Micro Computer Feedback Report for the Strategic Leader Development Inventory

James E. Hopkins
U.S. Army Research Institute

May 1993

United States Army Research Institute for the Behavioral and Social Sciences

Approved for public release; distribution is unlimited.
**Title:** Micro Computer Feedback Report for the Strategic Leader Development Inventory

**Performance Organization:** U.S. Army Research Institute for the Behavioral and Social Sciences

**Author(s):** Hopkins, James E. (ARI) TCN 92-272

**Abstract:**
This report describes the FeedBack micro computer program written to print reports for participants who have responded to the preliminary form of the Strategic Leader Development Inventory (SLDI). The SLDI is a self-assessment inventory enabling comparison of self-ratings on a number of positive and negative leadership dimensions with those from former superiors, peers, and subordinates. The final form of the SLDI is now being developed by the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) Strategic Leadership Technical Area, in collaboration with the U.S. Army War College and the Industrial College of the Armed Forces.

In its present form, the FeedBack program produces a 2-page assessment containing eight graphs for each participant, reflecting self-ratings compared with those from others. Future plans call for revision of the SLDI based on factor analysis of data obtained during academic year 1992. The revision will produce SLDI forms with fewer items and a cleaner factor structure. The revised feedback will then require modification of the code described in this report.

**Subject Terms:** Strategic Leader Development Inventory, Computer Printer Control Language, Assembler, Leadership, ASM

**Distribution Statement:** Approved for public release; distribution is unlimited.
Micro Computer Feedback Report for the Strategic Leader Development Inventory

James E. Hopkins
U.S. Army Research Institute

Strategic Leadership Technical Area
Thomas O. Jacobs, Chief

Manpower and Personnel Research Division
Zita M. Simutis, Director

U.S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue, Alexandria, Virginia 22333-5600

Office, Deputy Chief of Staff for Personnel
Department of the Army

May 1993

Army Project Number
2Q162785A791

Manpower, Personnel, and Training

Approved for public release; distribution is unlimited.
Enhancing the continued growth of leadership skills is a central element in the Senior Service College mission. Feedback from others is a useful tool for that purpose.

The Strategic Leadership Development Inventory (SLDI) is a questionnaire designed to obtain feedback from seniors, peers, and subordinates on dimensions of leader actions and attributes thought to be important for senior leader development. In practice, perceptions from each of these sources can be compared both among themselves and with perceptions of the individual who provides a self-description. The comparison can provide insights about an individual's "blind spots" and indications about how future development might be guided.

A crucial element using the SLDI is the cost-effective generation of feedback to leaders in a form that truly aids understanding and development. The present report documents the development of software that will accept formatted data from a mark-sense scoring machine, perform the statistical computations necessary to develop feedback sheets for individual students, and then print the sheets.

This work was made possible by the U.S. Army Summer Associateship Program for High School Science and Mathematics Faculty, through which expertise was made available for the critical software development part of the project.

EDGAR M. JOHNSON
Acting Director
ACKNOWLEDGMENTS

As a member of the Largo High School Mathematics Department, I participated in the U.S. Army Summer Associateship Program for High School Science and Mathematics Faculty. My summer associateship was supported by the U.S. Army Research Institute for the Behavioral and Social Sciences, Strategic Leadership Technical Area, Dr. T. O. Jacobs, Chief, under the auspices of the U.S. Army Research Office Scientific Sciences Program administered by Battelle.

I wish to express my appreciation to Dr. Jacobs and Dr. Stewart for allowing me to be part of the Strategic Leader Development Inventory project. They told me what they wanted the FeedBack program to accomplish and gave me the responsibility of completing the task.
MICRO COMPUTER FEEDBACK REPORT FOR THE STRATEGIC LEADER DEVELOPMENT INVENTORY

EXECUTIVE SUMMARY

Requirement:

To develop an automated capability for generating completed feedback forms for Senior Service College students who had completed (and on whom former superiors, peers, and subordinates had completed) the Strategic Leader Development Inventory (SLDI).

Procedure:

The SLDI was generated from descriptions of effective and ineffective senior leader behavior. Content analysis of these descriptions generated dimensions that were then represented by logical clusters of items. These items made up the preliminary form of the SLDI. To provide feedback to participating students, code was written to accept data files generated from scanning scoring sheets. The code computes quartile points for all four data distributions and then prints feedback sheets showing the distribution of responses for each dimension; first, second, and third quartiles; the individual's self-rating; and the rating of that individual by former superiors, peers, and subordinates.

Findings:

The individual may thus compare himself or herself with others in the same class, and with the perceptions of these significant others from whom data were obtained about himself or herself.

Utilization of Findings:

The feedback system was used for the academic year 1992 classes at the U.S. Army War College and the Industrial College of the Armed Forces. It will be revised as the SLDI is revised for academic year 1993 and subsequent years, based on factor analysis of 1992 data, and will become operational at both as an additional tool for leader development.
# MICRO COMPUTER FEEDBACK REPORT FOR THE STRATEGIC LEADER DEVELOPMENT INVENTORY

## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>SLDI DATA FILES</td>
<td>1</td>
</tr>
<tr>
<td>FEEDBACK DOCUMENTATION</td>
<td>3</td>
</tr>
<tr>
<td>APPENDIX A. SAMPLE FEEDBACK GRAPHS</td>
<td>A-1</td>
</tr>
<tr>
<td>B. DESCRIPTION OF SLDI</td>
<td>B-1</td>
</tr>
<tr>
<td>C. CONVERTING THE POS FILE</td>
<td>C-1</td>
</tr>
<tr>
<td>D. FORMAT FOR FEEDBACK DATA FILES</td>
<td>D-1</td>
</tr>
<tr>
<td>E. LOCATING WORDPERFECT'S PRINTER PORT</td>
<td>E-1</td>
</tr>
<tr>
<td>F. FORMAT FOR FEEDBACK VARIABLE STRINGS</td>
<td>F-1</td>
</tr>
<tr>
<td>G. SOURCE CODE FOR THE FEEDBACK PROGRAM</td>
<td>G-1</td>
</tr>
</tbody>
</table>
MICRO COMPUTER FEEDBACK REPORT FOR THE STRATEGIC LEADER DEVELOPMENT INVENTORY

INTRODUCTION

The Strategic Leader Development Inventory (SLDI) is currently under development by the U.S. Army Research Institute for the Behavioral and Social Sciences (ARI), Strategic Leadership Technical Area (SLTA), in collaboration with the U.S. Army War College (USAWC) and the Industrial College of the Armed Forces (ICAF). The SLDI is a self-assessment survey to enable participants to learn more about themselves.

Participants in the SLDI complete a self-assessment survey and select three superiors, three peers, and four subordinates to evaluate them. After all the surveys are completed, the participants receive a report summarizing the results. The SLDI reports enable the participants to learn how they are perceived by their superiors, peers, and subordinates. Coupling these reports with their own ratings, the participants can develop a better understanding of their strengths and weaknesses.

My assignment was to write a micro computer program that would print graphs for the SLDI reports. Because the SLDI is designed to provide the participants with information about themselves, I chose to name the micro computer program FeedBack.

The FeedBack program produces a 2-page assessment containing eight graphs. Each graph displays a participant’s evaluation and the average rating of his or her peers, superiors, and subordinates. The first page contains four positive factor graphs entitled the Success Factors. The second page contains four negative factor graphs entitled Failure Factors. Appendix A contains a sample of the FeedBack graphs.

Accompanying the FeedBack graphs is a summary of the goals of the SLDI and the characteristics included in each factor. A copy appears in Appendix B. In addition to the written description, the participants discuss their assessments in group forums.

SLDI DATA FILES

Before the SLDI answers are ready for the FeedBack program, they must be processed by two other computer programs. An optical scanner transfers the data from the answer sheets to a computer disk. A statistical program is used for factor analysis and to compute the average score used by the FeedBack program.

I helped compile the computer data from the SLDI answer sheets for the ICAF class of 93. An optical scanner at the Army War College was used to read the answer sheets. Each side of the answer sheet was read into a separate ASCII data file. I used the following names for the data files: Self_IA, Self_IB, Self_II, Super_I, Super_II, Peer_I, Peer_II, Subor_I, and Subor_II. The "I" in the file name means part one of the SLDI data or the positive questions. The "II" on the file name means part two of the SLDI data or the negative questions. The "Self" part one data is in two files because there were more than 100 questions. Self_IA contains the answer to the first 100 questions. Self_IB contains the answers beginning with question 101.
The information from one answer sheet became a single line in the data file. The first three digits of the line are the participant's identification number (ID). The first answer will be in column 4, the next answer in column 5, etc. The scanner converted the letters from the answer sheet into numbers (A = 5, B = 4, C = 3, D = 2, E = 1). A space means no answer was given, and an underline means the scanner could not determine the answer.

The optical scanner does a good job, but there are going to be mistakes in the data files. I used an ASCII text editor to correct data errors. Most of the errors were the result of human mistakes, like marking two answers to the same question. This would appear in the data file as an underline. This type of error occurs when an answer is changed and the first answer is not completely erased. I used the original answer sheet to determine the desired response. If one mark is not darker than the other, the underscore is converted to a space, meaning no answer.

I checked the first three digits of each line to make sure it was a valid ID number. Errors occur when the participant forgets to bubble in the ID number on the answer sheet. This then appears as three spaces at the beginning of the data line. I located the original answer sheet and added the missing ID number to the data file.

I discovered that the scanner made errors. The most common error occurred when an answer was left blank. The scanner always marked a blank answer with a space. Sometimes the scanner added an extra space to the data line. The extra space moves all the other answers one digit to the right. The answers do not match the questions. Fortunately this type of error is easy to detect. The number of answers in each data line are the same; therefore, each data line should be the same length. I used an ASCII text editor to check the end of each line. If a line was not the correct length, I compared the scanned answers to the original answer sheet to locate the error.

Analysis of the answers was accomplished with the statistical program "SPSS." I created a DERAIL file by combining all the part II files into one data file named DERAIL. Before combining the data files, I inserted a letter after the ID number. The letter is needed to identify the data as Self, Peer, Superior, or Subordinate. I used "A" for Superior, "B" for Peer, "C" for Superior, and "D" for Self. I recommend that in the future SPSS use the four separate part II files as input. This change will eliminate the need for creating a DERAIL file.

The SPSS analysis of the data created two data files containing average scores for the FeedBack program. Names of the Feedback data files must end with ".POS" for the Success Factors and ".NEG" for the Failure Factors. The NEG file created by SPSS was ready for printing using the FeedBack program.

The POS file had extra data and was edited using an ASCII text editor. Appendix C contains instructions for converting the POS file to the format needed by the FeedBack program. I recommend that in the future SPSS create a POS file without the extra data. The correct format for the POS and NEG FeedBack data files is listed in Appendix D.
FEEDBACK DOCUMENTATION

I wrote the FeedBack program using 8086 micro assembler language. It will operate correctly on a computer using an MS DOS or a compatible operating system. The FeedBack program is designed to print FeedBack graphs on a Hewlett Packard LaserJet or compatible printer. A compatible printer must support Hewlett Packard's Printer Control Language (PCL).

The FeedBack program is menu driven using a Lotus style menu bar. The first line of the menu lists the commands and the second line is a sentence describing the highlighted command. An information box at the bottom of the screen provides instructions for operating the menu bar.

An important feature of the FeedBack program is the use of the <Esc> key to cancel a command. Users feel in control of the program when they can undo a command by pressing the <Esc> key.

Between the Menu Bar and the program title there is a two line Program Status Display. If a data file is open, it will show the file name and the number of ID's contained in the file. Before the graphs can be printed, the data file must be ranked to compute the percentiles. The Program Status Display informs the user if the percentiles are computed. It also lists the status of the selected printer port. It must say "printer is ready" before the report can be printed.

The term "printer is ready" is misleading. It means there is a peripheral attached to the LPT port that is ready to receive data. FeedBack assumes the peripheral is a printer. If the user is not sure which port the printer is using, there are two choices. The easiest may be the trial and error method. Make sure the printer is turned on and use the "LPT" command to select one of the three parallel ports. If FeedBack says the "printer is ready" try printing a report.

Another method to locate the printer port is to examine a program that uses the printer. The most common printing program is a word processor. Use the word processors documentation to learn what port the word processor is using. See Appendix E for instructions on using WordPerfect to learn the printer port assignment.

If the printer is connected to one of the COM ports, the DOS "MODE" command can be used to redirect LPT1 to the desired COM port. Consult the DOS documentation for instructions on use of the "MODE" command. After LPT1 has been reassigned, FeedBack will think it is sending data to LPT1 but DOS will be redirecting the output to the assigned COM port. The ICAF graphs for the class of 93 were printed using a computer with a Zenith DOS operating system. The Zenith "CONFIGUR" command was used to redirect the LPT1 output to the desired COM port.

