80:
/* cursor down */
                        if(rec==rec_count-1) break;
                        colors(LIGHTGRAY,BLACK);        /* unlite last entry      */
                        disp_alt(alt_array[rec],count);
                        if(count==18) {         /* bottom of screen */
                                rec++;
                                if(rec>rec_count-1) rec=rec_count-1;
                                else {
                                        gotoxy(1,1);
                                        delline();
                                }
                        }
                        else
                                disp_alt(alt_array[rec++],count++);
                        colors(CYAN,BLACK);
                        disp_alt(alt_array[rec],count);
                break;

                case 79:
        /* end of list */
                        if((rec_count-1)>18) {
                                rec=rec_count-18;
                                count=18;
                        }
                        else {
                                rec=0;
                                count=rec_count;
                        }
                        colors(LIGHTGRAY,BLACK);
                        for(i=1;i<count;i++)
                                disp_alt(alt_array[rec++],i);
                        colors(CYAN,BLACK);
                        gotoxy(1,count); cprintf("%77s"," ");
                        disp_alt(alt_array[rec],count);
                break;

                case 71:
        /* top of list */
                        rec=0;
                        count=1;
                        colors(LIGHTGRAY,BLACK);
                        temp=rec_count>18?19:rec_count+1;
                        for(i=1;i<temp;i++)
                                disp_alt(alt_array[i-1],i);
                        colors(CYAN,BLACK);
                        gotoxy(1,count); cprintf("%77s"," ");
                        disp_alt(alt_array[rec],count);
                break;

                case 73:
        /* page up */
                        if(rec_count<19) break;
                        rec-=36;
                        if(rec<0) rec=0;
                        colors(LIGHTGRAY,BLACK);
                        for(i=1;i<19;i++)
                                disp_alt(alt_array[rec++],i);
                        colors(CYAN,BLACK);
                        gotoxy(1,count); cprintf("%77s"," ");
                        rec+=count-19;
                        disp_alt(alt_array[rec],count);
                break;

                case 81:
        /* page down */
                        if(rec_count<19) break;
                        rec+=36;
                        if(rec>rec_count-1) rec=rec_count-1;
                        rec=rec-17;
                        colors(LIGHTGRAY,BLACK);
                        for(i=1;i<19;i++)
                                disp_alt(alt_array[rec++],i);
                        colors(CYAN,BLACK);
                        gotoxy(1,count); cprintf("%77s"," ");
                        rec+=count-19;
                        disp_alt(alt_array[rec],count);
                break;
```

```
                case 59:
            /* help */
                        open_win(0,18,8,62,17,CYAN,BLUE);
                        gotoxy(18,1); cprintf("SCROLL HELP");
                        gotoxy(13,10); cprintf("Press any key to exit");
                        textcolor(WHITE);
                        gotoxy(3,3); cprintf("Cursor Up");
                        gotoxy(3,4); cprintf("Cursor Down");
                        gotoxy(3,5); cprintf("Page Up");
                        gotoxy(3,6); cprintf("Page Down");
                        gotoxy(3,7); cprintf("HOME");
                        gotoxy(3,8); cprintf("END");
                        textcolor(DARKGRAY);
                        gotoxy(13,3); cprintf("Moves up one record");
                        gotoxy(15,4); cprintf("Moves down one record");
                        gotoxy(11,5); cprintf("Moves up one screen");
                        gotoxy(13,6); cprintf("Moves down one screen");
                        gotoxy(8,7);  cprintf("Moves to startning of all
records");
                        gotoxy(7,8);  cprintf("Moves to end of all records");
                        getch();
                        close_win(0,18,8,62,17);
                        window(2,5,79,23);
                break;

            }
        }
        if(input==13) return(alt_array[rec]);   /* return selection */
        return(0);
}                                       /* end of function scroll */

/*************************************************************/

struct alt *delete_alt(struct alt * ptr)   /* delete alt data */
{
        struct alt *temp;
        open_win(1,25,10,54,15,BLUE,YELLOW);
        textcolor(WHITE);
        gotoxy(3,2); cprintf("Press RETURN to DELETE");
        gotoxy(3,5); cprintf("ESC to ABORT");
        textcolor(YELLOW);
        gotoxy(3,3); cprintf("%s",ptr->alt_name);
        if(getch()==13) {          /* if return is pressed */

            close_win(1,25,10,54,14);
            if(!ptr->next && !ptr->prior) {  /* only one record */
                    start=last=NULL;
                    return(start);
            }

            if(!ptr->prior) {               /* first record */
                    start=ptr->next;
                    ptr->next->prior=NULL;
                    return(start);
            }

            if(!ptr->next) {                /* last record */
                    last=ptr->prior;
                    ptr->prior->next=NULL;
                    return(start);
            }


            temp=ptr->prior;                /* middle record */
            temp->next=ptr->next;
            ptr->next->prior=temp;
            return(start);

        }
        close_win(1,25,10,54,15);
        window(1,1,80,25);
        return(start);                      /* return new pointer */
}                                   /* end of function delete */

/*********************************************************************/
void disp_alt(struct alt *ptr, int i)     /* display data for scroll */
```

```
{
        gotoxy(1,i);  cprintf("%77s"," ");
        gotoxy(5,i);  cprintf("%s",ptr->alt_name);
        if((fabs(ptr->alt_lcc)<.01) || (fabs(ptr->alt_lcc)>999999.99)) {
                gotoxy(20,i); cprintf("%10.2e",ptr->alt_lcc);}
        else {
                gotoxy(20,i); cprintf("%10.2f",ptr->alt_lcc);}
        return;
}                                       /* end of display */
/*************************************************************************/
void save_alt_data(struct alt *ptr,struct info *defaults)
{                                       /* save alt data */
        FILE *fp;
        char *alt_file;
        struct alt *info;
        info=ptr;
        alt_file=(char *)malloc(14);
        strcpy(alt_file,defaults->name);
        strcat(alt_file,".ALT");
        window(19,8,46,10);
        textbackground(BLACK);
        clrscr();
        window(18,7,45,9);
        textbackground(MAGENTA);
        clrscr();
        border(WHITE,18,7,45,9);
        textcolor(YELLOW);
        gotoxy(3,2); cprintf("SAVING FILE:");
        colors(CYAN,BLACK);
        gotoxy(15,2); cprintf(" %s",alt_file);
        if((fp=fopen(alt_file,"wb"))==NULL) {
                cprintf("Cannot open file\n");
                exit(1);
        }
        while(info) {
                fwrite(info,sizeof(struct alt),1,fp);
                info=info->next;
        }
        fclose(fp);
        return;
}                                       /* end of save alt data */


/*************************************************************************/
void pop_alt_menu(void)                 /* pops up alt menu */
{
        int i;
        window(1,1,80,25);
        message(messages[4]);
        window(2,5,17,11);
        colors(MAGENTA,YELLOW);

        for(i=0;i<4;i++){
                gotoxy(2,i+2);
                cprintf("%s",menu[i]);              /*print menu*/
        }
        colors(CYAN,WHITE);                     /* hilite first item */
        gotoxy(2,2);
        cprintf("%s",menu[0]);
        mg=2;ng=0;
}                                        /* end of pop up menu */

/*************************************************************************/

int alt_menu(void)                      /* gets alt menu input */
{
        int move;
        while((move=getch()) !=13)          /* check for return */
        {
                if( move==0)                     /* check for extended code */
                {
                        switch(getch())
                        {
                                case 80:                                /* down cursor */
                                        colors(MAGENTA,YELLOW);
                                        gotoxy(2,mg++); cprintf("%s",menu[ng++]);
```

```
                                        if(ng>3) {mg=2;ng=0;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,mg); cprintf("%s",menu[ng]);
                                        break;

                                case 72:                                /* up cursor */
                                        colors(MAGENTA,YELLOW);
                                        gotoxy(2,mg--); cprintf("%s",menu[ng--]);
                                        if(ng<0) {mg=5;ng=3;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,mg); cprintf("%s",menu[ng]);
                                        break;
                        }
                }
                if (move==27) return(0);                        /* escape */

        }
        return(ng+1);                                   /* return selection */
}

                                /* end of menu select */
/*************************************************************************/
struct alt *display_alt_data(struct alt *ptr)     /* display alt data */

{
        int i=1;
        colors(LIGHTGRAY,BLACK);
        window(2,5,79,23);
        clrscr();

                                        /* print data */
        while(ptr && i<19) {

                gotoxy(5,i);  cprintf("%s",ptr->alt_name);
        if((fabs(ptr->alt_lcc)<.01) || (fabs(ptr->alt_lcc)>999999.99)) {
                gotoxy(20,i++); cprintf("%10.2e",ptr->alt_lcc);}
        else {
                gotoxy(20,i++); cprintf("%10.2f",ptr->alt_lcc);}

                ptr=ptr->next;
        }
        window(1,1,80,25);
        return 0;
}                                                       /* end of display data */

/*************************************************************************/
struct alt *add_alt_data(struct alt *ptr, struct alt *top)
{                                       /* add alt data */

        if(last==NULL) {          /* list is empty, this is the first */
                last=ptr;
                ptr->prior=NULL;
                ptr->next=NULL;
                return(ptr);
        }                           /* else add data onto end of list */

        last->next=ptr;
        ptr->prior=last;          /* prior is previous end of file  */
        last=ptr;                 /* new end is ptr */
        ptr->next=NULL;
        return(top);

}                           /* end of add data */

/*************************************************************************/
struct alt *get_alt_data(struct alt *ptr,int flag)
{                           /* get user input */

        struct perf *pperf;
        unsigned int k=1,iy=4;
        int pcheck=0,j=1;
        pperf=begin;
        window(1,1,80,25);
        if (flag != 2)
                message(messages[3]);
/*      else                        */
/*              message(messages[1]); */
        open_win(1,45,3,71,20,MAGENTA,WHITE);
```