When using the FeedBack program with a monochrome monitor, check to see if the words in the second line of the menu are clearly displayed. If the words are difficult to
read, restart FeedBack with the command "FeedBack m". Some laptop computers use monochrome monitors but operate in a color video mode. The "m" command can be used to force the Feedback program into the monochrome mode.

When beginning the FeedBack program, the user should select the "File" command which allows FeedBack to locate the SPSS files containing the report data. The name of the "Positive Factors" data file ends with ".POS". The name of the "Failure Factors" data file name ends with ".NEG". Both data files are stored in ASCII and can be read using an ASCII text editor. WordPerfect can import the file as a DOS text file. I recommend not saving the file from WordPerfect. The data file will be damaged if WordPerfect splits any of the data lines. A data line in the FeedBack file begins with a three digit identification number (ID) ranging from "001" to "999". The FeedBack program reads the data file one line at a time. If the first three digits of the line are an ID number, the program assumes the line is a data line. See Appendix D for the format of the data line.

After selecting a data file, it must be ranked to allow FeedBack to compute the percentiles needed to produce the printout. The "Rank" command also locate any values that are larger than 5.0 or smaller than 1.0 in the data file. If an error is found, the user will be shown the offending data and given its row and column position in the data file. An ASCII text editor can be used to correct any errors. After the errors are corrected, use FeedBack to rank the file again before printing.

In an ID data line, the values are a three digit ASCII number. To save space the decimal point is left out so 2.35 will appear as "235". Feedback rounds all of its input to two digits. "235" would become "2.4" and "234" would become "2.3". Feedback does all its error checking on the rounded numbers not the original data values. Therefore a 5.01 is out of bounds but the rounded value is not, so the value would be accepted. A 5.05 would round to 5.1 and would be an error.

If the rank command completes its task without any errors, the ranking information is added to the end of the data file. By appending the ranking data to the file, it allows the ranking command to be skipped the next time the data file is used. The ranking information will appear at the end of the data file as one long line of numbers. The ranking data line will always begin with an ID number of "000" which is a reserved ID number and must not be assigned to a participant. The format for the ranking data line is listed in Appendix F.

When viewing the data file with a text editor it is advisable to remove the rank data line. Many text editors will split the rank data into two or more lines. The FeedBack program expects the data in one long line. The easiest way around this potential problem is to erase the rank data line and use the FeedBack program to rank the file again.

It is possible to rank a file more than one time without removing the old "000" data line. The last ranking will be the one used to print the graphs. The ranking information
will not be saved in the data file if an error is detected. If a FeedBack data file has more than one "000" data line, I suggest removing all the "000" lines and asking FeedBack to rank the file.

Programming for future expansion was a high priority in the design of the FeedBack program. Because the SLDI is under development it will continue to change and the FeedBack program will need to be updated. The names of the current factors can be changed using an ASCII text editor. Increasing the number of factors requires making small changes in many sections of the source code. It will take approximate three days to update the source code to increase the number of factors.

I believe the SLDI has great potential. The survey questions must be improved to produce a wider range of statistically stable factors. The FeedBack program has the potential to become an expert system producing graphs and individualized analysis.
APPENDIX A:

Sample Feedback Graphs
| DIMENSIONS: | STRATEGIC LEADER DEVELOPMENT INVENTORY |  
| ID Number: 101 | Success Factors | Scored: 08/25/92 |

| DIMENSIONS: | Raw Scores: | ←Below Average | Better Than Most | The Best → | |
| CONCEPTUAL EFFECTIVENESS | Self | Peers | Superiors | Subordinates | |
| TEAM BUILDING | Raw Scores: | Self | Peers | Superiors | Subordinates | |
| STRATEGIC POTENTIAL | Raw Scores: | Self | Peers | Superiors | Subordinates | |
| PERFORMANCE UNDER STRESS | Raw Scores: | Self | Peers | Superiors | Subordinates |
STRATEGIC LEADER DEVELOPMENT INVENTORY

ID Number: 101

DIMENSIONS:

LIMITED PERSPECTIVE
Raw Scores: 1 Never 2 Occasionally 3 Always

Self Peers Superiors Subordinates

EGOCENTRIC
Raw Scores: 1 Never 2 Occasionally 3 Always

Self Peers Superiors Subordinates

CAREERIST
Raw Scores: 1 Never 2 Occasionally 3 Always

Self Peers Superiors Subordinates

UNPROFESSIONAL
Raw Scores: 1 Never 2 Occasionally 3 Always

Self Peers Superiors Subordinates

★ = Score | 1 = 25% | 2 = 50% | 3 = 75% | ★ and ★★ = High

Scored: 08/24/92
APPENDIX B:

Description of SLDI
STRATEGIC LEADER DEVELOPMENT INVENTORY (SLDI)

GENERAL: There are three levels of leadership: Direct, Senior, and Strategic. Direct leaders command units -- battalions, squadrons, ships, and, in some instances, branches or directorates. Senior leaders operate at a level higher. They command organizations and face problems much broader in scope and complexity. Their leadership becomes indirect. You get things done by working through a large number of "others". You can't personally influence everything that needs to be done. Strategic leaders command Joint/Combined operations. They are even more removed from the direct action. These are three and four star billets and civilian equivalents.

Your selection for attendance at a Senior Service College (SSC) signifies you have been successful Direct leaders. The SSC aids you in making a critical transition -- from the Direct to the Senior leadership level. It also exposes you to guest speakers and other experiences so that you can see what the requirements of Strategic leaders are.

A specific rationale is behind the development of this instrument and two theories support the rationale. The specific rationale and two theories will be briefly explained. Then, each factor assessed with the SLDI is described.

SPECIFIC RATIONALE FOR THE SLDI: The SLDI is an assessment tool. You learn more about yourself with the data it provides. The logic behind the SLDI is very simple. You must accurately know yourself before you can accurately assess and understand the strengths and weaknesses of others. You cannot be a good leader without adequate self-knowledge.

You must know yourself for another important reason. You cannot develop yourself without this information. This will probably be your final school assignment. Development beyond this assignment will be your responsibility.

The SLDI taps multiple frames of reference: Yours and those of your peers, subordinates, and superiors. You can "triangulate" from these multiple perspectives to form a more rounded and accurate understanding of your strengths and weaknesses.

THEORIES UNDERLYING THE SLDI: Elliot Jaques' and Robert Kegan's theories underlie the SLDI's factors we'll define later.

Jaques' theory states individuals vary in ability to deal with abstraction and complexity in thought processing. Direct, senior, and strategic leaders must use successively more complex thought processing because the conceptual demands of the positions they occupy become progressively more difficult. The problems to be solved become more and more unstructured or non-routine.
Senior and strategic leaders must develop a vision of desired future states, develop plans to achieve them, and proactively manage the process of getting there. How far you can project plans ahead for your work is an indicator of the complexity you bring to the job. Table 1 summarizes the variables being described here. It shows the timespans -- "planning horizons" -- associated by organizational level using the Army as an example. Business organization equivalents are also shown.

<table>
<thead>
<tr>
<th>TIMESPAN</th>
<th>WORK LEVEL</th>
<th>MILITARY</th>
<th>BUSINESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 YRS</td>
<td>VII</td>
<td>ARMY (General)</td>
<td>CORPORATION</td>
</tr>
<tr>
<td>20 YRS</td>
<td>VI</td>
<td>CORPS (Lt. General)</td>
<td>GROUP</td>
</tr>
<tr>
<td>10 YRS</td>
<td>V</td>
<td>DIVISION (Maj. General)</td>
<td>STRATEGIC UNIT</td>
</tr>
<tr>
<td>5 YRS</td>
<td>IV</td>
<td>BRIGADE (Brig General/ Colonel)</td>
<td>GENERAL MANAGEMENT</td>
</tr>
<tr>
<td>2 YRS</td>
<td>III</td>
<td>BATTALION (Lt. Colonel)</td>
<td>OPERATING UNIT</td>
</tr>
<tr>
<td>1 YR</td>
<td>II</td>
<td>COMPANY (Captain)</td>
<td>SECTION</td>
</tr>
<tr>
<td>3 MTHS</td>
<td>I</td>
<td>PLATOONS SQUADS (NCOs)</td>
<td>SUPERVISORS, OPS CLERKS</td>
</tr>
</tbody>
</table>

Table 1. LEVELS OF WORK

Research suggests the transition from Work Level III to IV is critical and difficult to make. You have been conditioned in your career to be good at leading directly. It is challenging to learn indirect leadership skills. You will be facing problems too complex for you to solve on your own. The SLDI is intended to aid the transition process. To make the other major transition, from work level V to VI and VII, you will most likely be on your own.

Kegan's theory is about emotional maturing. It specifies where one's self-definition comes from. Kegan believes there are six stages of maturity (stage 0 thru V). We won't describe them all, only the two we have found apply to SSC students and more senior officers. These are stages III and IV. Based on
empirical research done at ICAF and the Army War College, we can predict that most of you are in transition between these two stages and some have reached the more advanced stage (IV).

If you are in stage III, you are not fully capable of seeing yourself as you really are. What you think are "good" and "bad" person characteristics and qualities are defined by your organization and by the society you live in. People in stage III are sometimes called "organization men or women." They respect their contemporaries, appreciate mutual and reciprocal relationships, can become a part and product of "group think" (they will not usually take on positions contrary to the norm for fear of rejection and ridicule) and otherwise define themselves as their context defines them. True self-perception, what they are actually like, is lost in an external "ideal". They are not yet emotionally ready to develop a definition and an identity of themselves apart from their primary reference group.

For you to reach stage IV, being fully capable of self-definition, you must accurately understand yourself as you are. Acceptance and accountability are vital keys. You can accomplish this through a process Kegan, Lewis, Kuhnert and Maginnis refer to as "De-Centering." This means stepping "outside" yourself to see yourself as others see you. Their perceptions may be different than yours, but still valid. You must be able to accept without defensiveness the differing perceptions of others and synthesize them to form a more objective, holistic "picture" of yourself. The SLDI has been designed to aid in developing this more objective, holistic "picture".

Research suggests that cognitive and emotional development are not independent. They are inter-related. SSC students need to advance to the fourth level of cognitive and ego development to be effective problem solvers at the senior leadership level. Otherwise, they cannot "see" problems objectively (and, thus, define them correctly) or have the thinking capacity to deal with them effectively.

SLDI FACTORS.

Through an analysis process we identified four "SUCCESS" and "FAILURE" factors from the questions you and your superiors, peers, and subordinates answered. SUCCESS factors are those things that, if developed well, can lead to successful performance in successively higher positions. FAILURE factors are practices that could lead to what has been called "De-Railment", failure to achieve the potential one has. The four SUCCESS factors are called:

0 Conceptual Effectiveness.

 Broad Perspective -- understands the
perspectives of superiors, how the mission of own unit meshes with that of others, values long range planning, and thinks strategically.

Conceptual Grasp -- understands complex situations, is comfortable with paradox and contradiction, and can pinpoint cause and effect relationships in complex situations.

Time Horizon -- develops long-term objectives and anticipates resources for achieving them, has a good sense of how future world events may affect the military and works to insure own initiatives are carried out by successors.

Analytic Clarity -- can work from the abstract to the concrete. This means developing a concept and then making it a reality - gets to the heart of the matter quickly and sorts out what is important from what isn't.

Conceptual Flexibility -- willing to adjust quickly when obstacles are encountered, has an understanding that guidelines are not fixed or rigid, remains focused when the unexpected occurs, and changes courses of action when new information emerges.

Conceptual Complexity -- views all sides of a problem and alternative ways of solving them, manages more than one project at a time, considers many contingencies when making operational plans, envisions multiple courses of action when considering various scenarios, and integrates own plans with those of other units.

Personal Objectivity -- has a good grasp of personal strengths and weaknesses, maintains own objectivity when others get caught up in the heat of the moment and has a coherent rationale for own actions.

O Team Building.

Judgement/Character -- able to judge quality in others, provides wise counsel to others, maintains a balance between work and personal life, has a good, non-hostile sense
of humor, shows confidence and humility, puts mission before career, and sets high but realistic standards.

Communication Effectiveness -- keeps subordinates informed and encourages them to express disagreement. You are able to understand subordinates' points of view and their problems. You are willing to tell subordinates things they might not want to hear about themselves and help them to understand the bigger picture, maintain a sincere interest in what others have to say, are approachable, and listen well.

Team Orientation -- actively works to build effective teams without losing sight of individuals. Empower others to accomplish the mission and recognizes good performers from those that only "look" good. Works hard for subordinates, backs them when appropriate, delegates authority and responsibility, and shows interest in their professional development.

Creates Good Work Climate -- does not play favorites, resolves conflict among subordinates and gains their trust and support. Creates a supportive work context, treats subordinates fairly, helps them learn from mistakes and able to get them to be effective without the use of rank or position.