```
        gotoxy(3,2);  cprintf("NAME:");          /* print titles */
        gotoxy(3,3);  cprintf("LCC ($):");
        while(pperf)  {                          /* print data */
                gotoxy(3,iy++);  cprintf("%s:",pperf->perf_name);
                pperf=pperf->next;
        }


        if(flag==1) {                       /* not in edit mode            */
                strcpy(ptr->alt_name," ");      /* initialize variables        */
                ptr->alt_lcc=0.0;
                strcpy(ptr->calt_lcc," ");

                while(pcheck<=pcount)  {
                        strcpy(ptr->calt_perf_crit[pcheck]," ");
                        ptr->alt_perf_crit[pcheck]=0.0;
                        pcheck++;

                }
        }

        else {                                  /* print data to edit          */
                iy=4;
                pcheck=0;
                textcolor(WHITE);
                gotoxy(12,2); cprintf("%-15s",ptr->alt_name);
                gotoxy(12,3); cprintf("%-15s",ptr->calt_lcc);
                while(pcheck<=pcount)  {
                        gotoxy(12,iy++); cprintf("%-15s",ptr->calt_perf_crit[pcheck++]);
                }
        }

        textcolor(WHITE);

        while(j && j!=27) {                     /* keep going unless j=0 or ESC */
                switch(k) {
                        case 1:                                 /* alt name */
                                j=get_input(11,2,14,ptr->alt_name,CYAN,WHITE);
                                k=(j<0) ? 1:2;
                                break;


                        case 2:                                 /* lcc */
                                j=get_input(11,3,14,ptr->calt_lcc,CYAN,WHITE);
                                ptr->alt_lcc=atof(ptr->calt_lcc);
                                k+=j;
                                break;


                        case 3:
                                iy=4;
                                pcheck=0;
                                while(pcheck<=pcount)  {        /* performance ind values */
                                        j=get_input(11,iy,14,ptr-
>calt_perf_crit[pcheck],CYAN,WHITE);
                                        ptr->alt_perf_crit[pcheck]=atof(ptr-
>calt_perf_crit[pcheck]);
                                        pcheck+=j;
                                        if (pcheck<0){
                                                k=2;
                                                break;
                                        }
                                        else (iy+=j);
                                }
                                if(k!=2) j=0;
                                break;
                }

        }
        close_win(1,45,3,71,20);
        window(1,1,80,25);
        message(messages[0]);
        window(2,5,17,11);
        if(j==27) {
                eflag=1;            /* if escape */
                return(ptr);
        }
        else {
```

```
                eflag=0;
                return(ptr);}
}

/*************************************************************************/

void draw_alt_screen(struct info *defaults)    /* defaults screen */
{

        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(2,1); cprintf("[ALT: NONE]");
        gotoxy(36,1);
        cprintf("ALTERNATIVES DATA");
        textcolor(WHITE);

        gotoxy (5,2); cprintf("LCC TYPE: %s", defaults->alt_lcc_type);

        gotoxy(4,3); cprintf("ALTERNATIVE");
        gotoxy(7,4); cprintf("NAMES");
        gotoxy(25,3); cprintf("LCC");
        gotoxy(25,4); cprintf("($)");
        colors(RED,WHITE);
        window(1,1,80,25);
        message(messages[0]);

}                                       /* end of draw defaults */


/*************************************************************************/
void maxmin_check(struct alt *ckptr, struct perf *pptr)
{                                       /* perf ind val cross check */
        float maxnum=0.0,minnum=0.0;
        int i=0;
        while (pptr)  {                     /* find min and max for given perf ind */
                maxnum=pptr->perf_10;
                if ((pptr->perf_5)>maxnum) {
                        maxnum=pptr->perf_5;
                }
                if ((pptr->perf_0)>maxnum) {
                        maxnum=pptr->perf_0;
                }
                minnum=pptr->perf_0;
                if ((pptr->perf_5)<minnum) {
                        minnum=pptr->perf_5;
                }
                if ((pptr->perf_10)<minnum)  {
                        minnum=pptr->perf_10;
                }
                if (minnum<=ckptr->alt_perf_crit[i] && ckptr->alt_perf_crit[i]<=maxnum)
{
                        i++;                            /* check alternatives value */
                        pptr=pptr->next;
                }
                else{
                        window(2,15,38,21);
                        clrscr();
                        textbackground(BLUE);
                        clrscr();
                        border(WHITE,2,15,38,21);
                        textcolor(YELLOW);          /* not within limits */
                        gotoxy(2,2);  cprintf("The alternative value entered");
                        gotoxy(2,3);  cprintf("exceeds the performance indicator");
                        gotoxy(2,4);  cprintf("data boundaries.  Please enter a");
                        gotoxy(2,5);  cprintf("new value for %s between",pptr->perf_name);
                        if((fabs(minnum)<.01) || (fabs(maxnum)>9999.99)) {
                                gotoxy(2,6);  cprintf("%7.2e and %7.2e
(inclusive).",minnum,maxnum);}
                        else {
                                gotoxy(2,6);  cprintf("%7.2f and %7.2f
(inclusive).",minnum,maxnum);}
                        get_alt_data(ckptr,2);      /* get data to edit */
                        if (eflag)  return;
                }
        }
}                                       /* end of maxmin check */
```

```
/*************************************************************************/
void perf_maxmin()
{                                       /* perf ind max/min display */
        struct perf *pptr;
        float maxnum=0.0,minnum=0.0,max_array[12],min_array[12];
        int i=0,iy=0;
        pptr=begin;
        while (pptr)  {                         /* find min and max for given perf ind */
                maxnum=pptr->perf_10;
                if ((pptr->perf_5)>maxnum) {
                        maxnum=pptr->perf_5;
                }
                if ((pptr->perf_0)>maxnum) {
                        maxnum=pptr->perf_0;
                }
                max_array[i]=maxnum;
                minnum=pptr->perf_0;
                if ((pptr->perf_5)<minnum) {
                        minnum=pptr->perf_5;
                }
                if ((pptr->perf_10)<minnum)  {
                        minnum=pptr->perf_10;
                }
                min_array[i++]=minnum;
                pptr=pptr->next;}

        window(2,8,42,22);
        clrscr();
        textbackground(BLUE);
        clrscr();
        border(WHITE,2,8,42,22);
        textcolor(WHITE);          /* display limits */
        gotoxy(6,1);  cprintf("PERFORMANCE INDICATOR LIMITS");
        gotoxy(2,2);  cprintf("name         minumum       maximum");
        textcolor(YELLOW);
        pptr=begin;
        while(pptr && iy<=pcount)  {
                if((fabs(min_array[iy])<.01) || (fabs(max_array[iy])>999999.99)) {
                        gotoxy(2,iy+3);  cprintf("%-10s   %10.2e  %10.2e",pptr-
>perf_name,min_array[iy],max_array[iy]);}
                else {
                        gotoxy(2,iy+3);  cprintf("%-10s   %10.2f  %10.2f",pptr-
>perf_name,min_array[iy],max_array[iy]);}
        iy++;
        pptr=pptr->next;
        }
}
                /* end of perf_maxmin */


/*************************************************************************/
int alt_warn(void)
{
        struct alt *aptr;
        int flag=0,count=0;
        aptr=start;
        while(aptr)  {
                count++;
                aptr=aptr->next;}
        if(count==15)  flag=1;
        else flag=0;
        return(flag);
}
```

SECTION 5.5 - CALC.C

```
/* CALC.C ... This is the calculations module */
/* for the Utility Decision Evaluation Display program */
/* Pam Rice   14 November 1991 */

#include "ded.h"                                /* ded header file */
#include <stdio.h>          /* standard i/o header file */
#include <dos.h>            /* dos header file */
#include <stdlib.h>         /* standard library header file */
#include <conio.h>          /* color i/o header file */
#include <dir.h>            /* directory header file */
#include <string.h>         /* string header file */
```

```
#include <math.h>          /* math header file */


/* declare function prototypes */

void save_calc_data(struct calc *,struct info *);  /* save calc */
void save_perf_data(struct perf *,struct info *);  /* save perf */
void save_alt_data(struct alt *,struct info *);    /* save alt */
void pop_calc_menu(void);       /* pop up calc menu */
void limit_ck(void);            /* perf ind limit check */
void reg_calc(void);            /* regular calculations */
void mod_calc(void);            /* requirements calc */
void draw_calc_screen(void);    /* default screen */
int  calc_menu(void);           /* calc menu */
struct calc *add_calc_data(struct calc *,struct calc *);   /* add data */
struct alt *get_alt_data(struct alt*,int);          /* get user input */

/* define external relationships */
extern char *messages[];        /* messages */
extern int pcount;              /* perf ind count */
extern struct perf *begin;      /* struct perf pointer */
extern struct perf *end;        /* struct perf pointer */
extern struct alt *start;       /* struct alt pointer */
extern struct alt *last;        /* struct alt pointer */
extern struct calc *c_begin;    /* struct calc pointer */
extern struct calc *c_end;      /* struct calc pointer */


/* define global variables */
static char *menu[2]={                /* calc menu */
       "Regular calculation",
       "Requirement line calculation"
};

int mc,nc;                            /* menu counters */


/*********************************************************************/

void calc(struct info *defaults)            /* calc module */
{
       int k;
              open_win(1,25,10,54,13,BLUE,YELLOW);
              pop_calc_menu();                                          /* pop up
calc */
              while ((k=calc_menu()) !=3 && k!=0)
              {                                                         /* while not
exit, loop */
                     switch(k)
                     {
                     case 1:                                  /* Regular calculation */
                            close_win(1,25,10,54,13);
                            limit_ck();       /* check perf ind limits */
                            reg_calc();       /* calculate */
                            save_calc_data(c_begin,defaults);  /* save data */
                            save_perf_data(begin,defaults);
                            save_alt_data(start,defaults);
                            break;

                     case 2:                                  /* Calc based on requirement
> perf_5 */
                            close_win(1,25,10,54,13);
                            limit_ck();       /* check perf ind limits */
                            mod_calc();       /* calculate */
                            save_calc_data(c_begin,defaults);   /* save data */
                            save_perf_data(begin,defaults);
                            save_alt_data (start,defaults);
                            break;
                     }
                 break;
              }
       if(!k) close_win(1,25,10,54,13);
       window(1,1,80,25);
/*     message(messages[1]);   */
       return;
}                                          /* end of function calc */
/*********************************************************************/
void pop_calc_menu(void)                /* pops up calc menu */
```