Drive/Energy Level -- has high energy level and enthusiasm, a strong work ethic, hangs in there when things get tough, engenders enthusiasm in subordinates.

**Strategic Potential.**

Manages Self-Development -- seeks to become knowledgeable in areas outside current job responsibilities; works to correct own weaknesses; manages own career direction; solicits feedback to grow professionally; optimistic about the future.

Shows Cultural/Political Sensitivity -- persuades others to support desired actions, knows who to talk to to get things done, shows judgement in politically sensitive
matters, recognizes the unique concerns of minorities and women in and outside of the military culture, accepts the fact that politics are a key part of organizational life, recognizes the potential impact of the external political environment on own plans and programs and accepts community standards as legitimate constraints on personal behavior.

0 **Performance Under Stress.** Seizes opportunities when they arise, takes calculated risks, takes charge in crisis situations, works well under pressure, and dependable in key situations.

The four **FAILURE** factors are:

0 **Limited Perspective.** This factor is related to the first positive factor, but it is not necessarily "the other side of the coin". It is defined by six dimensions which are described below. Lower scores are preferred.

- Technical/Tactical Incompetence -- fails to achieve technical competence in new areas, fails to get the facts straight, shows lapses of common sense, ignores important details, judged by others as being generally technically ineffective.

- Insulated -- inaccessible to subordinates, generally is unapproachable, favors management by memorandum rather than through face-to-face communication, works within a very limited "inner circle," is secretive -- unwilling to share thoughts with others.

- Indecisive -- shrinks from making hard decisions. Easily influenced by what others think -- particularly by higher ranking officials, reluctant to make a decision without achieving a consensus, fails to stay focused on primary issues. In short, likes to "play it safe."

- Narrow Perspective -- cannot develop a long-term vision, tied to standard ways of doing things, parochial -- would have a hard time adapting to a multi-agency or joint environment, has difficulty being political when necessary, tends to get bogged down in details.
Lacks Conceptual Grasp -- crisis oriented -- always putting out "brush fires", reactive rather than proactive, prefers to work on one project at a time and be linear rather than multi-linear and integrative.

Dependent on Clear Structure -- needs extensive guidance to get things done, displays generally an intolerance of uncertainty, looks for the perfect solution to problems.

0 Egocentric.

Self-Centered -- likes to draw attention, is arrogant and thinks the rules apply only to others, holds to own position even in the face of contradicting information, takes special privileges, impressed with own rank and status.

Defensive -- has difficulty recognizing own limitations, will not admit to not having all the answers, suspicious of others, reacts negatively to dissenting opinions.

Disregards Others -- criticizes subordinates in front of others and generally "talks down" to them, tends to take credit for others' work, and berates others even for honest mistakes.

Temperamental -- acts impulsively and easily loses control, loses temper easily, jumps to conclusions, makes snap judgments about people.

Micromanages -- gets bogged down in details, nit picks about what others have done, insists on precision in trivial matters.

Inflexible -- wants everything done own way, generally autocratic in dealing with subordinates.

Untrusting -- has hidden agendas, fails to meet established deadlines, gossips and complains about others behind their back, is vindictive, tolerates back stabbing, encourages destructive competition among subordinates.
0 Careerist. Looks out for self more than for others, puts own career and interests ahead of the goals of the organization and promoting professional development of others. Will not "rock the boat" when needed. Lets others take the heat for failures. Willing to abuse subordinates to further career.

0 Unprofessional. Behaves with questionable ethics, uses foul language excessively, fails to maintain physical fitness, and "falls on sword" over unimportant issues.

INTERPRETING THE SLDI SUMMARY DATA SHEETS.

Two Data Sheets summarizing your scores on the SLDI accompany this hand-out. One covers the SUCCESS factors and the second one covers the FAILURE factors.

The average score for self, peers, superiors, and subordinates is shown for each SUCCESS and FAILURE factor. The "diamond" shows the average of items for you, and your peers, superiors, and subordinates on each factor. These averages are specific to the particular individuals that provided assessments for you. Pay particular attention to these averages -- they are a "benchmark" specific to YOU. Pay attention to the degree of each rating (whether it is "high or low" to identify strengths and weaknesses) and to the discrepancies between your assessments of self and the assessments of each of the other reference groups. Large discrepancies mean that others are not perceiving you as you perceive yourself. This can be a problem for reasons already stated. On the other hand, you may have intentionally caused the discrepancies because you made a conscious decision to portray yourself differently to each of the reference groups. You may be the only one who can assess what the reasons for any large discrepancies between how you rated yourself and others rated you.

The light and dark shaded bars are used to display normative information. These bars show the range of scores across all respondents of each type. Also coded within each bar are the 25th, 50th, and 75th percentiles. This allows you to see where the scores you provided about yourself, by your peers, superiors, and subordinates are located with respect to all such ratings provided for the entire class of '93. A legend defining the percentiles is at the bottom of each summary sheet. Anchors for the factor rating scale are shown at the top of each summary. Note that for the SUCCESS factors, higher scores are better. For the FAILURE factors, lower score are better.
APPENDIX C:

Converting the POS File
Strategic Leader Development Inventory

For the ICAF data printed in August of 92 the SPSS program created a POS data file with the format listed below. This file was edited before it can be used by the Feedback program. I used an ASCII text editor to move and delete some columns. See Appendix D for the correct format of the POS and NEG data files.

Original New
Columns Columns
1-3 1-3 ID number range: '001' to '499'
raw scores format: 2.03 will be "203"

4-6 delete Self on Conceptual Effectiveness (Self vs Peers)
7-9 delete Self on Team Building (Self vs Peers)
10-12 10-12 Self on Strategic Potential (Self vs Peers)
13-15 delete Self on Performance Under Stress (Self vs Peers)
16-18 4-6 Self on Conceptual Effectiveness (Self vs Sup.)
19-21 delete Self on Team Building (Self vs Sup.)
22-24 delete Self on Strategic Potential (Self vs Sup.)
25-27 delete Self on Performance Under Stress (Self vs Sup.)
28-30 delete Self on Conceptual Effectiveness (Self vs Sub.)
31-33 7-9 Self on Team Building (Self vs Sub.)
34-36 delete Self on Strategic Potential (Self vs Sub.)
37-39 13-15 Self on Performance Under Stress (Self vs Sub.)
40-42 16-18 Peer mean on Conceptual Effectiveness
43-45 19-21 Peer mean on Team Building
46-48 22-24 Peer mean on Strategic Potential
49-51 25-27 Peer mean on Performance Under Stress
52-54 28-30 Superiors mean on Conceptual Effectiveness
55-57 31-33 Superiors mean on Team Building
58-60 34-36 Superiors mean on Strategic Potential
61-63 37-39 Superiors mean on Performance Under Stress
64-66 40-42 Subordinates mean on Conceptual Effectiveness
67-69 43-45 Subordinates mean on Team Building
70-72 46-48 Subordinates mean on Strategic Potential
73-75 49-51 Subordinates mean on Performance Under Stress
APPENDIX D:

Format for FeedBack Data Files
Strategic Leader Development Inventory

Line format of input ASCII data files:

Columns format for a each data line of the .POS data file:

Columns
1-3   ID number range: '001' to '999'
      raw scores format: 2.03 will be "203"
4-6   Self on Conceptual Effectiveness (Self vs Superiors)
7-9   Self on Team Building (Self vs Peers)
10-12  Self on Strategic Potential (Self vs Subordinates)
13-15 Self on Performance Under Stress (Self vs Subordinates)
16-18 Peer mean on Conceptual Effectiveness
19-21 Peer mean on Team Building
22-24 Peer mean on Strategic Potential
25-27 Peer mean on Performance Under Stress
28-30 Superiors mean on Conceptual Effectiveness
31-33 Superiors mean on Team Building
34-36 Superiors mean on Strategic Potential
37-39 Superiors mean on Performance Under Stress
40-42 Subordinates mean on Conceptual Effectiveness
43-45 Subordinates mean on Team Building
46-48 Subordinates mean on Strategic Potential
49-51 Subordinates mean on Performance Under Stress

DOS File name: *.NEG = Negative Factors data file
Columns format for a each data line of the .NEG data file:

Columns
1-3   ID number range: '001' to '999'
      raw scores format: 2.03 will be "203"
4-6   Self on Limited Perspective
7-9   Self on Egocentric
10-12 Self on Careerist
13-15 Self on Unprofessional
16-18 Peer mean on Limited Perspective
19-21 Peer mean on Egocentric
22-24 Peer mean on Careerist
25-27 Peer mean on Unprofessional
28-30 Superiors mean on Limited Perspective
31-33 Superiors mean on Egocentric
34-36 Superiors mean on Careerist
37-39 Superiors mean on Unprofessional
40-42 Subordinates mean on Limited Perspective
43-45 Subordinates mean on Egocentric
46-48 Subordinates mean on Careerist
49-51 Subordinates mean on Unprofessional
APPENDIX E:

Locating WordPerfect’s Printer Port
To use WordPerfect to discover the printer port assignment use following steps:

1. Start the WP program.
   If you make a mistake in steps 2 - 5 use the F1 key to cancel the command.

2. Use the Shift-F7 command to bring up the print menu.

3. Use the "S" command to display the Select Printer screen.

4. Use the "Edit" command listed at the bottom of the screen.

5. The edit screen lists the current port assignment on the upper half of the screen. It should say port: LPT1, LPT2, or LPT3.
   If it says COM1 or COM2 FeedBack cannot be used on this computer unless the DOS 'MODE' command has been used to reassign the LPT port to the listed COM port.

6. To return to the main screen without making any changes to WordPerfect, press the F1 key until the word processing screen appears.

7. Use the F7 command to exit WordPerfect.
APPENDIX F:

Format for FeedBack Variable Strings
### Strategic Leader Development Inventory

Offsets in data files, [FileBuf] and [PerCnt] memory strings

<table>
<thead>
<tr>
<th>Data File Columns:</th>
<th>[FileBuf]</th>
<th>[PerCnt]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3 ascii bytes)</td>
<td>(3 ascii bytes)</td>
<td>Low</td>
</tr>
<tr>
<td>1-3</td>
<td>ID number</td>
<td>0 - 2</td>
</tr>
<tr>
<td></td>
<td>(ascii words)</td>
<td></td>
</tr>
</tbody>
</table>

#### Self Data Locations

| 4-6               | Self Dim1 | 3 - 5 | 5 | 7 | 9 | 11 | 13 |
| 7-9               | Self Dim2 | 6 - 8 | 15 | 17 | 19 | 21 | 23 |
| 10-12             | Self Dim3 | 9 - 11 | 25 | 27 | 29 | 31 | 33 |
| 13-15             | Self Dim4 | 12 - 14 | 35 | 37 | 39 | 41 | 43 |

#### Peers Data Locations

| 16-18             | Peer Dim1 | 15 - 17 | 45 | 47 | 49 | 51 | 53 |
| 19-21             | Peer Dim2 | 18 - 20 | 55 | 57 | 59 | 61 | 63 |
| 22-24             | Peer Dim3 | 21 - 23 | 65 | 67 | 69 | 71 | 73 |
| 25-27             | Peer Dim4 | 24 - 26 | 75 | 77 | 79 | 81 | 83 |

#### Superiors Data Locations

| 28-30             | Supr Dim1 | 27 - 29 | 85 | 87 | 89 | 91 | 93 |
| 31-33             | Supr Dim2 | 30 - 32 | 95 | 97 | 99 | 101 | 103 |
| 34-36             | Supr Dim3 | 33 - 35 | 105 | 107 | 109 | 111 | 113 |
| 37-39             | Supr Dim4 | 36 - 38 | 115 | 117 | 119 | 121 | 123 |

#### Subordinates Data Locations

| 40-42             | Subd Dim1 | 39 - 41 | 125 | 127 | 129 | 131 | 133 |
| 43-45             | Subd Dim2 | 42 - 44 | 135 | 137 | 139 | 141 | 143 |
| 46-48             | Subd Dim3 | 45 - 47 | 145 | 147 | 149 | 151 | 153 |
| 49-51             | Subd Dim4 | 48 - 50 | 155 | 157 | 159 | 161 | 163 |

F-2
APPENDIX G:

Source Code for the FeedBack Program
F:FeedBack.ASM  Summer 1992  James E. Hopkins

A program to print Strategic Leader Development Inventory's
self feedback reports on a HP Laser Jet printer.

MODEL small

STACKSIZE EQU 2024

STACK STACKSIZE

INCLUDE FBD.ASM ; data for printer proc
INCLUDE FBR.ASM ; ranking procedures
INCLUDE FBA.ASM ; print procedures
INCLUDE FB.B.ASM ; printer subroutines
INCLUDE FBM.ASM ; menu procedures
INCLUDE FBT.ASM ; title procedures
INCLUDE FBJ.ASM ; universal procedures
INCLUDE FBS.ASM ; select file procedures
INCLUDE FBF.ASM ; file procedures
INCLUDE FBE.ASM ; edit path procedures
INCLUDE FBN.ASM ; input ID number proc

DATA ; the data segment.
ErrCode db 0 ; ret error msg to DOS
; note: If Debug is ON the printing time will be twice as long.
Debug db 0 ; 0 = OFF  Other = ON

; video data
Vidmode db 0 ; video mode
vidpage db 0 ; video page
vidcurs dw 0 ; cursor type
vidfont dw 0 ; font size
vidattr db 07h ; default Lt White/Black
vidbord db 07h ; border color

; Color variables
Color db 07h ; active color
System db 07h ; default Lt White/Black
Menu db 0 ; Menu main color
Normal db 07h ; Main display screen
Hilite db 0 ; display screen titles
MenuMes db 0 ; menu messages line
Warning db 0 ; accent color
Border db 0 ; display screen box

; Memory Block variables
VarSeg dw 0 ; seg of var mem block
DirSeg dw 0 ; seg of dir mem block
MaxFile db 0 ; number of files 0-250
MaxDim db 0 ; number of dimensions
BarPos dw 0101h ; position of hilite bar

G-2
Data file variables

PosTyp db 'POS'.O ;Positive dim data type
NegTyp db 'NEG'.O ;Negative dim data type
FiTyp db '.SLD'.O ;file type
FileNa db '?????????.SLD'.O ;ASCII file name
SearchNa db '?????????.SLD'.O ;ASCII file name
FileHd dw 0 ;file handle
FileDr db 0 ;0 = default, 1 = A etc
DataEr db 0 ;0 = FALSE Other = TRUE
EOF db 0 ;0 = FALSE Other = TRUE
Report db 0 ;0 = Pos. Other = Neg.
Ranked db 0 ;0 = NO Other = YES
MaxNo dw 0 ;number of ID's in file

;buffer used for data storage:
Flibuf db 192 DUP (0h),0h ;file data input

;Lowest, Highest, Median, 25th and 75th percentiles for 4 groups
;and 4 dimensions = 160 bytes
:Data format: 0Dh,0Ah,1.D,#.lowest highest, median, 25% 75% etc stored in ASCII
PerCnt db 192 DUP (0h),0h ;

;Printer port (the program expects an HP Laser Jet assigned to a parallel port)
LPT dw 0 ;default = LPT1
;0 = LPT1, 1 = LPT2, & 3 = LPT3

:Path Editor variables
Path db 82 DUP (0h) ;input ASCII string.
Input db 82 DUP (0h)
Search db 82 DUP (0h)
Digit db 1 ;0 = OFF Other = ON
Insert db 0 ;0 = OFF Other = ON
EndFd db 0 ;0 = OFF Other = ON

;Sound string
Beep dw 6000,2,4500,2,0

.CODE ;the code segment

MAIN:

;---Determine Color and Graphics Mode
MOV AX,@data ;get data segment
MOV DS,AX ;put in data segment reg
CALL COLOR MODE ;define default colors
CALL TEXT VIDEO ;save default settings

;---Main procedure for Feedback
CALL INTERRUPT_HANDLER ;INT23 & INT24 handlers
CALL RELEASE_MEM ;release unused memory
JC Error ;display Dos error
CALL MAIN_MENU ;Program's MAIN LOOP
JC Error ;display Dos error
CALL CLOSE_FILE ;close SLDI file if open
JC Error ;display DOS error.

;---Exit to DOS ;program always ends here
Exit: CALL RESTORE_VIDEO ;restore users settings
MOV AL,[ErrCode] ;load error level number
MOV AH,4Ch ;Exit function number
INT 21h ;return to DOS

;---End of Main procedure for FeedBack
Error: CALL DISPLAY_ERROR ;show DOS extend error
JMP SHORT EAi ;---End of the source code
END MAIN

.DATA

;--- HP PCL strings used to position a point on the graph
NextNo db 32,1Bh,'&k2S',1Bh,'&a+19C',1Bh,'&kOS',0 ;space + 19 comp. spaces
NextUn db 1Bh,'&k2S',1Bh,'&a+2C',1Bh,'&kOS',0 ;2 compressed spaces
HalfSp db 1Bh,'&k2S',1Bh,'&a+1C',1Bh,'&kOS',0 ;1 compressed spaces
BackSp db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS',0 ;1 compressed spaces
FullBk db 1Bh,'&k2S',1Bh,'&a-2C',1Bh,'&kOS',0 ;1 compressed spaces

;--- HP PCL strings used by printing procedures
Heder db 1Bh,'(s3BSTRATEGIC LEADER DEVELOPMENT INVENTORY',0
IDstr db 'ID Number: ' ;3 digit ASCII number
ID db '123',0
PosT db 1Bh,'&a+3C',1Bh,'&fOSSuccess Factors',1Bh,'&f1S',1Bh,'&a+2R'
db 1Bh,'&a+3C',1Bh,'(sOB' ;post row/col & unbold
db ' ',1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS"Q'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS-'
db 'Below Average Better Than Most The Best ' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS-' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS"P'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS-' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS"P'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
PosT db 1Bh,'&a+3C',1Bh,'&fOSFailure Factors',1Bh,'&f1S',1Bh,'&a+2R'
db 1Bh,'&a+3C',1Bh,'(sOB' ;post row/col & unbold
db ' ',1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS"Q'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS-' ;
db 'Never Occasionally Always '
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS-' ;
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS"P'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&kOS' ;
PosT db 1Bh,'&a+3B','0 ;bold and EndOfString
DTstr db 'Scored: '
Date db '07/24/92',0 ;file date
Distr db 'DIMENSIONS:',1Bh,'(sOB',0 ;unBold, EndStMarker

G-4
FFeed db 0Ch,0       ;formfeed string
Point db 4,0       ;4 = "D"
Median db 186,0   ;179 = "|"
Left db 179,0     ;179 = "|"
Right db 179,0    ;179 = "|"
UntLt db 177,0    ;177 =  
UntRk db 178,0    ;178 =  
TenLT db 13 DUP (177),0
TenRk db 13 DUP (178),0

;---An HP PCL string used by Initialize_HP procedure
Init db 1Bh,'E'    ;reset printer
db 1Bh,'&10'      ;landscape
db 1Bh,'&k0S'     ;10.0 cpi
db 1Bh,'10U'      ;PC-8 symbol set
db 1Bh,'a0P',0    ;Fixed spacing

;---An HP PCL string used by Restore_HP procedure
Rest db 1Bh,'&10O' ;portrait
db 1Bh,'8U'       ;Roman-8 symbol set
db 1Bh,'a1P'      ;Proportional spacing
db 1Bh,'E',0      ;reset printer

;---An HP PCL string used by HPGOTOXY procedure
GoTo dw 0         ;ASC II column number
db 'C',1Bh,'&a'    ;set hp laser to
db 'R',0          ;ASC II row number
db 'R',0          ;end of string marker

;---An HP PCL string to draw a graphing box and the present cursor position
Box  db 1Bh,'&OS Raw Scores: 1' ;starting Push
ticks and "2"
db 1Bh,'&k2S --------',1Bh,'&k0S' ;compressed mode
db 1Bh,'&OS',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&OS',1Bh,'&a+6R',193,1Bh,'&f1S2' ;bottom tick and "2"
ticks and "3"
db 1Bh,'&k2S --------',1Bh,'&k0S' ;compressed mode
db 1Bh,'&OS',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&OS',1Bh,'&a+6R',193,1Bh,'&f1S3' ;bottom tick and "3"
ticks and "4"
db 1Bh,'&k2S --------',1Bh,'&k0S' ;compressed mode
db 1Bh,'&OS',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&OS',1Bh,'&a+6R',193,1Bh,'&f1S4' ;bottom tick and "4"
ticks and "5"
up db 1Bh,'&k2S --------',1Bh,'&k0S' ;compress and "5"
db 1Bh,'&f1S',0Ah ;ending Pop + line feed = next line
top line
db 1Bh,'&OS',218,13 DUP (196),194,49 DUP (196),191
db 1Bh,'&f1S',0Ah ;next line
self line
db 1Bh,'&OS',179, 'Self ,179,1Bh,'&a+49C',179
db 1Bh,'&f1S',0Ah ;next line
peers line
db 1Bh,'&OS',179, 'Peers ',179,1Bh,'&a+49C',179
db 1Bh,'&f1S',0Ah ;next line

G-5
; superiors line
    db  1Bh,'&MOS',179,' Superior',179,1Bh, '&a+49C',179
    db  1Bh,'&f1S',0Ah ; next line
; subordinates line
    db  1Bh,'&MOS',179,' Subordinates',179,1Bh, '&a+49C',179
    db  1Bh,'&f1S',0Ah ; next line
; bottom line
    db  192,13 DUP (185),193, 4B DUP (196),217,0 ; EndOfString marker
    ; An HP PCL string to draw a graphing box and the present cursor position
    inform db  1Bh,'&MOS',218,63 DUP (196),191
        db  1Bh,'&f1S',0Ah ; next line
    ; self line
    db  1Bh,'&MOS',179
        db  ' °D = Score || = 25% || = 50% || = 75% || = Range ' ; next line
    db  179,1Bh,'&f1S',0Ah
; bottom line
    db  192,63 DUP (196),217,0 ; EndOfString marker
    ; An HP PCL string to label Positive Dimension #1
    Pos1 db  1Bh,'(s3B' ; bold ON
        db  'CONCEPTUAL EFFECTIVENESS'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Positive Dimension #2
    Pos2 db  1Bh,'(s3B' ; bold ON
        db  'TEAM BUILDING'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Positive Dimension #3
    Pos3 db  1Bh,'(s3B' ; bold ON
        db  'STRATEGIC POTENTIAL'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Positive Dimension #4
    Pos4 db  1Bh,'(s3B' ; bold ON
        db  'PERFORMANCE UNDER STRESS'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Derailment Dimension #1
    Neg1 db  1Bh,'(s3B' ; bold ON
        db  'LIMITED PERSPECTIVE'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Derailment Dimension #2
    Neg2 db  1Bh,'(s3B' ; bold ON
        db  'EGOCENTRIC'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString
    ; An HP PCL string to label Derailment Dimension #3
    Neg3 db  1Bh,'(s3B' ; bold ON
        db  'CAREERIST'
        db  1Bh,'(s0B' ; bold OFF
        db  0 ; current EndOfString

G-6
---An HP PCL string to label Derulment Dimension #4
Neg4  db 1Bh,'(s38'
     ;bold ON
db 'UNPROFESSIONAL'
db 1Bh,'(s08'
     ;bold OFF
db 0      ;current EndOfString
.CODE
Rank the data to compute lowest, 25th, 50th, 75th percentiles, and last
Input = None
Output = If completed [Ranked] <> 0 or TRUE
        If [DataErr] = 0 a '000' data line to appended to the file
        [PerCnt + 5] points to percentile variables

PROC RANK_DATA
    PUSH AX            ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    CALL IS_RANK       ;'is file open, unranked
    JC     RD5          ;and [MaxNo] > 0 ?
    CALL GET_VAR_BLK   ;create var mem block
    JC     RD5          ;exit on DOS error
    XOR AX, AX         ;zero to ax
    MOV [DataErr], AL   ;set data error = False
    CALL RANK_WAIT_MESS ;estimate how much time
    CALL RANK_INSTRU    ;bottom message box
    MOV CX, 16          ;number of var to rank
    RD0: CALL READ_VAR  ;one var for all ID's
    CMP AX, 3           ;were 3 or more found?
    JNC    RD1           ;Yes continue else
    CALL ID_ERR         ;display error message
    JMP SHORT RD2       ;loop to next variable
    RD1: CALL PROGRESS_MESS ;tell user of progress
    CALL VAR_SORT       ;sort in DOS mem block
    CALL CHECK_DATA     ;is data in bounds
    JC     RD4           ;carry flag = abort
    CALL STORE_VAR      ;get median,25% & 75%
    ;---Check keyboard buffer to see if the <Esc> key been pressed?
    RD2: MOV AX, 0600h   ;DOS function # 6
    MOV DL, 0FFh         ;board buffer.
    INT 21h              ;NO key pressed continue
    JZ     RD3            ;was it the <ESC> key?
    CMP AL, 1Bh          ;if NO continue
    JNZ    RD3            ;If YES inform user
    CALL ESC_YN          ;carry flag = abort
    JC     RD4            ;loop until each column is ranked.
    RD3: LOOP RD0        ;loop until cx = 0
    MOV AL, OFFh         ;[ranked]< >0 = TRUE
    MOV [Ranked], AL     ;mark file ranked
    ;---If no errors write data line to file
    CMP BYTE PTR [DataErr], 0       ;any errors found?
    JNZ    RD4             ;If Yes exit else

G-7
CALL APPEND_FILE ;save data to file

normal exit point

RD4: CALL RELEASE_VAR_BLK ;release mem var block

RD5: CLC
POP DX ;restore registers
POP CX
POP BX
POP AX
RET

ENDP RANK_DATA

Is the data file open, unranked, and ID > 0
Input = None
Output = Carry flag if Not Ready.