```
{
        int i;
        window(1,1,80,25);
        message(messages[4]);
        window(25,10,54,13);
        colors(BLUE,YELLOW);

        for(i=0;i<2;i++){
                gotoxy(2,i+2);
                cprintf("%s",menu[i]);              /* print menu*/
        }
        colors(CYAN,WHITE);                      /* hilite first item */
        gotoxy(2,2);
        cprintf("%s",menu[0]);
        mc=2;nc=0;
}

/*********************************************************************/

int calc_menu(void)                        /* gets calculation input */
{
        int move;
        while((move=getch()) !=13)          /* check for return */
        {
                if( move==0)                      /* check for extended code */
                {
                        switch(getch())
                        {
                                case 80:                                  /* down cursor */
                                        colors(BLUE,YELLOW);
                                        gotoxy(2,mc++); cprintf("%s",menu[nc++]);
                                        if(nc>1) {mc=2;nc=0;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,mc); cprintf("%s",menu[nc]);
                                        break;

                                case 72:                                   /* up cursor */
                                        colors(BLUE,YELLOW);
                                        gotoxy(2,mc--); cprintf("%s",menu[nc--]);
                                        if(nc<0) {mc=3;nc=1;}
                                        colors(CYAN,WHITE);
                                        gotoxy(2,mc); cprintf("%s",menu[nc]);
                                        break;
                        }
                }
                if (move==27) return(0);                  /* escape */

        }
        return(nc+1);                                    /* return selection */
}                                                      /* end of menu select */

/*********************************************************************/
void limit_ck(void)             /* perf ind limit check */
{
        struct perf *pptr;
        struct alt *aptr;
        float maxnum=0.0,minnum=0.0;
        int i=0;
        pptr=begin;
        aptr=start;
        while (pptr)  {              /* calc min & max for each indicator */
                maxnum=pptr->perf_10;
                if ((pptr->perf_5)>maxnum) {
                        maxnum=pptr->perf_5;
                }
                if ((pptr->perf_0)>maxnum) {
                        maxnum=pptr->perf_0;
                }
                minnum=pptr->perf_0;
                if ((pptr->perf_5)<minnum) {
                        minnum=pptr->perf_5;
                }
                if ((pptr->perf_10)<minnum)  {
                        minnum=pptr->perf_10;
                }
                while(aptr) {              /* check that each alt value is w/in range */
                        if (minnum<=aptr->alt_perf_crit[i] && aptr-
>alt_perf_crit[i]<=maxnum)  {
```

```
                        aptr=aptr->next;
                        }
                        else{                   /* if not, edit values */
                                draw_calc_screen();
                                window(2,15,38,21);
                                clrscr();
                                textbackground(BLUE);
                                clrscr();
                                border(WHITE,2,15,38,21);
                                textcolor(YELLOW);
                                gotoxy(2,2);  cprintf("The %s value entered",aptr-
>alt_name);
                                gotoxy(2,3);  cprintf("exceeds the performance
indicator");
                                gotoxy(2,4);  cprintf("data boundaries.  Please enter a");
                                gotoxy(2,5);  cprintf("new value for %s between",pptr-
>perf_name);
                                gotoxy(2,6);  cprintf("%f and %f
(inclusive).",minnum,maxnum);
                                get_alt_data(aptr,2);       /* get user input */
                        }
                }
        i++;
        pptr=pptr->next;            /* go to next indicator */
        }
window(1,1,80,25);
return;
}                               /* end of function limit_check */
/*****************************************************************/
void draw_calc_screen(void)     /* draw default screen */
{
        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(30,1);  cprintf("CALCULATIONS SCREEN");
        colors(RED,WHITE);
        window(1,1,80,25);
        message(messages[4]);
}


/*****************************************************************/
void reg_calc(void)             /* performs regular calc */
{
        struct perf *pptr;
        struct alt *aptr;
        struct calc *cptr;
        struct calc *temp;
        unsigned int perf_rate_tot=0,i=1,j=1;
        float hold=0.0,xp=0.0,xm=0.0;
        pptr=begin;
        aptr=start;
        c_begin=c_end=NULL;
        cptr=c_begin;
        while (pptr)  {                     /* calculate perf_rate_tot and perf_num */
                perf_rate_tot+=pptr->perf_rate;
                pptr->perf_num=i++;
                pptr=pptr->next;
        }
        pptr=begin;
        i=0;
        while (pptr)  {                     /* calculate perf_wt & equation values
(a0,a1,a2) */
                pptr->perf_wt=((pptr->perf_rate)/(float)perf_rate_tot)*100.;
                pptr->a0=0.02*(pptr->perf_10)-0.04*(pptr->perf_5)+0.02*(pptr->perf_0);
                pptr->a1=0.4*(pptr->perf_5)-0.3*(pptr->perf_0)-0.1*(pptr->perf_10);
                pptr->a2=pptr->perf_0;
                pptr=pptr->next;
        }
        while (aptr) {          /* calculate total number of alternatives */
                i++;
                aptr=aptr->next;
        }
        i++;
        while (j!=i)  { /* enter alternative numbers */
                temp=(struct calc *)malloc(sizeof(struct calc));
                temp->alt_num=j++;
```

```
                c_begin=add_calc_data(temp,c_begin);
                cptr=c_begin;
                cptr=cptr->next;
        }
        pptr=begin;
        i=0;
        while (pptr)  {                                                         /*
calculate utility values for alternatives */
                aptr=start;
                cptr=c_begin;
                while (aptr && cptr)  {
                        if (fabs(pptr->a0)>0.001)  {            /* use quadratic eqn if
x**2 present */
                                hold=((pptr->a1)*(pptr->a1))-4.0*(pptr->a0)*((pptr->a2)-
(aptr->alt_perf_crit[i]));
                                xp=(-(pptr->a1)+sqrt((double) hold))/(2.0*(pptr->a0));
                                xm=(-(pptr->a1)-sqrt((double) hold))/(2.0*(pptr->a0));
                                if (xp>=0.0 && xp<=10.0)  cptr->alt_util_val[i]=xp;
                                else  cptr->alt_util_val[i]=xm;
                        }
                        else  {     /* if a straight line use this */
                                cptr->alt_util_val[i]=((aptr->alt_perf_crit[i])-(pptr-
>a2))/(pptr->a1);
                        }
                aptr=aptr->next;
                cptr=cptr->next;
                }
         i++;
         pptr=pptr->next;
         }
         pptr=begin;        /* calculate alt_wt_score */
         i=0;
         while (pptr)  {
                cptr=c_begin;
                while (cptr)  {
                        cptr->alt_wt_score[i]=(cptr->alt_util_val[i])*(pptr->perf_wt);
                        cptr=cptr->next;
                 }
         i++;
         pptr=pptr->next;
         }
         cptr=c_begin;                  /* calculate alt_wt_ind */
         while (cptr)  {
                i=0;
                cptr->alt_wt_ind=0.0;
                while(i<=pcount)  {
                        cptr->alt_wt_ind+=cptr->alt_wt_score[i++];
                }
                cptr=cptr->next;
         }
         aptr=start;
         cptr=c_begin;
         while(cptr)  {
                strcpy(cptr->altname,aptr->alt_name);
                cptr->altlcc=aptr->alt_lcc;
                aptr=aptr->next;
                cptr=cptr->next;
         }
  window(1,1,80,25);
  return;
}                                               /* end of reg_calc */
/*****************************************************************/
void mod_calc(void)             /* calc based on perf>5 values */
{                               /* throws out alts that don't meet reqts */
        struct perf *pptr;
        struct alt *aptr;
        struct calc *cptr;
        struct calc *temp;
        unsigned int perf_rate_tot=0,i=1,j=1,jj=0,flag[15];
        float hold=0.0,xp=0.0,xm=0.0;
        pptr=begin;
        aptr=start;
        c_begin=c_end=NULL;
        cptr=c_begin;
        while (jj<15) {
                flag[jj++]=0;}
        while (pptr)  {                         /* calculate perf_rate_tot and perf_num */
                perf_rate_tot+=pptr->perf_rate;
```

```
                pptr->perf_num=i++;
                pptr=pptr->next;
        }
        pptr=begin;
        i=0;
        while (pptr)  {                     /* calculate perf_wt & equation values
(a0,a1,a2) */
                pptr->perf_wt=((pptr->perf_rate)/(float)perf_rate_tot)*100.;
                pptr->a0=0.02*(pptr->perf_10)-0.04*(pptr->perf_5)+0.02*(pptr->perf_0);
                pptr->a1=0.4*(pptr->perf_5)-0.3*(pptr->perf_0)-0.1*(pptr->perf_10);
                pptr->a2=pptr->perf_0;
                pptr=pptr->next;
        }
        while (aptr) {          /* calculate total number of alternatives */
                i++;
                aptr=aptr->next;
        }
        i++;
        while (j!=i)  { /* enter alternative numbers */
                temp=(struct calc *)malloc(sizeof(struct calc));
                temp->alt_num=j++;
                c_begin=add_calc_data(temp,c_begin);
                cptr=c_begin;
                cptr=cptr->next;
        }
        pptr=begin;
        i=0;
        while (pptr)  {                                                          /*
calculate utility values for alternatives */
                aptr=start;
                cptr=c_begin;
                while (aptr && cptr)  {
                        if ((pptr->a0)!=0)  {           /* use quadratic eqn if x**2
present */
                                hold=((pptr->a1)*(pptr->a1))-4.0*(pptr->a0)*((pptr->a2)-
(aptr->alt_perf_crit[i]));
                                xp=(-(pptr->a1)+sqrt((double) hold))/(2.0*(pptr->a0));
                                xm=(-(pptr->a1)-sqrt((double) hold))/(2.0*(pptr->a0));
                                if (xp>=0.0 && xp<=10.0)  cptr->alt_util_val[i]=xp;
                                else  cptr->alt_util_val[i]=xm;
                        }
                        else  {    /* if a straight line use this */
                                cptr->alt_util_val[i]=((aptr->alt_perf_crit[i])-(pptr-
>a2))/(pptr->a1);
                        }
                aptr=aptr->next;
                cptr=cptr->next;
                }
         i++;
         pptr=pptr->next;
         }
         pptr=begin;       /* calculate alt_wt_score */
         i=0;
         while (pptr)  {
                cptr=c_begin;
                j=0;
                while (cptr)  {
                        if((cptr->alt_util_val[i])<5)   /* if won't meet reqt */
                        {
                                cptr->alt_wt_score[i]=0.0;   /* set equal to zero */
                                flag[j]=1;  }
                        else {
                                cptr->alt_wt_score[i]=(cptr->alt_util_val[i])*(pptr-
>perf_wt);
                                if (flag[j]!=1)  flag[j]=0;
                        }
                        cptr=cptr->next;
                        j++;
                 }
         i++;
         pptr=pptr->next;
         }
         cptr=c_begin;                /* calculate alt_wt_ind */
         j=0;
         while (cptr)  {
                i=0;
                cptr->alt_wt_ind=0.0;
                while(i<=pcount)  {
```

```
                        if(flag[j])  {                          /* if doesn't meet reqt */
                                cptr->alt_wt_ind=0.0;   /* set equal to zero */
                                i=pcount+1;}
                        else  {
                                cptr->alt_wt_ind+=cptr->alt_wt_score[i++];
                        }
                }
                j++;
                cptr=cptr->next;
        }
        aptr=start;
        cptr=c_begin;
        while(cptr)  {
                strcpy(cptr->altname,aptr->alt_name);
                cptr->altlcc=aptr->alt_lcc;
                aptr=aptr->next;
                cptr=cptr->next;
        }

  window(1,1,80,25);
  return;

}                                       /* end of function mod_calc */

/*************************************************************************/
struct calc *add_calc_data(struct calc *ptr, struct calc *top)
{                               /* add data to calc */

        if(c_end==NULL) {          /* list is empty, this is the first */
                c_end=ptr;
                ptr->prior=NULL;
                ptr->next=NULL;
                return(ptr);
        }                           /* else add data onto end of list */

        c_end->next=ptr;
        ptr->prior=c_end;          /* prior is previous end of file  */
        c_end=ptr;                 /* new end is ptr */
        ptr->next=NULL;
        return(top);

}                               /* end of add */

/*************************************************************************/
void save_calc_data(struct calc *ptr,struct info *defaults)
{                               /* save calc data */
        FILE *fp;
        char *calc_file;
        struct calc *info;
        info=ptr;
        calc_file=(char *)malloc(14);
        strcpy(calc_file,defaults->name);
        strcat(calc_file,".CAL");
        window(19,8,46,10);
        textbackground(BLACK);
        clrscr();
        window(18,7,45,9);
        textbackground(MAGENTA);
        clrscr();
        border(WHITE,18,7,45,9);
        textcolor(YELLOW);
        gotoxy(3,2); cprintf("SAVING FILE:");
        colors(CYAN,BLACK);
        gotoxy(15,2); cprintf(" %s",calc_file);
        if((fp=fopen(calc_file,"wb"))==NULL) {
                cprintf("Cannot open file\n");
                exit(1);
        }
        while(info) {
                fwrite(info,sizeof(struct calc),1,fp);
                info=info->next;
        }
        fclose(fp);
        return;
}

         /* end of save */
/*************************************************************************/
```

SECTION 5.6 - REPORT.C

```
/* REPORT.C ... This is the reports module */
/* for the Decision Evaluation Display program */
/* Pam Rice   2 November 1991 */

#include "ded.h"
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <dir.h>
#include <string.h>
#include <math.h>

#define MAXALT 15                                                    /* maximum number
of alternatives */

/* declare function prototypes */
void draw_report_screen(void);
int select_report(void);
void perf_report1(void);
void perf_report2(void);
void alt_report1(struct info *);
void alt_report2(struct info *);
void alt_util_report1(struct info *);
void alt_util_report2(struct info *);
void weight_score_report1(struct info *);
void weight_score_report2(struct info *);
void plot_ded(void);
int screen_check(void);
int screen_check1(void);
int page_check(void);
int page_check1(void);

/* define external relationships */
extern int pcount;
extern char *messages[];
extern struct perf *begin;
extern struct perf *end;
extern struct alt *start;
extern struct alt *last;
extern struct calc *c_begin;
extern struct calc *c_end;


int pflag;

/*************************************************************************/
void report(struct info *defaults)
{

        pflag=0;
        for(;;)  {
                draw_report_screen();
                        switch(select_report()){

                                case 1:
                                        if(pflag)  perf_report1();
                                        else perf_report2();
                                break;

                                case 2:                                  /* Alternatives data
*/
                                        if(pflag)  alt_report1(defaults);
                                        else alt_report2(defaults);
                                break;

                                case 3:                                  /* Alternative utility
data   */
                                        if(pflag)  alt_util_report1(defaults);
                                        else alt_util_report2(defaults);
                                break;
```

```
                                case 4:                                 /* Weighted score
report                   */
                                        if(pflag)  weight_score_report1(defaults);
                                        else weight_score_report2(defaults);
                                break;

                                case 5:
                        /* Return to main menu */
                                        cursoff();
                                        window(1,1,80,25);
                                        return;
                        }

        }

}                                       /* end of function report */


/*************************************************************/
void perf_report2()                     /* report for performance indicator data & */
{                                      /* utility curve data  */


        struct perf *ptr;
        int i=1;
        pflag=0;
                window(1,1,80,25);
                textbackground(LIGHTGRAY);
                clrscr();
                border(BLUE,1,1,80,24);
                gotoxy(36,1);
                cprintf("PERF DATA");
                textcolor(WHITE);

                gotoxy(4,2); cprintf("PERFORMANCE");
                gotoxy(5,3); cprintf("INDICATOR");
                gotoxy(20,3); cprintf("RANK");
                gotoxy(28,3); cprintf("RATING");
                gotoxy(43,2); cprintf("UTILITY CURVE BENCHMARKS");
                gotoxy(40,3); cprintf("0");
                gotoxy(51,3); cprintf("5");
                gotoxy(63,3); cprintf("10");
                gotoxy(70,3); cprintf("WEIGHT");
                colors(RED,WHITE);
                window(1,1,80,25);
                message(messages[4]);
                colors(LIGHTGRAY,BLACK);
                window(2,4,79,23);
                clrscr();
                ptr=begin;
                while(ptr && i<19) {

                        gotoxy(5,i);  cprintf("%s",ptr->perf_name);
                        gotoxy(18,i); cprintf("%3u",ptr->perf_rank);
                        gotoxy(27,i); cprintf("%4u",ptr->perf_rate);
                        if((fabs(ptr->perf_0)<.01) || (fabs(ptr->perf_0)>9999.99)) {
                                gotoxy(36,i); cprintf("%7.2e",ptr->perf_0);}
                        else {
                                gotoxy(36,i); cprintf("%7.2f",ptr->perf_0);}
                        if((fabs(ptr->perf_5)<.01) || (fabs(ptr->perf_5)>9999.99)) {
                                gotoxy(47,i); cprintf("%7.2e",ptr->perf_5);}
                        else {
                                gotoxy(47,i); cprintf("%7.2f",ptr->perf_5);}
                        if((fabs(ptr->perf_10)<.01) || (fabs(ptr->perf_10)>9999.99)){
                                gotoxy(59,i); cprintf("%7.2e",ptr->perf_10);}
                        else {
                                gotoxy(59,i); cprintf("%7.2f",ptr->perf_10);}
                        gotoxy(68,i++);  cprintf("%7.2f",ptr->perf_wt);
                        ptr=ptr->next;
                        }
                getch();
        window(1,1,80,25);
}

/*************************************************************/
void perf_report1()                     /* report for performance indicator data & */
{                                      /* utility curve data hardcopy printout */
```

```
        FILE *fptr;
        struct perf *ptr;
        ptr=begin;
        pflag=1;
        fptr=fopen("prn","wb");
        open_win(1,40,10,61,14,RED,WHITE);
        gotoxy(3,2);  cprintf("Make sure printer");     /* printer warning message */
        gotoxy(3,3);  cprintf(" is online   ");
        gotoxy(3,4);  cprintf("Press any key");
        getch();
        close_win(1,40,10,61,14);

        fprintf(fptr,"                      PERFORMANCE INDICATORS\n\r\n\r");
        fprintf(fptr,"PERFORMANCE                          UTILITY CURVE
BENCHMARKS\n\r");
        fprintf(fptr," INDICATOR   RANK   RATING      0            5          10
WEIGHT\n\r");
        fprintf(fptr,"_____________________________________________________________________
_____\n\r");
        while (ptr)  {
                fprintf(fptr,"%-10s",ptr->perf_name);
                fprintf(fptr,"%6u",ptr->perf_rank);
                fprintf(fptr,"%8u",ptr->perf_rate);
                fprintf(fptr,"%12.2f",ptr->perf_0);
                fprintf(fptr,"%12.2f",ptr->perf_5);
                fprintf(fptr,"%12.2f",ptr->perf_10);
                fprintf(fptr,"%12.2f\n\r",ptr->perf_wt);
                ptr=ptr->next;
        }
        fprintf(fptr,"\n\r\f");
        fclose(fptr);
}
                                                /* end of perf ind print out */