PROC IS_RANK
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX

is file selected?
MOV BX,[FileHd] ;get file handle
CMP BX,0 ;is a file open?
JNZ RK1

CALL FILE_ERR
JMP SHORT RK4

were the percentiles in the data file?
RK1: MOV CL,[Ranked] ;are percentiles set?
CMP CL,0 ;if NO goto next test
JZ RK2
CALL PERCT_ERR ;Rerank file?
JC RK5 ;carry flag = NO

is MaxNo > two?
RK2: MOV AX,[MaxNo]
CMP AX,3
JNC RK3
CALL ID_ERR

RK4: STC ;set error flag
RK5: POP DX ;restore registers
POP CX
POP BX
POP AX
RET

ENDP IS_RANK

Input = none
Output = none

PROC ID_ERR
CALL CLEAR_MESSAGE ;warning color
MOV AL,[Warning]
MOV CL,[Color] ;save original color
MOV [Color],AL ;set color
MOV AX,020Ah ;row 3/Col 8
CALL GOTOYX ;set cursor
CALL CSTR_OUT ;display warning

db 'Can not rank values if less than three scores.'
db 'Press Any Key.',0

MOV [Color],CL ;restore original color
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
RET
ENDP ID_ERR

Input = none
Output = none

PROC PERCT_ERR

CALL CLEAR MESSAGE

MOV AL,[Warning] ;warning color
MOV CL,[Color] ;save original color
MOV [Color],AL ;set color
MOV AX,0208h ;row 3/Col 8
CALL GOTOYX ;set cursor
CALL CSTR_OUT ;display warning

db 'This data file is already ranked.'
db 'Rank it again? Y/[N]',0

MOV [Color],CL ;restore original color
CALL HIDE_CUR
CALL ERR_SOUND

PER1: CALL GET_CHAR

AND AL,5Fh ;turn off bits 6 & 8
CMP AL,'N' ;is it No?
JZ PER3 ;if yes exit
CMP AL,0Dh ;is it <Enter>?
JZ PER3 ;if not continue
CMP AL,'Y' ;is it Yes?
JNZ PER1 ;if not get another

PER2: CLC
JMP SHORT PER4

PER3: STC ;clear carry flag

PER4: RET
ENDP PERCT_ERR

Release the memory variable block.

Input = None
Output = Carry flag if DOS error
[VarSeg] = starting segment address for variable block.

PROC RELEASE VAR BLK

PUSH BX
PUSH CX
PUSH DX
PUSH ES
XOR AX,AX ;zero AX
CMP [VarSeg],AX ;is VarSeg assigned?
JZ REL1

; if not assigned go on

; release assigned memory block
MOV AX,[VarSeg]
MOV ES,AX
MOV AX,4000h
INT 21h
JC REL1

; initialize variable
XOR AX,AX
MOV [VarSeg],AX
CLC

REL1: POP ES
POP DX
POP CX
POP BX
RET

ENDP RELEASE_VAR_BLK

Create a byte array to be used to rank each variable.

Input = [MaxNo] > 0
Output = Carry flag if DOS error
[VarSeg] = Starting segment address of memory block.
[MaxNo] = total number ID's in the file.

PROC GET_VAR_BLK
PUSH BX
PUSH CX
PUSH DX
PUSH ES
CALL RELEASE_VAR_BLK
JNC CRV0
JMP CRV9

CRV0: MOV AX,[MaxNo]

MOV CL,3
SHR AX,CL
MOV BX,AX
INC BX
INC BX
MOV AH,48h
INT 21h
JC CRV2
MOV [VarSeg],AX
JMP SHORT CRV8

CRV2: MOV CL,[Color]
MOV AL,[Warning]
MOV [Color],AL
MOV AX,0207h
CALL GOTOXY
CALL CSTR

db " Not enough memory to rank the variables. ", 0
db " Press Any Key to Continue. ", 0
MOV [Color],CL
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
STC
JMP SHORT CRV9
CRV8: CLC
CRV9: POP ES
POP DX
POP CX
POP BX
RET
ENDP GET_WAR_BLK

Clear Input Buffer.
Input = None
Output = None 192 hex 0 to [FillBuf]

PROC CLEAR FILBUF
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH ES
;Make ES = DS
MOV AX,DS
MOV ES,AX
MOV CX,96
MOV BX,Offset FillBuf
XOR AX,AX
MOV [BX],AX
MOV DI,BX
INC DI
INC DI
MOV SI,BX
CLD
REP MOVSW
CLC
POP ES
POP DX
POP CX
POP BX
POP AX
RET
ENDP CLEAR_FILBUF

Clear Percentilte variables.
Input = None
Output = None 192 hex 0 to [PerCnt]
PROC  CLEAR_PERCNT
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH ES
    ;---fill [FIBuf] with 192 hex 0's
    MOV AX,DS
    MOV ES,AX
    MOV CX,96
    MOV BX,Offset PerCnt
    MOV [BX],AX
    MOV AX,0
    MOV DI,BX
    INC DI
    INC DI
    MOV SI,BX
    CLD
    REP MOVSW
    CLC
    POP ES
    POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP  CLEAR_PERCNT

Read a variable for each ID number into DOS memory block.

Input  =  None
Output =  AX = No of variables to sort
         Carry flag if DOS error
[VarSeg] = starting segment address for variable block.
NOTE: Offset for each variable are computed from the loop counter.
     BX = stores offset ptr in memory blk or No of variables found.
     DX = stores offset in [FIBuf] (loop count x 3)
PROC  READ VAR
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DS
    XOR AX,AX
    MOV BX,AX
    CMP [VarSeg],AX
    JZ RVR9
    ;---compute offset in [FIBuf]
    MOV AX,CX
    MOV AH,AL
    SHL AL,1
    ADD AL,AL
    XOR AH,AH
    MOV AX,DS
    MOV BX,ES
    MOV AX,[BX]
    ADD AX,DX
    MOV CX,96
    MOV BX,Offset PerCnt
    MOV [BX],AX
    MOV AX,0
    MOV DI,BX
    INC DI
    INC DI
    MOV SI,BX
    CLD
    REP MOVSW
    CLC
    POP ES
    POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP  READ_VAR

G-12
MOV CX, Offset Fill Buf
ADD AX, CX
MOV DX, AX
CALL GOTO_TOP
JC RVR9

; set default ID string to ASCII zeros
MOV DI, Offset ID
MOV AX, 3030h
MOV [DI], AX
XOR AH, AH
INC DI
INC DI
MOV [DI], AX

; locate ID number in the data file
RVR1: CALL CLEAR FILBUF
CALL READ_LINE
JNC RVR2
MOV AL, OFFH
MOV [EOF], AL

; is this a '000' data line?
RVR2: MOV CX, 3
MOV DI, Offset ID
MOV SI, Offset Fill Buf
CLD
REPZ CMPSB
JZ RVR4

; is this an ID number line?
MOV SI, Offset Fill Buf
MOV CX, 3
RVR4: MOV AL, [SI]
CMP AL, '0'
JC RVR5
CMP AL, ':'
JNC RVR5
INC SI
LOOP RVR4

; is this the EndOfLine or a <spaces>?
; NOTE: this filter is designed to allow errors into the sorting array.
; The CHECK DATA procedure will report false data values.
MOV SI, DX
MOV AX, 2000h
CMP [SI], AH
JZ RVR5
CMP [SI], AL
JZ RVR5
CALL VAR TO.BLK
INC BX
INC BX

; is this the last line?
RVR5: XOR AL, AL
CMP AL, [EOF]
JZ RVR1
; Return no of variables
SHR AX,1
CLC
POP DS
POP DX
POP CX
POP BX
RET

ENDP READ VAR

; Move twobyte ASCII number in data file to memory [VarSeg]
; Input = BX = Offset in [VarSeg]
; DX = Offset in [FileBuf]
; Output = None
PROC VAR TO BLK
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH ES
MOV AX,[VarSeg] ;ptr to base of membik
MOV ES,AX ;ES set to memory blk
MOV SI,DX ;source pointer
MOV DI,BX ;destination pointer
INC DI ;skip first word
INC DI
MOV AL,'S' ;check for rounding
CMP [SI + 2],AL ;round the number?
JC VTB1
;---round the number
INC SI ;point to unit byte
INC BYTE PTR [SI] ;advance units digit
CMP BYTE PTR [SI], ':' ;is it a '9' + 1 ?
JNZ VTB0 ;OK! continue
;---If over flow adjust both digits
MOV AL,'0' ;ASCII 0 to AL
MOV [SI],AL ;replace with zero
DEC SI ;ptr tens digit
INC BYTE PTR [SI] ;advance tens digit
JMP SHORT VTB1 ;move rounded word

VTB0: DEC SI ;ptr to tens digit
;---copy word to memory for sorting
VTB1: MOVSW
CLC

VTB2: POP ES
POP DX
POP CX
POP BX
POP AX
RET

ENDP VAR TO BLK
Sort the Word Variables in [SegVar].

Input = AX = count of word variables.
Output = None

Note: this routine reassigns the DS and ES registers to [SegVar]

Special Note: It does not sort the first word of [SegVar] so
ranking variables begin at [SegVar] + 2 Offset and
go to (2 x numbers found) Offset.

This sort is based on the following TPASCAL procedure:
PROCEDURE Sort;         {A Shell Sort}
VAR
    Gap,J : Integer;
    Temp : string[13];
    TempNo : Integer;
Begin
    Gap := MaxRec Div 2;
    While gap > 0 Do
      Begin
        For I := (Gap + 1) to MaxRec Do
          Begin
            J := I-Gap;
            While J > 0 Do
              Begin
                  Begin
                    Temp := A[J];
                    A[J+Gap] := Temp;
                    J := J-Gap;
                  End;
                Else J := 0;
              End;
            End;
            Gap := Gap DIV 2;
          End;
      End;
    End;
    The following registers hold the above variables:
    AX = Gap; BX = J; CX = I; DX = MaxRec; and BP = temp storage

PROC VAR SORT
PUSH AX                     ;save registers
PUSH BX
PUSH CX
PUSH DX
PUSH DS
PUSH ES
PUSH BP
MOV DX,AX                   ;store MaxRec in DX
MOV AX,[VarSeg]             ;get index base segment
MOV DS,AX                   ;reassign the DS & ES
MOV ES,AX ;to ptr to the index.
XOR BX,BX ;zero buffer pointer
MOV AX,2020h ;two spaces to pad
MOV [BX],AX ;first two unused bytes
MOV AX,DX ;Gap = MaxRec
SHR AX,1 ;Gap = Gap Div by 2

VARS1: CMP AX,0 ;when Gap = 0 exit.
JLE VARS4 ;exit if <= 0
MOV CX,AX ;J is stored in CX
INC CX ;J = Gap + 1

VARS2: MOV BX,CX ;J in BX
SUB BX,AX ;J = 1 - Gap
JZ VARS3 ;skip if J = 0
JC VARS3 ;skip if J is < 0.
CALL COMPARE_VAR ;repeat until J = 0

VARS3: INC CX ;J = J + 1
CMP DX,CX ;Is J < or = MaxRec
JNC VARS2 ;If yes then loop.
SHR AX,1 ;Gap = Gap Div by 2
JMP SHORT VARS1

VARS4: POP BP ;restore registers
POP ES
POP DS
POP CX
POP BX
POP AX
RET ;sort is complete.

;---Compare and swap words if needed.

Input = AX = Gap; BX = J; DS & ES point to the base of index file.
Output = [none] Items swapped in memory if needed

PROC COMPARE_VAR
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
PUSH DX,AX ;save Gap in DX

; Compare the first 2 bytes of each pointer

COMV1: MOV BP,BX ;save J in BP
ADD AX,BX ;AX = J + Gap
SHL AX,1 ;ptr to J+Gap in mem
SHL BX,1 ;ptr to J in mem
CLD ;auto-Inc SI, DI
MOV DI,AX ;offset of J + Gap
MOV SI,BX ;offset of J
MOV CX,2 ;byte counter
REPE CMPSB ;compare strings
JLE COMV3 ;exit if < or =.

; Swap the 2 bytes of index record if string A > string A+Gap

G-16
MOV  DI, AX ; offset of J + Gap
MOV  SI, BX ; offset of J
MOV  AX, [SI] ; read word each str.
MOV  BX, [DI]
MOV  AX, [SI], BX ; write word each str.
MOV  AX, DX ; restore gap to AX
MOV  BX, BP ; restore J to BX
SUB  BX, AX ; J = J - gap
JZ  COMW ; exit if J = 0.
JNC  COMV1 ; continue if J > 0.
COMV3: POP  DX ; restore registers
POP  CX
POP  BX
POP  AX
RET ; return to Shell_Sort
ENDP  COMPARE_VAR
ENDP  VAR_SORT

; Copy first, last, median, 25th and 75th percentiles the [PerCrt] data string.
; Input = AX = Number of variables found
; Round AX to and even number = ptr to 50%  50/2 = 25%  50% + 25% = 75%
; CX = loop counter (to compute which variable)
; 6(counter-1) + 5 = position in PerCrt
; Output = Median and 25% and 75% stored in [PerCrt]

PROC  STORE_VAR
PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX
PUSH  ES
PUSH  AX
PUSH  DX, AX ; save number found
;——compute offset in PerCrt buffer based upon loop counter
DEC  CX ; loop count -1
MOV  AX, CX ; counter -1 to AL
MOV  CL, 10 ; multiplier
MUL  CL ; AX = AL times 10
ADD  AX, 5 ; offset for median value
MOV  BX, Offset PerCrt ; begin of var string
ADD  BX, AX ;BX= ptr to med variable
MOV  AX, [VarSeg] ;base of memory block
MOV  ES, AX ; ES ptr to block seg
;——get the lowest
MOV  DI, 2 ; ptr to lowest score
MOV  AX, [ES:DI] ; get lowest score
MOV  [BX], AX ; store lowest score
INC  BX ; advance [perCrt]
INC  BX ; word pointer.
;——get the highest

G-17
MOV AX,DX ;get 50% ptr
SHL AX,1 ;multiply by 2
MOV DI,AX ;ptr to highest score
MOV AX,[ES:DI] ;get last score
MOV [BX],AX ;store last score
INC BX
INC BX ;word pointer.

;---get the 50%
TEST DL,01h ;is the number even?
JZ STV1 ;if Yes, goto next test
INC DL ;if NO make it even

STV1: MOV DI,DX
MOV AX,[ES:DI] ;get 50% value
MOV [BX],AX ;store 50% value
INC BX ;advance [percent]
INC BX ;word pointer.

;---get the 25% and 27%
MOV AX,DX
SHR AX,1
TEST AL,01h ;is the number even?
JZ STV2 ;if Yes, goto next test
INC AX ;if NO make it even

STV2: MOV DI,DX
ADD DX,AX
MOV AX,[ES:DI] ;get 25% value
MOV [BX],AX ;store 25% value
INC BX ;advance [percent]
INC BX ;word pointer.
MOV DI,DX
MOV AX,[ES:DI] ;get 25% value
MOV [BX],AX ;store 25% value
CLC ;clear carry flag
POP ES
POP DX
POP CX
POP BX
POP AX
RET

ENDP STORE_VAR

; Input = none
; Output = none

PROC RANK_WAIT_MESS
PUSH AX
PUSH BX
PUSH CX
PUSH DX

;---please wait message to screen.
XOR AX,AX
CALL MENU_BOX
MOV CL,[Color] ;clear menu area
;save original attri
MOV AL,[Warning] ; warning color
MOV [Color],AL ; set color
MOV AX,010Bh ; row 3/Col 12
CALL GOTOXY ; set cursor
CALL CSTR_OUT ; display warning
db ' Please wait ....... Ranking the data file: ',0
MOV AX, Offset FileNa
CALL DSTR_OUT
CALL CSTR_OUT
db '.',0
MOV [Color],CL ; restore original attri
CALL HIDE_CUR
CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP RANK_WAIT_MESS

Input = AX = number of scores

CX = loop count 16 = columns 49 - 51

15 = columns 46 - 48 etc

Output = message to the screen

PROC PROGRESS_MESS
PUSH AX
PUSH BX
PUSH CX
PUSH DX

; please wait message to screen.
MOV BX,AX ; save no. of scores
MOV DL,[Color] ; save original attri
MOV AL,[Menu] ; menu color
MOV [Color],AL ; set color
MOV AX,010Bh ; row 3/Col 12
CALL GOTOXY ; set cursor
CALL CSTR_OUT ; display warning
db ' Please wait ....... Ranking ',0
MOV AX,BX ; restore no. of scores
CALL BIN_OUT
CALL CSTR_OUT
db ' scores in columns ',0
MOV AX,CX ; loop count to AX
SHL AX,1 ; multiplier by two
ADD AX,CX ; AX = 3(loop count)
INC AX
CALL BIN_OUT
CALL CSTR_OUT
db ' and ',0
INC AX
CALL BIN_OUT
CALL CSTR_OUT

G-19
db .0
MOV [Color].DL ;restore original attr
CALL HIDE_CUR
CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP PROGRESS_MESS

Check ends of sort for out of bounds data.
Input = AX = Number of variables found
Output = carry flag = abort ranking
EndOfArray = Offset AX x 2
BeginOfArray = Offset 2
:Note: The second byte of out of bounds data maybe be rounded up one ASCII no.

PROC CHECK_DATA
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH ES
SHL AX,1 ;No. found x 2 = offset
MOV DI,AX ;to EndOfArray
MOV AX,[VarSeg] ;base of memory block
MOV ES,AX ;ES ptr to block seg
MOV AX,[ES:DI] ;get end value
MOV DX,AX ;save value in DX
:==is endofarray value larger than "50"?
CMP AL,6 ;is digit > 5
JNC CDK3 ;if Yes then error
CMP AL,5 ;is it = '5'?
JNZ CKD1 ;is Yes check for '0'
CMP AH,0 ;is it = '0'
JNZ CDK3 ;if NO then error
JMP SHORT CKD2
CKD1: CMP AH,0 ;is digit > 9
JNC CDK3 ;if Yes then error
:==is beginofarray value less than "10" ?
CKD2: MOV DL,2 ;to BeginOfArray
MOV AX,[ES:DI] ;get end value
MOV DX,AX ;save value in DX
CMP AL,1 ;is digit < '1'?
JC CDK3 ;if Yes then error
CMP AH,0 ;is it < '0'?
JNC CDK4 ;if NO then normal exit
:==report data error
CDK3: CALL DATA_ERR ;inform user of error

G-20
CDK4: POP ES
                ;restore registers
    POP    DX
    POP    CX
    POP    BX
    POP    AX
    RET

ENDP    CHECK_DATA

Inform user of err found in the data file.
        Input CX = Loop counter (used to compute column number)
        DX = WORD that is out of bounds
        Output: Carry flag = abort ranking
;Note: The second byte of out of bounds data may be rounded up one ASCII no.
;A zero line no. means the GET_LINE_NO search failed. This should never happen!

PROC    DATA ERR
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX

;---- compute offset In [FilBuf]
        MOV AX,CX          ;loop counter to AX
        MOV AH,AL          ;save number in AH
        SHL AL,1           ;counter x 2
        ADD AL,AH          ;+ org counter = times 3
        XOR AH,AH          ;convert to 16 bits
        INC AX             ;change to 1 - ? form
        MOV CX,AX          ;CX = column no 1 - ?
        CALL GET_LINE_NO   ;line No. of error
        JNC DAE1           ;BP = Word found
        XOR AX,AX          ;0 = line not found????

DAE1:   MOV BX,AX     ;BX = line no 1 - ?
        MOV AL,[Warning]  ;warning color
        MOV DL,[Color]    ;save original color
        MOV [Color],AL    ;set color
        MOV AX,0207h       ;row 3/Col 6
        CALL GOTOXY       ;set cursor
        CALL CSTR OUT     ;display warning
        db " Data Error in file: ",0
        MOV SI, Offset ID + 1
        MOV AX,BP
        MOV [SI],AX
        MOV AX,SI
        CALL DSTR_OUT     ;display warning
        CALL CSTR_OUT     ;display warning
        db " in line ",0
        MOV AX,BX
        CALL BIN OUT
        CALL CSTR_OUT     ;display warning
        db ", column ",0
        MOV AX,CX

G-21
CALL BIN OUT
CALL CSTR_OUT
db " *. Press Any Key. ",0
MOV [Color],DL
CALL HIDE CUR
CALL ERR SOUND
CALL GET_CHAR
CMP AL,1Bh
JNZ DAE2
CALL ESC_YN
JC DAE3
DAE2: CALL CLEAR_MESSAGE
CLC
DAE3: POP DX
POP CX
POP BX
POP AX
RET
 ENDP DATA_ERR

--------- Locate a WORD in a given column of the data file. ---------

Input = DX = WORD (looking for word or word + 1)
CX = Column counter (1 - ? Form)
Output = AX = Line Number (1 to ?? form)
BP = WORD found
Carry flag = no find

PROC GET_LINE_NO
PUSH BX
PUSH CX
PUSH DX
PUSH DS
CALL GOTO_TOP
JC FDW4

;---assign buffer offset
MOV BX, Offset FIBuf
DEC CX
ADD BX,CX
XOR CX,CX
MOV BP,DX
DEC DH
FDW1: INC DH
CALL CLEAR_FIBBUF
CALL READ_LINE
JNC FDW2
MOV AL.,0FFH
MOV [EOF],AL
JZ FDW3
FDW2: INC CX
CMP [BX],DX
JZ FDW3
DEC DH
G-22
CMP [BX],DX ; is this a match
JZ FDW3

; --- is this the last line ?
XOR AL,AL ; zero AX register
CMP AL, [EOF] ; is EndOfFile TRUE?
JZ FDW1 ; False = get next line
STC ; cf = word not found
JMP FDW4 ; mark not found

; --- OK! Word is found
FDW3: MOV BP,DX ; return WORD in BP
MOV AX,CX ; line number to AX
CLC ; clear carry flag
FDW4: POP DS ; restore registers
POP DX
POP CX
POP BX
RET

ENDP GET_LINE_NO

; Input = none
; Output = carry flag = abort printing
PROC ESC YN
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL CLEAR MESSAGE ; store original Color
MOV CL,[Color] ; store original Color
MOV AL, [Warning] ; set color
MOV [Color], AL ; set color
MOV AX,020Dh ; row 3/Col 12
CALL GOTOYX ; set cursor
CALL CSTR_OUT ; display warning
db " Do you want to ABORT the ranking process? ",0
db " Y/N ",0
MOV [Color], CL ; restore original color
ESY1: CALL HIDE_CUR
CALL GET CHAR
AND AL,0DFh ; turn off bit 6
CMP AL, 'N' ; is it No?
JZ ESY4 ; if yes exit
ESY2: CMP AL, 'Y' ; is it Yes?
JNZ ESY3 ; if not continue
STC ; set carry flag = abort
JMP SHORT ESY5 ; exit
ESY3: CALL ERR SOUND
JMP SHORT ESY1
ESY4: CALL CLEAR MESSAGE ; empty message line
CLC ; clear cf = continue
ESY5: POP DX
POP CX
POP BX
POP AX
RET
ENDP ESC_YN

--- Instructions for rank command.
  Input = None
  Output = None

PROC RANK_INSTRU
  ; save registers
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  MOV AX,1500h ; row 21, column 0
  CALL MENU_BOX ; draw menu box
  MOV CL,[Color] ; get assigned color
  MOV AL,[Menu] ; get menu color
  MOV [Color].AL ; set menu color
  MOV AX,160Ah ; row 22, column 12
  CALL GOTOXY
  CALL CSTR
  OUT db 'Press the <Esc> key to pause or cancel the '
  db 'ranking of scores.',0
  CALL HIDE_CUR
  MOV [Color].CL ; restore assigned color
  POP DX ; restore registers
  POP CX
  POP BX
  POP AX
  RET
ENDP RANK_INSTRU

--- Append the [PerCnt] string to the data file.
  Input = None
  Output = [PerCnt] variables to end of data file.

PROC APPEND_FILE
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  PUSH ES
  MOV AX,DS ; Make ES = DS
  MOV ES,AX
  MOV BX,Offset PerCnt
  ; place <return> and <line feed> beginning of data string
  MOV AX,0A0Dh ; line feed & carry ret
  MOV [BX].AX ; place in data string
  INC BX

G-24
INC BX

--- place end ID of '000' in data string
MOV AX,'00'
MOV [BX],AX
INC BX
INC BX
MOV [BX],AX

--- place two (<return> + <line feed>) at end of string
MOV BX,Offset PerCnt + 165
MOV AX,0A0DH            ;line feed & carry ret
MOV [BX],AX             ;place in data string
INC BX                  ;ptr to PerCnt + 103
INC BX
MOV [BX],AX             ;place in data string

--- place file pointer to the End of File.
MOV BX,[FileHd]        ;get file handle
XOR CX,CX              ;set offset = 0
MOV DX,CX              ;set offset = 0
MOV AX,4202h           ;set file pointer no.
INT 21h                ;set to End of File
JC APP1                 ;exit if error.