/*****************************************************************/
void alt_report2(struct info *defaults)                    /* report for alternatives
data */
{
        struct alt *ptr;
        struct perf *pptr,*hold_pptr[4];
        int i=1,ix=30,pcheck=0,scrn_cnt=1,pcnt=4,j=0,k=1,cnt_scrn=0,ptemp=0,page_flag=0;
        char buffer[3][2646],input;
        scrn_cnt=screen_check();
        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(36,1);
        cprintf("ALTERNATIVES DATA");
        textcolor(WHITE);
        gotoxy(4,2); cprintf("LCC TYPE: %s",defaults->alt_lcc_type);
        gotoxy(4,3); cprintf("ALTERNATIVE");
        gotoxy(7,4); cprintf("NAMES");
        gotoxy(18,3); cprintf("LCC");
        gotoxy(18,4); cprintf("($)");
        colors(LIGHTGRAY,BLACK);
        window(2,5,79,23);
        clrscr();
        ptr=start;
        while(ptr && i<19) {
                gotoxy(3,i);  cprintf("%s",ptr->alt_name);
                if((fabs(ptr->alt_lcc)<.01) || (fabs(ptr->alt_lcc)>9999.99)) {
                        gotoxy(16,i); cprintf("%7.2e",ptr->alt_lcc);}
                else {
                        gotoxy(16,i); cprintf("%7.2f",ptr->alt_lcc);}
                i++;
                ptr=ptr->next;
        }
        window(1,1,80,25);
        pptr=begin;
        hold_pptr[0]=begin;
        while(k<scrn_cnt)  {
                pcnt=4;
                while(j<pcnt)  {
                        pptr=pptr->next;
                        pcnt--; }
```

```
                hold_pptr[k]=pptr;
                k++;
        }
        pptr=begin;
        pcnt=4;
        textcolor(WHITE);
        gotoxy(30,3); cprintf("PERFORMANCE INDICATORS");
        ix=30;
        while(pptr && pcnt)  {
                gotoxy(ix,4); cprintf("%s",pptr->perf_name);
                ix=ix+12;
                pptr=pptr->next;
                pcnt--;}
        window(2,5,79,23);
        i=1;
        textcolor(BLACK);
        ptr=start;
        while(ptr && i<19)  {
                pcnt=4;
                ix=30;
                pcheck=0;
                while(pcheck<=pcount && pcnt)  {
                        if((fabs(ptr->alt_perf_crit[pcheck])<.01) || (fabs(ptr-
>alt_perf_crit[pcheck])>9999.99)) {
                                gotoxy(ix,i); cprintf("%7.2e",ptr-
>alt_perf_crit[pcheck++]);
                                ix=ix+12;}
                        else {
                                gotoxy(ix,i); cprintf("%7.2f",ptr-
>alt_perf_crit[pcheck++]);
                                ix=ix+12; }
                        pcnt--;
                }
                ptr=ptr->next;
                i++;
        }
        ptemp=pcheck;
        if(scrn_cnt>1)  {
                gettext(30,2,78,23,buffer[cnt_scrn]);
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[5]);}
        else  {
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[4]);
                }
        colors(LIGHTGRAY,BLACK);
        while((input=getch())==0 && input!=27) {
                switch(getch())
                        {
                        case 59:                        /* help */
                                gotoxy(10,10);
                                cprintf("HELP");
                        break;

                        case 73:                        /* page up */
                                if (pptr)  ptemp=ptemp-4;
                                if(page_flag!=0 && page_flag<scrn_cnt-1)     --page_flag;
                                if(scrn_cnt>1 && cnt_scrn>0)
                                        puttext(30,2,78,23,buffer[--cnt_scrn]);
                        break;

                        case 81:                                                        /*
page down */
                                if(scrn_cnt>1 && (cnt_scrn<scrn_cnt-1))  {
                                        if(pptr && page_flag<scrn_cnt-1)  {
                                                pptr=hold_pptr[++page_flag];
                                                pcnt=4;
                                                window(30,2,78,23);
                                                clrscr();
                                                window(1,1,80,25);
                                                colors(LIGHTGRAY,WHITE);
                                                gotoxy(30,3); cprintf("PERFORMANCE
INDICATORS");
                                                ix=30;
                                                while(pptr && pcnt)  {
```

```
                                                gotoxy(ix,4); cprintf("%s",pptr-
>perf_name);
                                                ix=ix+12;
                                                pptr=pptr->next;
                                                pcnt--;}
                                        window(2,5,78,23);
                                        i=1;
                                        textcolor(BLACK);
                                        ptr=start;
                                        while(ptr && i<19)  {
                                                ix=30;
                                                pcheck=ptemp;
                                                pcnt=4;
                                                while(pcheck<=pcount && pcnt)  {
                                                        if((fabs(ptr-
>alt_perf_crit[pcheck])<.01) || (fabs(ptr->alt_perf_crit[pcheck])>9999.99)) {
                                                                gotoxy(ix,i);
cprintf("%7.2e",ptr->alt_perf_crit[pcheck++]);
                                                                ix=ix+12;}
                                                        else {
                                                                gotoxy(ix,i);
cprintf("%7.2f",ptr->alt_perf_crit[pcheck++]);
                                                                ix=ix+12; }
                                                pcnt--;
                                                }
                                        ptr=ptr->next;
                                        i++;
                                        }
                                gettext(30,2,78,23,buffer[++cnt_scrn]);
                                ptemp=pcheck;
                                }
                                else  {
                                        puttext(30,2,78,23,buffer[++cnt_scrn]);
                                }
                        break;
                        }
                window(1,1,80,25);
                }
        }
window(1,1,80,25);
return;
}
/*************************************************************/
void alt_report1(struct info *defaults)                 /* report for alternatives data
*/
{                                                                                       /*
hardcopy printout */
        FILE *fptr;
        struct alt *ptr;
        struct perf *pptr;
        int pcheck=0,break_cnt=1,pcnt=4,cnt_break=0,ptemp=0,page_cnt=0;
        pptr=begin;
        ptr=start;
        pflag=1;
        break_cnt=screen_check();
        page_cnt=page_check();
        fptr=fopen("prn","wb");
        open_win(1,40,10,61,14,RED,WHITE);
        gotoxy(3,2);  cprintf("Make sure printer");     /* printer warning message */
        gotoxy(3,3);  cprintf(" is online   ");
        gotoxy(3,4);  cprintf("Press any key");
        getch();
        close_win(1,40,10,61,14);

        fprintf(fptr,"                          ALTERNATIVES DATA\n\r\n\r");
        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type);
        fprintf(fptr,"ALTERNATIVE       LCC                PERFORMANCE INDICATOR
VALUES\n\r");
        fprintf(fptr,"   NAME           ($)");
        while (pptr && pcnt)  {
                fprintf(fptr,"%12s",pptr->perf_name);
                pptr=pptr->next;
                pcnt--; }
        fprintf(fptr,"\n\r");
        while (ptr)  {
                pcnt=4;
                pcheck=0;
                fprintf(fptr,"%-14s",ptr->alt_name);
```

```
                fprintf(fptr,"%10.2f",ptr->alt_lcc);
                while(pcheck<=pcount && pcnt)  {
                        fprintf(fptr,"%12.2f",ptr->alt_perf_crit[pcheck++]);
                        pcnt--;}
                fprintf(fptr,"\n\r");
                ptr=ptr->next;
        }
        ptemp=pcheck;
        if(page_cnt||(break_cnt==1))  fprintf(fptr,"\n\r\f");
        else  fprintf(fptr,"\n\r\n\r\n\r");
        cnt_break++;
        while(break_cnt>1 && cnt_break<break_cnt)  {
                pcnt=4;
                if(page_cnt) {
                        fprintf(fptr,"                              ALTERNATIVES
DATA\n\r\n\r");
                        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type);}
                fprintf(fptr,"ALTERNATIVE       LCC                 PERFORMANCE INDICATOR
VALUES\n\r");
                fprintf(fptr,"   NAME           ($)");
                while (pptr && pcnt)  {
                        fprintf(fptr,"%12s",pptr->perf_name);
                        pptr=pptr->next;
                        pcnt--; }
                fprintf(fptr,"\n\r");
                ptr=start;
                while (ptr)  {
                        pcheck=ptemp;
                        pcnt=4;
                        fprintf(fptr,"%-14s",ptr->alt_name);
                        fprintf(fptr,"%10.2f",ptr->alt_lcc);
                        while(pcheck<=pcount && pcnt)  {
                                fprintf(fptr,"%12.2f",ptr->alt_perf_crit[pcheck++]);
                                pcnt--;}
                        fprintf(fptr,"\n\r");
                        ptr=ptr->next;
                }
                ptemp=pcheck;
                cnt_break++;
                if(page_cnt && cnt_break!=break_cnt)  fprintf(fptr,"\n\r\f");
                if(!page_cnt && cnt_break==break_cnt) fprintf(fptr,"\n\r\f");
                else  fprintf(fptr,"\n\r\n\r\n\r");
        }
        fclose(fptr);
}
                                                /* end of alt print out */



/*************************************************************/
void alt_util_report2(struct info *defaults)                    /* report for
alternatives data */
{
        struct alt *ptr;
        struct perf *pptr,*hold_pptr[4];
        struct calc *cptr;
        int i=1,ix=38,pcheck=0,scrn_cnt=1,pcnt=3,cnt_scrn=0,ptemp=0,j=0,k=1,page_flag=0;
        char buffer[4][2646],input;
        scrn_cnt=screen_check1();
        pflag=0;
        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(36,1);
        cprintf("ALTERNATIVES UTILITY DATA");
        textcolor(WHITE);
        gotoxy(4,2); cprintf("LCC TYPE: %s",defaults->alt_lcc_type);
        gotoxy(4,3); cprintf("ALT");
        gotoxy(4,4); cprintf("NUM");
        gotoxy(12,3); cprintf("ALTERNATIVE");
        gotoxy(16,4); cprintf("NAME");
        gotoxy(28,3); cprintf("LCC");
        gotoxy(28,4); cprintf("($)");
        colors(LIGHTGRAY,BLACK);
        window(2,5,79,23);
        clrscr();
```