--- inform user if disk is full
MOV AX,169             ;number of bytes needed
CALL IS FULL           ;is room available?
JC APP1                ;if NO skip write

--- Append 169 bytes to the file.
MOV AX,4000h           ;write to file: func. no.
MOV CX,169             ;no. of bytes to write
MOV DX,Offset PerCnt   ;ptr to data to write
INT 21h                ;write to the file

--- No error checking because nothing is lost if the write fails.
APP1:       CLC           ;clear carry flag
            POP ES          ;restore registers
            POP DX
            POP CX
            POP BX
            POP AX
            RET

ENDP APPEND_FILE

.CODE

--- Print a report for each ID number in the data file.
Input = None
Output = None
BX = DOS timer ticks + 25 seconds
PROC PRINT_ALL_REPORTS
PUSH AX                ;save registers
PUSH BX
PUSH CX
PUSH DX
CALL GOTO_TOP         ;file ptr to top of file
G-25
JC PRA5
CALL PROGRESS MESSAGE ;exit on DOS error
CALL PRINT INSTRU ;inform user of progress
CALL INITIALIZE_HP ;display bottom box
JC PRA4 ;hp to portrait mode
;exit if printer error

;get DOS timer ticks
PRA1: MOV AH,0
INT 1Ah
MOV BX,DX
MOV AX,455
ADD BX,AX
JC PRA1

;locate next ID number
CALL FIND_DATA_LINE
JC PRA3
CALL PROGRESS MESSAGE
CALL PRINT TITLE
JC PRA4

;print each dimension
CALL PRINT DIM1
JC PRA4
CALL PRINT DIM2
JC PRA4
CALL PRINT DIM3
JC PRA4
CALL PRINT DIM4
JC PRA4
CALL EJECT
CALL PRA4

;is 25 seconds up yet?
PRA2: MOV AH,0
INT 1Ah
CMP DX,BX
JC PRA2

;loop until all graphs are printed
JMP SHORT PRA1

;normal exit point
PRA3: CALL RESTORE_HP
CLC
JMP SHORT PRA5

;abort exit point
PRA4: CALL RESTORE_HP
STC
PRA5: POP DX
POP CX
POP BX
POP AX
RET

ENDP PRINT_ALL_REPORTS

; --Print a report for a user supplied ID number.
; Input = None
Output = None

PROC PRINT ONE REPORT
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX

    --- get ID number
    CALL GET_ID
    JC PRI5

    --- locate ID number in the data file
    CALL LOCATE
    JC PRI5

    --- printing of a report begins here
    CALL PROGRESS MESSAGE
    CALL PRINT_INSTRU
    CALL INITIALIZE_HP
    JC PRI4
    CALL PRINT_TITLE
    JC PRI4

    --- print each dimension
    CALL PRINT_DIM1
    JC PRI4
    CALL PRINT_DIM2
    JC PRI4
    CALL PRINT_DIM3
    JC PRI4
    CALL PRINT_DIM4
    JC PRI4
    CALL EJECT

PRI3:    CALL RESTORE_HP
    CLC
    JMP SHORT PRI5

PRI4:    CALL RESTORE_HP
    STC

PRI5:    POP DX
    POP CX
    POP BX
    POP AX

    RET
ENDP PRINT ONE REPORT

;--- Print positive dimension number 1 on an HP laser.
; Input = None
; Output = Carry flag = abort printing
PROC PRINT_DIM1
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX

    --- print dimension name

G-27
MOV AX, 0B08h
CALL HPGOTOYX
JNC P10
JMP P13

; Is this a POS or NEG dimensions?
P10: CMP BYTE PTR [Report], 0
    JZ P11
    MOV AX, Offset Neg1
    JMP SHORT P12

P11: MOV AX, Offset Pos1
CALL PRINT_STRING
    JC P13

; Draw chart outline
MOV AX, 0B22h
CALL HPGOTOYX
JC P13
MOV AX, Offset Box
CALL PRINT STRING
JC P13

; Chart percentile for self, peers, superiors and subordinates
MOV AX, 0A30h
MOV BX, Offset Percnt + 5
CALL CHART_RANGE_L
JC P13
CALL CHART_PERCENTILES
JC P13
INC AH
MOV BX, Offset Percnt + 45
CALL CHART_RANGE_D
JC P13
CALL CHART_PERCENTILES
JC P13
INC AH
MOV BX, Offset Percnt + 85
CALL CHART_RANGE_L
JC P13
CALL CHART_PERCENTILES
JC P13
INC AH
MOV BX, Offset Percnt + 125
CALL CHART_RANGE_D
JC P13
CALL CHART_PERCENTILES
JC P13

; Chart points for self, peers, superiors and subordinates
MOV AX, 0A30h
MOV BX, Offset FIBuf + 3
CALL CHART_POINT
JC P13
INC AH
MOV BX, Offset FIBuf + 15
CALL CHART_POINT

JC P13 ; exit on error
INC AH ; row to superiors
MOV BX, Offset FLBuf + 27 ; ptr to self point
CALL CHART_POINT ; ax = starting row/col
JC P13 ; exit on error
INC AH ; row to subordinates
MOV BX, Offset FLBuf + 30 ; ptr to self point
CALL CHART_POINT ; ax = starting row/col
JC P13 ; exit on error

; check if to see if any special messages need to be printed.
; NOTE: to be completed at a later date.

CLC ; clear carry flag
P13: POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP PRINT_DIM1

; Print positive dimension number 2 on an HP laser.
; Input = None
; Output = Carry flag = abort printing
PROC PRINT_DIM2
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX

; print dimension name
MOV AX, 1300h ; row/col hex
CALL HPGOTOYX ; set position
JNC P20 ; if on error continue
JMP P23 ; exit on error

; Is this a POS or NEG dimensions?
P20: CMP BYTE PTR [Report], 0 ; get type of report
JZ P21 ; jump if positive
MOV AX, Offset Neg2 ; ptr to derallment str
JMP SHORT P22 ; jump to print the str

P21: MOV AX, Offset Pos2 ; ptr to positive str
P22: CALL PRINT_STRING ; print the string
JC P23 ; exit on printer error

; draw chart outline
MOV AX, 1022h ; row/col hex
CALL HPGOTOYX ; set position
JC P23 ; exit on error
MOV AX, Offset Box ; ptr to draw box string
CALL PRINT_STRING ; draw the dim box
JC P23 ; exit on printer error

; chart percentile for selfs, peers, superiors and subordinates
MOV AX, 1230h ; row/col of '1' hex
MOV BX, Offset PerCnt + 15 ; ptr to self data
CALL CHART_RANGE_L ; draw shaded area
JC P23 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P23 ;exit on error
INC AH ;row to peers
MOV BX,Offset Percnt + 55 ;ptr to peers data
CALL CHART_RANGE_D ;draw shaded area
JC P23 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P23 ;exit on error
INC AH ;row to peers
MOV BX,Offset Percnt + 95 ;ptr to superiors data
CALL CHART_RANGE_L ;draw shaded area
JC P23 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P23 ;exit on error
INC AH ;row to superiors
MOV BX,Offset Percnt + 135 ;ptr to superiors data
CALL CHART_RANGE_D ;draw shaded area
JC P23 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P23 ;exit on error
---chart points for self, peers, superiors and subordinates
MOV AX,1230h ;restore row/col of '1'
MOV BX,Offset FIBuf + 6 ;ptr to self point
CALL CHART_POINT ;ax = starting row/col
JC P23 ;exit on error
INC AH ;row to peers
MOV BX,Offset FIBuf + 18 ;ptr to self point
CALL CHART_POINT ;ax = starting row/col
JC P23 ;exit on error
INC AH ;row to superiors
MOV BX,Offset FIBuf + 30 ;ptr to self point
CALL CHART_POINT ;ax = starting row/col
JC P23 ;exit on error
INC AH ;row to subordinates
MOV BX,Offset FIBuf + 42 ;ptr to self point
CALL CHART_POINT ;ax = starting row/col
JC P23 ;exit on error
---check if to see if any special messages need to be printed.
;NOTE: to be completed at a later date.

CLC ;clear carry flag
P23: POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP PRINT_DIM2

---Print positive dimension number 3 on an HP laser.
;Input = None
;Output = Carry flag = abort printing

G-30
PROC PRINT_DIM3
PUSH AX
PUSH BX
PUSH CX
PUSH DX

--print dimension name
MOV AX,1808h  ;row/col hex
CALL HPGOTOYX ;set position
JNC P30        ;if on error continue
JMP P33        ;exit on printer error

--is this a POS or NEG dimensions?
P30: CMP BYTE PTR [Report],0     ;get type of report
    JZ P31 ;jump if positive
    MOV AX,Offset Neg3  ;ptr to derallrement str
    JMP SHORT P32 ;jump to print the str

P31: MOV AX,Offset Pos3  ;ptr to positive str
P32: CALL PRINT_STRING ;print the string
    JC P33 ;exit on printer error

--draw chart outline
MOV AX,1822h  ;row/col hex
CALL HPGOTOYX ;set position
JC P33 ;exit on error
MOV AX,Offset Box  ;ptr to draw box string
CALL PRINT_STRING ;draw the dim box
JC P33 ;exit on printer error

--chart percentile for selfs, peers, superiors and subordinates
MOV AX,1A30h  ;row/col of '1' hex
MOV BX,Offset PerCnt + 25  ;ptr to self data
CALL CHART_RANGE_L ;draw shaded area
JC P33 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P33 ;exit on error
INC AH ;row to peers
MOV BX,Offset PerCnt + 65  ;ptr to peers data
CALL CHART_RANGE_D ;draw shaded area
JC P33 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P33 ;exit on error
INC AH ;row to superiors
MOV BX,Offset PerCnt + 105 ;ptr to superiors data
CALL CHART_RANGE_L ;draw shaded area
JC P33 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P33 ;exit on error
INC AH ;row to subordinates
MOV BX,Offset PerCnt + 145 ;ptr to subordinates
CALL CHART_RANGE_D ;draw shaded area
JC P33 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P33 ;exit on error

--chart points for self, peers, superiors and subordinates
MOV AX,1A30h  ;restore row/col of '1'
G-31
MOV  BX, Offset FliBuf + 9  ;ptr to self point
CALL  CHART_POINT  ;ax = starting row/col
JC  P33  ;exit on error
INC  AH  ;row to peers
MOV  BX, Offset FliBuf + 21  ;ptr to self point
CALL  CHART_POINT  ;ax = starting row/col
JC  P33  ;exit on error
INC  AH  ;row to superiors
MOV  BX, Offset FliBuf + 33  ;ptr to self point
CALL  CHART_POINT  ;ax = starting row/col
JC  P33  ;exit on error
INC  AH  ;row to subordinates
MOV  BX, Offset FliBuf + 45  ;ptr to self point
CALL  CHART_POINT  ;ax = starting row/col
JC  P33  ;exit on error

;----check if to see if any special messages need to be printed.
;   NOTE: to be completed at a later date.

CLC  ;clear carry flag

P33:  POP  DX  ;restore registers
       POP  CX
       POP  BX
       POP  AX
       RET

ENDP  PRINT_DIM3

;----Print positive dimension number 4 on an HP laser.
;   Input = None
;   Output = Carry flag = abort printing
PROC  PRINT_DIM4
       PUSH  AX  ;save registers
       PUSH  BX
       PUSH  CX
       PUSH  DX

       ;print dimension name
       MOV  AX, 2308h  ;row/col hex
       CALL  HPGOTOXY  ;set position
       JNC  P40  ;if on error continue
       JMP  P43  ;exit on printer error

;----Is this a POS or NEG dimensions?
P40:  CMP  BYTE PTR [Report], 0  ;get type of report
       JZ  P41  ;jump if positive
       MOV  AX, Offset Neg4  ;ptr to derailment str
       JMP  SHORT P42  ;jump to print the str

P41:  MOV  AX, Offset Pos4  ;ptr to positive str
       CALL  PRINT_STRING  ;print the string
       JC  P43  ;exit on printer error

;----draw chart outline
       MOV  AX, 2022h  ;row/col hex
       CALL  HPGOTOXY  ;set position
       JC  P43  ;exit on error
       MOV  AX, Offset Box  ;ptr to draw box string

G-32
CALL PRINT_STRING ;draw the dim box
JC P43 ;exit on printer error

;chart percentile for selfs, peers, superiors and subordinates
MOV AX,2230h ;row/col of '1' hex
MOV BX,Offset PerCnt + 35 ;ptr to self data
CALL CHART_RANGE_L ;draw shaded area
JC P43 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P43 ;exit on error
INC AH ;row to peers
MOV BX,Offset PerCnt + 75 ;ptr to peers data
CALL CHART_RANGE_D ;draw shaded area
JC P43 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P43 ;exit on error
INC AH ;row to superiors
MOV BX,Offset PerCnt + 115 ;ptr to superiors data
CALL CHART_RANGE_D ;draw shaded area
JC P43 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P43 ;exit on error
INC AH ;row to subordinates
MOV BX,Offset PerCnt + 155 ;ptr to superiors data
CALL CHART_RANGE_D ;draw shaded area
JC P43 ;exit on error
CALL CHART_PERCENTILES ;draw the data
JC P43 ;exit on error

;chart points for self, peers, superiors and subordinates
MOV AX,2230h ;restore row/col of '1'
MOV BX,Offset FibBuf + 12 ;ptr to self point
CALL CHART_POINT ;ax=starting row/col
JC P43 ;exit on error
INC AH ;row to peers
MOV BX,Offset FibBuf + 24 ;ptr to self point
CALL CHART_POINT ;ax=starting row/col
JC P43 ;exit on error
INC AH ;row to superiors
MOV BX,Offset FibBuf + 36 ;ptr to self point
CALL CHART_POINT ;ax=starting row/col
JC P43 ;exit on error
INC AH ;row to subordinates
MOV BX,Offset FibBuf + 48 ;ptr to self point
CALL CHART_POINT ;ax=starting row/col
JC P43 ;exit on error

;check if to see if any special messages need to be printed.