```
        ptr=start;
        cptr=c_begin;
        while(ptr && cptr && i<19) {
                gotoxy(5,i); cprintf("%u",cptr->alt_num);
                gotoxy(12,i);  cprintf("%s",ptr->alt_name);
                if((fabs(ptr->alt_lcc)<.01) || (fabs(ptr->alt_lcc)>9999.99)) {
                        gotoxy(27,i); cprintf("%7.2e",ptr->alt_lcc);}
                else {
                        gotoxy(27,i); cprintf("%7.2f",ptr->alt_lcc);}
                i++;
                ptr=ptr->next;
                cptr=cptr->next;
        }
        window(1,1,80,25);
        pptr=begin;
        hold_pptr[0]=begin;
        while(k<scrn_cnt)  {
                pcnt=3;
                while(j<pcnt)  {
                        pptr=pptr->next;
                        pcnt--; }
                hold_pptr[k]=pptr;
                k++;
        }
        pptr=begin;
        pcnt=3;
        cptr=c_begin;
        textcolor(WHITE);
        gotoxy(38,3); cprintf("UTILITY SCALE VALUES");
        ix=38;
        while(pptr && pcnt)  {
                gotoxy(ix,4); cprintf("%s",pptr->perf_name);
                ix=ix+12;
                pptr=pptr->next;
                pcnt--;}
        window(2,5,79,23);
        i=1;
        textcolor(BLACK);
        ptr=start;
        while(ptr && i<19)  {
                pcnt=3;
                ix=38;
                pcheck=0;
                while(pcheck<=pcount && pcnt)  {
                        if((fabs(cptr->alt_util_val[pcheck])<.01) || (fabs(cptr-
>alt_util_val[pcheck])>9999.99)) {
                                gotoxy(ix,i); cprintf("%7.2e",cptr-
>alt_util_val[pcheck++]);
                                ix=ix+12;}
                        else {
                                gotoxy(ix,i); cprintf("%7.2f",cptr-
>alt_util_val[pcheck++]);
                                ix=ix+12; }
                        pcnt--;
                }
                ptr=ptr->next;
                cptr=cptr->next;
                i++;
        }
        ptemp=pcheck;
        if(scrn_cnt>1)  {
                gettext(38,2,78,23,buffer[cnt_scrn]);
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[5]);}
        else  {
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[4]);
                }
        colors(LIGHTGRAY,BLACK);
        while((input=getch())==0 && input!=27) {
                switch(getch())
                        {
                        case 59:                    /* help */
                                gotoxy(10,10);
                                cprintf("HELP");
                        break;
```

```
                        case 73:                        /* page up */
                                if (pptr)  ptemp=ptemp-3;
                                if(page_flag!=0 && page_flag<scrn_cnt-1)      --page_flag;
                                if(scrn_cnt>1 && cnt_scrn>0)
                                        puttext(38,2,78,23,buffer[--cnt_scrn]);
                        break;

                        case 81:                                                        /*
page down */
                                if(scrn_cnt>1 && cnt_scrn<(scrn_cnt-1))  {
                                        if(pptr && page_flag<scrn_cnt-1)  {
                                                pptr=hold_pptr[++page_flag];
                                                pcnt=3;
                                                window(38,2,78,23);
                                                clrscr();
                                                window(1,1,80,25);
                                                colors(LIGHTGRAY,WHITE);
                                                gotoxy(38,3); cprintf("UTILITY SCALE
VALUES");
                                                ix=38;
                                                while(pptr && pcnt)  {
                                                        gotoxy(ix,4); cprintf("%s",pptr-
>perf_name);
                                                        ix=ix+12;
                                                        pptr=pptr->next;
                                                        pcnt--;}
                                                window(2,5,78,23);
                                                i=1;
                                                textcolor(BLACK);
                                                ptr=start;
                                                cptr=c_begin;
                                                while(ptr && i<19)  {
                                                        ix=38;
                                                        pcheck=ptemp;
                                                        pcnt=3;
                                                        while(pcheck<=pcount && pcnt)  {
                                                                if((fabs(cptr-
>alt_util_val[pcheck])<.01) || (fabs(cptr->alt_util_val[pcheck])>9999.99)) {
                                                                        gotoxy(ix,i);
cprintf("%7.2e",cptr->alt_util_val[pcheck++]);
                                                                        ix=ix+12;}
                                                                else {
                                                                        gotoxy(ix,i);
cprintf("%7.2f",cptr->alt_util_val[pcheck++]);
                                                                        ix=ix+12; }
                                                        pcnt--;
                                                        }
                                                ptr=ptr->next;
                                                cptr=cptr->next;
                                                i++;
                                                }
                                        gettext(38,2,78,23,buffer[++cnt_scrn]);
                                        ptemp=pcheck;
                                        }
                                        else  {
                                                puttext(38,2,78,23,buffer[++cnt_scrn]);
                                }
                        break;
                        }
                window(1,1,80,25);
                }
        }
window(1,1,80,25);
return;
}
/*************************************************************/
void alt_util_report1(struct info *defaults)          /* report for alternatives
utility data */
{                                                                               /*
hardcopy printout */
        FILE *fptr;
        struct alt *ptr;
        struct perf *pptr;
        struct calc *cptr;
        int pcheck=0,break_cnt=1,pcnt=3,cnt_break=0,ptemp=0,page_cnt=0;
        pptr=begin;
        ptr=start;
```

```
        cptr=c_begin;
        pflag=1;
        break_cnt=screen_check1();
        page_cnt=page_check1();
        fptr=fopen("prn","wb");
        open_win(1,40,10,61,14,RED,WHITE);
        gotoxy(3,2);  cprintf("Make sure printer");     /* printer warning message */
        gotoxy(3,3);  cprintf("   is online   ");
        gotoxy(3,4);  cprintf("  Press any key");
        getch();
        close_win(1,40,10,61,14);

        fprintf(fptr,"                          ALTERNATIVES UTILITY DATA\n\r\n\r");
        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type);
        fprintf(fptr,"ALT    ALTERNATIVE       LCC       PERFORMANCE INDICATOR UTILITY
VALUES\n\r");
        fprintf(fptr,"NUM       NAME           ($)");
        while (pptr && pcnt)  {
                fprintf(fptr,"%13s",pptr->perf_name);
                pptr=pptr->next;
                pcnt--; }
        fprintf(fptr,"\n\r");
        while (ptr && cptr)  {
                pcnt=3;
                pcheck=0;
                fprintf(fptr," %-5u",cptr->alt_num);
                fprintf(fptr,"%-14s",ptr->alt_name);
                fprintf(fptr,"%10.2f",ptr->alt_lcc);
                while(pcheck<=pcount && pcnt)  {
                        fprintf(fptr,"%12.2f",cptr->alt_util_val[pcheck++]);
                        pcnt--;}
                fprintf(fptr,"\n\r");
                ptr=ptr->next;
                cptr=cptr->next;
        }
        ptemp=pcheck;
        if(page_cnt||(break_cnt==1))  fprintf(fptr,"\n\r\f");
        else  fprintf(fptr,"\n\r\n\r\n\r");
        cnt_break++;
        while(break_cnt>1 && cnt_break<break_cnt)  {
                pcnt=3;
                if(page_cnt) {
                        fprintf(fptr,"                          ALTERNATIVES UTILITY
DATA\n\r\n\r");
                        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type);}
                fprintf(fptr,"ALT    ALTERNATIVE       LCC       PERFORMANCE INDICATOR
UTILITY VALUES\n\r");
                fprintf(fptr,"NUM       NAME           ($)");
                while (pptr && pcnt)  {
                        fprintf(fptr,"%13s",pptr->perf_name);
                        pptr=pptr->next;
                        pcnt--; }
                fprintf(fptr,"\n\r");
                ptr=start;
                cptr=c_begin;
                while (ptr && cptr)  {
                        pcheck=ptemp;
                        pcnt=3;
                        fprintf(fptr," %-5u",cptr->alt_num);
                        fprintf(fptr,"%-14s",ptr->alt_name);
                        fprintf(fptr,"%10.2f",ptr->alt_lcc);
                        while(pcheck<=pcount && pcnt)  {
                                fprintf(fptr,"%12.2f",cptr->alt_util_val[pcheck++]);
                                pcnt--;}
                        fprintf(fptr,"\n\r");
                        ptr=ptr->next;
                        cptr=cptr->next;
                }
                ptemp=pcheck;
                cnt_break++;
                if(page_cnt && cnt_break!=break_cnt)  fprintf(fptr,"\n\r\f");
                if(!page_cnt && cnt_break==break_cnt) fprintf(fptr,"\n\r\f");
                else  fprintf(fptr,"\n\r\n\r\n\r");
        }
        fclose(fptr);
}
                                                        /* end of alt util print out */
```

```
/*************************************************************/
void weight_score_report2(struct info *defaults)                    /* screen
report for alternatives */
{                                                                   /* weighted score data */
        struct perf *pptr,*hold_pptr[4];
        struct calc *cptr,*hold[MAXALT],*temp;
        int
i=1,ix=38,pcheck=0,scrn_cnt=1,pcnt=3,cnt_scrn=0,ptemp=0,j=0,k=0,in=0,out=0,sflag=2,page_
flag=0,m=0;
        char buffer[4][2646],input;

        open_win(1,40,10,65,15,BLUE,WHITE);
        gotoxy(3,2);  cprintf("Press 1 to sort");     /* select sorting */
        gotoxy(3,3);  cprintf("on weighted index.");
        gotoxy(3,4);  cprintf("Press any other key");
        gotoxy(3,5);  cprintf("for no sorting.");
        if(getch()==49)  sflag=1;
        else sflag=2;
        close_win(1,40,10,65,15);
        cptr=c_begin;
        while(cptr)  {
                hold[j++]=cptr;
                cptr=cptr->next;}
        if(sflag==1)  {
                for(out=0;out<j-1;out++)
                        for(in=out+1;in<j;in++)
                                if(hold[out]->alt_wt_ind<hold[in]->alt_wt_ind)  {
                                        temp=hold[in];
                                        hold[in]=hold[out];
                                        hold[out]=temp;
                                }
        }
        window(1,1,80,25);
        scrn_cnt=screen_check1();
        window(1,1,80,25);
        textbackground(LIGHTGRAY);
        clrscr();
        border(BLUE,1,1,80,24);
        gotoxy(28,1);
        cprintf("ALTERNATIVES WEIGHTED SCORE DATA");
        textcolor(WHITE);
        gotoxy(4,2); cprintf("LCC TYPE: %s",defaults->alt_lcc_type);
        gotoxy(4,3); cprintf("ALT");
        gotoxy(4,4); cprintf("NUM");
        gotoxy(12,3); cprintf("ALTERNATIVE");
        gotoxy(16,4); cprintf("NAME");
        gotoxy(28,3); cprintf("LCC");
        gotoxy(28,4); cprintf("($)");
        colors(LIGHTGRAY,BLACK);
        window(2,5,79,23);
        clrscr();
        while(k<j && i<19) {
                gotoxy(5,i); cprintf("%u",hold[k]->alt_num);
                gotoxy(12,i);  cprintf("%s",hold[k]->altname);
                if((fabs(hold[k]->altlcc)<.01) || (fabs(hold[k]->altlcc)>9999.99)) {
                        gotoxy(27,i); cprintf("%7.2e",hold[k]->altlcc);}
                else {
                        gotoxy(27,i); cprintf("%7.2f",hold[k]->altlcc);}
                i++;
                k++;
        }
        window(1,1,80,25);
        pptr=begin;
        hold_pptr[0]=begin;
        k=1;
        m=0;
        while(k<scrn_cnt)  {
                pcnt=3;
                while(m<pcnt && pptr)  {
                        pptr=pptr->next;
                        pcnt--; }
                hold_pptr[k]=pptr;
                k++;
        }
        pcnt=3;
```

```
        pptr=begin;
        textcolor(WHITE);
        gotoxy(38,3); cprintf("WEIGHTED SCORES");
        ix=38;
        while(pptr && pcnt)  {
                gotoxy(ix,4); cprintf("%s",pptr->perf_name);
                ix=ix+11;
                pptr=pptr->next;
                pcnt--;}
        gotoxy(71,3);  cprintf("WEIGHTED");
        gotoxy(71,4);  cprintf(" INDEX");
        window(2,5,79,23);
        i=1;
        k=0;
        textcolor(BLACK);
        while(k<j && i<19)  {
                pcnt=3;
                ix=38;
                pcheck=0;
                while(pcheck<=pcount && pcnt)  {
                        if((fabs(hold[k]->alt_wt_score[pcheck])<.01) || (fabs(hold[k]-
>alt_wt_score[pcheck])>9999.99)) {
                                gotoxy(ix,i); cprintf("%7.2e",hold[k]-
>alt_wt_score[pcheck++]);
                                ix=ix+11;}
                        else {
                                gotoxy(ix,i); cprintf("%7.2f",hold[k]-
>alt_wt_score[pcheck++]);
                                ix=ix+11; }
                        pcnt--;
                }
                if((fabs(hold[k]->alt_wt_ind)<.01) || (fabs(hold[k]-
>alt_wt_ind)>9999.99)) {
                        gotoxy(70,i);  cprintf("%6.2e",hold[k]->alt_wt_ind);
                }
                else {
                        gotoxy(70,i);  cprintf("%7.2f",hold[k]->alt_wt_ind);
                }
                k++;
                i++;
        }
        ptemp=pcheck;
        if(scrn_cnt>1)  {
                gettext(38,2,78,23,buffer[cnt_scrn]);
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[5]);}
        else  {
                window(1,1,80,25);
                colors(RED,WHITE);
                message(messages[4]);
                }
        colors(LIGHTGRAY,BLACK);
        while((input=getch())==0 && input!=27) {
                switch(getch())
                        {
                        case 59:                    /* help */
                                gotoxy(10,10);
                                cprintf("HELP");
                        break;

                        case 73:                    /* page up */
                                if (pptr)  ptemp=ptemp-3;
                                if(page_flag!=0 && page_flag<(scrn_cnt-1))   --page_flag;
                                if(scrn_cnt>1 && cnt_scrn>0)
                                        puttext(38,2,78,23,buffer[--cnt_scrn]);
                        break;

                        case 81:                                                    /*
page down */
                                if(scrn_cnt>1 && cnt_scrn<(scrn_cnt-1))  {
                                        if(pptr && page_flag<(scrn_cnt-1))  {
                                                pptr=hold_pptr[++page_flag];
                                                pcnt=3;
                                                window(38,2,78,23);
                                                clrscr();
                                                window(1,1,80,25);
                                                colors(LIGHTGRAY,WHITE);
```

```
                                                gotoxy(38,3); cprintf("WEIGHTED SCORES");
                                                ix=38;
                                                while(pptr && pcnt)  {
                                                        gotoxy(ix,4); cprintf("%s",pptr-
>perf_name);
                                                        ix=ix+11;
                                                        pptr=pptr->next;
                                                        pcnt--;}
                                                gotoxy(71,3);  cprintf("WEIGHTED");
                                                gotoxy(71,4);  cprintf(" INDEX");
                                                window(2,5,78,23);
                                                i=1;
                                                k=0;
                                                textcolor(BLACK);
                                                while(k<j && i<19)  {
                                                        ix=38;
                                                        pcheck=ptemp;
                                                        pcnt=3;
                                                        while(pcheck<=pcount && pcnt)  {
                                                                if((fabs(hold[k]-
>alt_wt_score[pcheck])<.01) || (fabs(hold[k]->alt_wt_score[pcheck])>9999.99)) {
                                                                        gotoxy(ix,i);
cprintf("%7.2e",hold[k]->alt_wt_score[pcheck++]);
                                                                        ix=ix+11;}
                                                                else {
                                                                        gotoxy(ix,i);
cprintf("%7.2f",hold[k]->alt_wt_score[pcheck++]);
                                                                        ix=ix+11; }
                                                        pcnt--;
                                                        }
                                                if((fabs(hold[k]->alt_wt_ind)<.01) ||
(fabs(hold[k]->alt_wt_ind)>9999.99)) {
                                                        gotoxy(70,i);
cprintf("%6.2e",hold[k]->alt_wt_ind);
                                                        }
                                                else {
                                                        gotoxy(70,i);
cprintf("%7.2f",hold[k]->alt_wt_ind);
                                                        }
                                                k++;
                                                i++;
                                                }
                                        gettext(38,2,78,23,buffer[++cnt_scrn]);
                                        ptemp=pcheck;
                                        }
                                        else  {
                                                puttext(38,2,78,23,buffer[++cnt_scrn]);
                                }
                        break;
                        }
                window(1,1,80,25);
                }
        }
window(1,1,80,25);
return;
}

/*************************************************************/
void weight_score_report1(struct info *defaults)                        /* hardcopy
for alternatives */
{                                                               /* weighted scores data */
        FILE *fptr;
        struct perf *pptr;
        struct calc *cptr,*hold[MAXALT],*temp;
        int
pcheck=0,break_cnt=1,pcnt=3,cnt_break=0,ptemp=0,j=0,k=0,in=0,out=0,sflag=2,page_cnt=0;;
        open_win(1,40,10,65,15,BLUE,WHITE);
        gotoxy(3,2);  cprintf("Press 1 to sort");     /* select sorting */
        gotoxy(3,3);  cprintf("on weighted index.");
        gotoxy(3,4);  cprintf("Press any other key");
        gotoxy(3,5);  cprintf("for no sorting.");
        if(getch()==49)  sflag=1;
        else sflag=2;
        close_win(1,40,10,65,15);
        cptr=c_begin;
        while(cptr)  {
                hold[j++]=cptr;
                cptr=cptr->next;}
```

```
        if(sflag==1)  {
                for(out=0;out<j-1;out++)
                        for(in=out+1;in<j;in++)
                                if(hold[out]->alt_wt_ind<hold[in]->alt_wt_ind)  {
                                        temp=hold[in];
                                        hold[in]=hold[out];
                                        hold[out]=temp;
                                }
        }
        pflag=1;
        window(1,1,80,25);
        break_cnt=screen_check1();
        page_cnt=page_check1();
        fptr=fopen("prn","wb");
        open_win(1,40,10,61,14,RED,WHITE);
        gotoxy(3,2);  cprintf("Make sure printer");     /* printer warning message */
        gotoxy(3,3);  cprintf("    is online    ");
        gotoxy(3,4);  cprintf("  Press any key");
        getch();
        close_win(1,40,10,61,14);
        pptr=begin;

        fprintf(fptr,"                        ALTERNATIVES WEIGHTED SCORE DATA\n\r\n\r");
        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type);
        fprintf(fptr,"ALT    ALTERNATIVE       LCC       WEIGHTED SCORES
WEIGHTED\n\r");
        fprintf(fptr,"NUM       NAME           ($)");
        while(pptr && pcnt)  {
                fprintf(fptr,"%12s",pptr->perf_name);
                pptr=pptr->next;
                pcnt--; }
        while(pcnt)  {
                fprintf(fptr,"           ");
                pcnt--;}
        fprintf(fptr,"        INDEX\n\r");

        while(k<j) {
                pcnt=3;
                fprintf(fptr," %-5u",hold[k]->alt_num);
                fprintf(fptr,"%-14s",hold[k]->altname);
                fprintf(fptr,"%10.2f",hold[k]->altlcc);
                pcheck=0;
                while(pcheck<=pcount && pcnt)  {
                        fprintf(fptr,"%12.2f",hold[k]->alt_wt_score[pcheck++]);
                        pcnt--;
                }
                while(pcnt)  {
                        fprintf(fptr,"            ");
                        pcnt--;}
                fprintf(fptr,"%12.2f\n\r",hold[k]->alt_wt_ind);
                k++;
        }
        ptemp=pcheck;
        if(page_cnt||(break_cnt==1))  fprintf(fptr,"\n\r\f");
        else fprintf(fptr,"\n\r\n\r\n\r");
        cnt_break++;
        while(break_cnt>1 && cnt_break<break_cnt)  {
                pcnt=3;
                if(page_cnt)  {
                        fprintf(fptr,"                       ALTERNATIVES WEIGHTED SCORE
DATA\n\r\n\r");
                        fprintf(fptr,"LCC TYPE: %s\n\r\n\r",defaults->alt_lcc_type); }
                fprintf(fptr,"ALT    ALTERNATIVE       LCC       WEIGHTED SCORES
WEIGHTED\n\r");
                fprintf(fptr,"NUM       NAME           ($)");
                while(pptr && pcnt)  {
                        fprintf(fptr,"%12s",pptr->perf_name);
                        pptr=pptr->next;
                        pcnt--; }
                while(pcnt)  {
                        fprintf(fptr,"           ");
                        pcnt--;}
                fprintf(fptr,"        INDEX\n\r");
                k=0;
                while(k<j) {
                        pcnt=3;
                        fprintf(fptr," %-5u",hold[k]->alt_num);
                        fprintf(fptr,"%-14s",hold[k]->altname);
```

```
                        fprintf(fptr,"%10.2f",hold[k]->altlcc);
                        pcheck=ptemp;
                        while(pcheck<=pcount && pcnt)  {
                                fprintf(fptr,"%12.2f",hold[k]->alt_wt_score[pcheck++]);
                                pcnt--;
                        }
                        while(pcnt)  {
                                fprintf(fptr,"            ");
                                pcnt--;}
                        fprintf(fptr,"%12.2f\n\r",hold[k]->alt_wt_ind);
                        k++;
                }
                ptemp=pcheck;
                cnt_break++;
                if(page_cnt && cnt_break!=break_cnt)  fprintf(fptr,"\n\r\f");
                if(!page_cnt && cnt_break==break_cnt) fprintf(fptr,"\n\r\f");
                else fprintf(fptr,"\n\r\n\r\n\r");
        }
        fclose(fptr);
}

/*************************************************************/
int screen_check()
{
        int scrn;
        div_t cnt;

        scrn=pcount+1;
        cnt=div(scrn,4);
        if(cnt.rem>0)  cnt.quot=(cnt.quot)+1;
        return (cnt.quot);
}
/*************************************************************/
int screen_check1()
{
        int scrn;
        div_t cnt;

        scrn=pcount+1;
        cnt=div(scrn,3);
        if(cnt.rem>0)  cnt.quot=(cnt.quot)+1;
        return (cnt.quot);
}
/*************************************************************/
int select_report()
{
        int m=2,n=0;
        char move;
        static char *selections[5]={
                " 1. Performance criteria/utility curve table",
                " 2. Alternatives table      ",
                " 3. Alternatives utility values",
                " 4. Weighted score table     ",
                " 5. Return to main menu     ",
        };

        window(15,3,62,9);
        colors(BLUE,YELLOW);
        clrscr();
        textbackground(BLUE);
        border(YELLOW,15,3,62,9);
        for(n=0;n<5;n++) {
                gotoxy(3,m++); cprintf("%s",selections[n]);
        }
        colors(CYAN,WHITE);                     /* hilite first item       */
        gotoxy(3,2); cprintf("%s",selections[0]);
        m=2,n=0;                                /* set counters to first   */

        while((move=getch()) !=13 && move !=27)       /* check for return */
        {
                if(move==0)                          /* check for extended code */
                {
                        switch(getch())
                        {
                                case 80:                        /* down cursor */
                                        colors(BLUE,YELLOW);
                                        gotoxy(3,m); cprintf("%s",selections[n++]);
                                        if(n>4) {m=1;n=0;}
```

```
                                colors(CYAN,WHITE);
                                gotoxy(3,++m); cprintf("%s",selections[n]);
                        break;

                        case 72:                        /* up cursor */
                                colors(BLUE,YELLOW);
                                gotoxy(3,m); cprintf("%s",selections[n--]);
                                if(n<0) {m=7,n=4;}
                                colors(CYAN,WHITE);
                                gotoxy(3,--m); cprintf("%s",selections[n]);
                        break;

                        case 59:                        /* F1 help key */
                                gotoxy(10,10);
                                printf("HELP");
                                break;


                        case 61:                        /* F3 print toggle key */
                                if(pflag==0) {
                                        pflag=1;
                                        window(1,1,80,25);
                                        window(25,15,55,18);
                                        clrscr();
                                        textbackground(BLUE);
                                        border(WHITE,25,15,55,18);
                                        gotoxy(2,2); cprintf("Current output to
printer    ");
                                        gotoxy(2,3); cprintf("Press F3 to output to
screen ");
                                        window(15,3,62,9);}

                                else {
                                        pflag=0;
                                        window(1,1,80,25);
                                        window(25,15,55,18);
                                        clrscr();
                                        textbackground(BLACK);
                                        border(WHITE,25,15,55,18);
                                        gotoxy(2,2); cprintf("Current output to
screen     ");
                                        gotoxy(2,3); cprintf("Press F3 to output to
printer");
                                        window(15,3,62,9);
                                }

                        break;
                }

        }
        if(move>48&&move<58) return (move-48);
    }
    if(move==27) return(20);
    window(1,1,80,25);
    return(++n);
}

/*************************************************************************/

void draw_report_screen()
{
    window(1,1,80,25);
    clrscr();
    textbackground(LIGHTGRAY);
    window(1,1,80,25);
    clrscr();
    border(WHITE,1,1,80,24);
    gotoxy(36,1); cprintf("REPORT MENU");
    if(pflag==0)  {
        window(25,15,55,18);
        clrscr();
        textbackground(BLACK);
        border(WHITE,25,15,55,18);
        gotoxy(2,2); cprintf("Current output to screen     ");
        gotoxy(2,3); cprintf("Press F3 to output to printer"); }
    else  {
        window(25,15,55,18);
        clrscr();
```

```
                textbackground(BLUE);
                border(WHITE,25,15,55,18);
                gotoxy(2,2); cprintf("Current output to printer    ");
                gotoxy(2,3); cprintf("Press F3 to output to screen ");}
        window(1,1,80,25);
        message(messages[6]);


}

/*************************************************************************/
int page_check()
{
        struct alt *ptr;
        int page,scrn,line_cnt,i=0;
        div_t cnt;

        ptr=start;

        while(ptr) {
                i++;
                ptr=ptr->next;}

        line_cnt=i+8;

        scrn=pcount+1;
        cnt=div(scrn,4);
        if(cnt.rem>0)  cnt.quot=(cnt.quot)+1;

        if((line_cnt*cnt.quot)<60)  page=0;
        else page=1;

        return (page);
}

/*************************************************************************/
int page_check1()
{
        struct alt *ptr;
        int page,scrn,line_cnt,i=0;
        div_t cnt;

        ptr=start;

        while(ptr) {
                i++;
                ptr=ptr->next;}

        line_cnt=i+8;

        scrn=pcount+1;
        cnt=div(scrn,3);
        if(cnt.rem>0)  cnt.quot=(cnt.quot)+1;

        if((line_cnt*cnt.quot)<60)  page=0;
        else page=1;

        return (page);
}
```

SECTION 5.7 SETUP.C

```
/* SETUP.C  ............. Setup module for DED */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include "ded.h"
#include <dir.h>



extern char *messages[];

int get_input(int,int,int,char *,int,int);
struct info *save_setup(struct info *);
```

```
struct info *setup(struct info *std_vals)
{
        char curdir[64];
        int j=1,k=1;

        message(messages[2]);
        open_win(1,20,7,60,17,RED,WHITE);
        gotoxy(19,1); cprintf("SETUP");
        colors(RED,WHITE);
        gotoxy(3,2); cprintf("Program Drive:");
        gotoxy(3,3); cprintf("Data Drive:");
        gotoxy(3,4); cprintf("Program Directory:");
        gotoxy(3,5); cprintf("Data Directory:");
        gotoxy(3,6); cprintf("Current File:");
        gotoxy(3,7); cprintf("LCC Type:");

        /* display non-changeable variables */

        textcolor(BLACK);
        gotoxy(18,2); cprintf("%c:",'A'+getdisk()); /* program drive */
        getcurdir(0,curdir);
        gotoxy(22,4); cprintf("\\%s",curdir);       /* program directory */
        gotoxy(15,3); cprintf("%c:",'A'+getdisk());  /* data drive */
        gotoxy(19,5); cprintf("\\%s",curdir);       /* data directory */

        /* display changeable variables */

        colors(BLUE,YELLOW);


        gotoxy(17,6); cprintf("%s",std_vals->name);        /* data file */
        gotoxy(15,7); cprintf("%s",std_vals->alt_lcc_type);  /* LCC type */

        colors(CYAN,BLACK);
        gotoxy(17,6); cprintf("%-9s",std_vals->name);       /* get file name */
        gotoxy(15,7); cprintf("%-3s",std_vals->alt_lcc_type);
        while (j && j!=27)  {
                switch(k)  {
                        case 1:
                        j=get_input(16,6,9,std_vals->name,MAGENTA,WHITE);
                        k=(j<0) ? 1:2;
                        break;

                        case 2:
                        j=get_input(14,7,4,std_vals->alt_lcc_type,MAGENTA,WHITE);  /* get
LCC type */
                        j=0;
                        break;
                }
        }
        close_win(1,20,7,60,17);
        save_setup(std_vals);                   /* save setup data */
        window(1,1,80,25);
        return 0;
}

/*********************************************************************/


struct info *read_setup(struct info *data)
{
        FILE *fp;
        if((fp=fopen("SETUP.DAT","rb"))==NULL)
                cprintf("Cannot open file\n");
        else {
                fread(data,sizeof(struct info),1,fp);
                fclose(fp);
        }
        return 0;
}

/*********************************************************************/


struct info *save_setup(struct info *data)
{
```

```
        FILE *fp;
        if((fp=fopen("SETUP.DAT","wb"))==NULL) {
                cprintf("Cannot open file\n");
                exit(1);
        }
        fwrite(data,sizeof(struct info),1,fp);
        fclose(fp);
        return 0;
}
```