;NOTE: to be completed at a later date.

CLC ;clear carry flag
P43: POP DX ;restore registers
POP CX
POP BX
POP AX

G-33
RET
ENDP PRINT_DEST

.CODE

; Send ASCII string to the Line Printer at port [LPT]
; Input = AX pointer to beginning of string in data section
; CH = number of tries if busy  CL = store char
; Output = Carry flag = abort printing
PROC PRINT STRING
    PUSH AX ; save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV BX,AX ; ptr to ASCII string
    XOR CH,CH ; zero loop counter
    ; Check keyboard buffer to see if the <Esc> key been pressed?
    PS1: MOV AX,0800H ; DOS function # 6
    MOV DL,OFFh
    INT 21h ; read char from keyboard buffer.
    JZ PS2 ; NO key pressed continue
    CMP AL,1Bh ; was it the <ESC> key?
    JNZ PS2 ; if NO continue
    CALL PRT_ERROR3 ; if YES inform user
    JC PS8 ; carry flag = abort
    ; get character to be sent to LPT port
    PS2: MOV AL,[BX] ; load Char to send
    CMP AL,0 ; is this end of string?
    JZ PS8 ; if yes normal exit.
    ; send character to assigned LPT port
    MOV AH,0 ; BIOS function number
    MOV DX,[LPT] ; get LPT port assign.
    INT 17h ; get port status
    CMP BYTE PTR [Debug],0 ; is debug ON?
    JZ PS3 ; if NO goto next test
    CALL SHOW AH ; bitmap of AH to screen
    ; test bit 5 of 8. If bit 5 = 0 then no power.
    PS3: TEST AH,10h ; is printer powered up?
    JNZ PS5 ; OKI<>0 goto next test
    CALL PRT_ERROR1 ; display error message
    JC PS8 ; carry flag = abort
    CALL PROGRESS MESSAGE ; inform user of progress
    JMP SHORT PS1 ; send same char again
    ; test bit 4 & 6 of 8. bit 4 = I/O error; 6 = printer out of paper.
    PS5: XOR CH,CH ; loop counter to zero
    TEST AH,28h ; I/O or out of paper?
    JZ PS6 ; if NO send char
    CALL PRT_ERROR2 ; if YES tell user.
    JC PS8 ; cf = abort
    CALL PROGRESS MESSAGE ; inform user of progress
    JMP SHORT PS1 ; send same char again

G-34
---test bit 1 of 8. If bit 1 = 1 then printer time-out.
PS6: TEST AH,01 ;is printer time-out?
    JZ PS7 ;if NO send next char
    CALL PRT ERROR4 ;inform user or timeout
    JC PS8 ;cf = abort else
    CALL PROGRESS MESSAGE ;inform user of progress
    JMP SHORT PS1 ;send same char again
PS7: INC BX ;point to next char
    JMP SHORT PS1 ;loop until finished
PS8: POP DX ;restore registers
    POP CX
    POP BX
    POP AX
    RET

.. Input = none
.. Output = carry flag = abort printing
PROC PRT ERROR1
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLEAR MESSAGE ;empty message line
    MOV CL,[Color] ;store original Color
    MOV AL,[Warning] ;warning color
    MOV [Color],AL ;set color
    MOV AX,020h ;row 3/Col 12
    CALL GOTOYX ;set cursor
    CALL CSTR OUT display warning
    db "Printer is off line. Do you want to try again? * ",0
    db " Y/N ",0
    MOV [Color],CL ;restore original color
PRE1: CALL HIDE CUR
    CALL ERR SOUND
    CALL GET CHAR
    AND AL,5Fh ;turn off bits 6 & 8
    CMP AL,'N' ;is it No?
        JZ PRE4 ;if yes exit
    PRE2: CMP AL,'Y' ;is it Yes?
        JNZ PRE3 ;if not continue
    CLC ;clear carry flag
    JMP SHORT PRE5 ;exit
PRE3: CALL ERR SOUND
    JMP SHORT PRE1
PRE4: CALL CLEAR MESSAGE ;empty message line
    STC ;set carry flag
    PRE5: POP DX
        POP CX
        POP BX
        POP AX
        RET
ENDP PRT ERROR1

G-35
Input = none
Output = carry flag = abort printing

PROC PRT_ERROR2
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLEAR_MESSAGE
    MOV CL,[Color] ; store original color
    MOV AL,[Warning] ; warning color
    MOV [Color],AL ; set color
    MOV AX,0207h ; row 3/Col 12
    CALL GOTOXY ; set cursor
    CALL CSTR_OUT ; display warning
    db " Printer Error. Check the paper. Do you want to continue? "
    db " Y/N ".0
    MOV [Color],CL ; restore original color
    PRR1: CALL HIDE_CUR
    CALL ERR_SOUND
    CALL GET_CHAR
    AND AL,5Fh ; turn off bit 6 & 8
    CMP AL,'N' ; is it No?
    JZ PRR4 ; if yes exit
    PRR2: CMP AL,'Y' ; is it Yes?
    JNZ PRR3 ; if not continue
    CLC ; clear carry flag
    JMP SHORT PRR5 ; exit
    PRR3: CALL ERR_SOUND
    JMP SHORT PRR1
    PRR4: CALL CLEAR_MESSAGE ; empty message line
    STC ; set carry flag
    PRR5: POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP PRT_ERROR2

Input = none
Output = carry flag = abort printing

PROC PRT_ERROR3
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLEAR_MESSAGE
    MOV CL,[Color] ; store original color
    MOV AL,[Warning] ; warning color
    MOV [Color],AL ; set color
    MOV AX,020Bh ; row 3/Col 12

G-36
CALL GOTOYX  ;set cursor
CALL CSTR_OUT  ;display warning
  db  "Do you want to ABORT the print instructions ? "
  db  "Y/N ".0
MOV [Color],CL  ;restore original color

PEE1: CALL HIDE CUR
CALL GET_CHAR
  AND AL,5Fh  ;turn off bit 6 & 8
  CMP AL,"N"  ;Is it No?
  JZ PEE4  ;if yes exit
  PEE2: CMP AL,"Y"  ;Is it Yes?
  JNZ PEE3  ;if not continue
  STC  ;set carry flag = abort
  JMP SHORT PEE5  ;exit
PEE3: CALL ERR SOUND
JMP SHORT PEE1

PEE4: CALL CLEAR_MESSAGE  ;empty message line
CLC  ;clear cf = continue
PEE5: POP DX
POP CX
POP BX
POP AX
RET

ENDP PRT_ERROR3

; Input = none
; Output = carry flag = abort printing
PROC PRT_ERROR4
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL CLEAR_MESSAGE  ;store original Color
MOV CL,[Color]
MOV AL, [Warning]  ;warning color
MOV [Color], AL
MOV AX,0207h  ;row 3/Col 12
CALL GOTOYX  ;set cursor
CALL CSTR_OUT  ;display warning
  db  "Printer Time-Out. Press any key to try again or <Esc> ";
  db  "to abort. ".0
  MOV [Color], CL  ;restore original color
  CALL HIDE CUR
  CALL ERR_SOUND
  CALL GET_CHAR
  CMP AL,18h  ;Is it <Esc>
  JNZ RPP1  ;if yes exit
  STC  ;set carry flag = abort
  JMP SHORT RPP2  ;exit
RPP1: CALL CLEAR_MESSAGE  ;empty message line
  CLC  ;clear cf = continue
RPP2: POP DX
      POP CX
      POP BX
      POP AX
      RET
ENDP PRT_ERROR4
ENDP PRINT_STRING

; Locate the next ID data line in the file.
; Input = None
; Output = Carry Flag if EndOfFile
; Carry flag if DOS error
PROC FIND_DATA_LINE
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  MOV AX, DS
  MOV ES, AX
; set default ID string to ASCII zeros
  MOV DI, Offset ID
  MOV AX, 3030h
  MOV [DI], AX
  XOR AH, AH
  MOV DI, Offset ID
  INC DI
  INC DI
  MOV [DI], AX
; locate ID number in the data file
  FID1: CALL READ_LINE
        CALL CLEAR_FILEBUF
        JNC FID2
        MOV AL, OFFFH
        MOV [EOF], AL
; is this a '000' data line?
  FID2: MOV CX, 3
        MOV DI, Offset ID
        MOV SI, Offset FileBuf
        CLD
        REPZ CMPSB
        JZ FID5
; is this an ID number line?
  FID4: MOV AL, [SI]
        CMP AL, '0'
        JC FID5
        CMP AL, ':'
        JNC FID5
        INC SI
        LOOP FID4
; copy ID to [ID] string
  G-38
MOV CX, 3 ; number of bytes to move
MOV DI, Offset ID ; ptr to ID number
MOV SI, Offset FIBuf ; ptr to data file line
CLD ; auto inc DI and SI
REP MOVSB ; copy three bytes to ID
JMP SHORT FID6 ; Exit found

FID6: XOR AL, [EOF] ; is EndOfFile TRUE?
CMP AL, [EOF] ; False = get next line
JZ FID1 ; carry flag = None
STC ; carry flag = None
JMP SHORT FID7 ; retrun EndOfFile

FID7: POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP FIND_DATA_LINE

--- Print a title on the HP laser.
Input = None
Output = None
PROC PRINT TITLE
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX, 0321h ; row/col hex
CALL HPGOTOYX
JC PT5 ; exit on printer error
MOV AX, Offset Heder ; ptr Title
CALL PRINT STRING ; string holds Bold ON
JC PT5 ; exit on printer error
MOV AX, 0508h ; row/col hex
CALL HPGOTOYX
JC PT5 ; exit on printer error
MOV AX, Offset IDStr ; ptr to demallment str
CALL PRINT STRING
JC PT5 ; exit on printer error
CMP BYTE PTR [Report], 0 ; get type of report
JZ PT1 ; jump if positive
MOV AX, Offset NegT ; ptr to demallment str
JMP SHORT PT2 ; jump to print the str

PT1: MOV AX, Offset PosT ; ptr to positive str
CALL PRINT STRING ; print the string
JC PT5 ; exit on printer error
MOV AX, 0553h ; row/col hex
CALL HPGOTOYX
JC PT5 ; exit on printer error
MOV AX, Offset DTStr ; ptr to demallment str
CALL PRINT STRING

G-39
JC PT5 ; exit on printer error
MOV AX,0786h ;row/col hex
CALL HPGOTOYX
JC PT5 ; exit on printer error
MOV AX,Offset DisStr
CALL PRINT_STRING ;string holds Bold OFF
JC PT5 ; exit on printer error
MOV AX,2822h ;row/col hex
CALL HPGOTOYX
JC PT5 ; exit on printer error
MOV AX,Offset Inform
CALL PRINT_STRING ;bottom information box
JC PT5 ; exit on printer error
PT5: POP DX ; restore registers
POP CX
POP BX
POP AX
RET
ENDP PRINT_TITLE

Is the data file, report type and LPT port ready?
Input = None
Output = Carry flag if Not Ready.
PROC IS PRINT
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
;—— is file selected?
MOV BX,[FileHd] ; get file handle
CMP BX,0 ; is file open?
JNZ PR1
CALL FILE_ERR
JMP SHORT PR4
;—— were the percentiles in the data file?
PR1: MOV CL,[Ranked] ; are percentiles set?
CMP CL,0 ; is a type selected?
JNZ PR2
CALL RANK_ERR
JMP SHORT PR4
;—— is the printer online?
PR2: CALL ON_LINE
JNC PR4
CALL LPT_ERR
PR4: STC ; set error flag
PR5: POP DX ; restore registers
POP CX
POP BX
POP AX
RET
ENDP IS PRINT
PROC LPT_ERR
  CALL 'CLEAR_MESSAGE
  MOV AL,' (Warning) ;'warning color
  MOV CL,'(Color) ;save original color
  MOV [Color],AL ;set color
  MOV AX,0207h ;row 3/Col 8
  CALL GOTOYX ;set cursor
  CALL CSTR_OUT ;display warning
  db 'Printer Not On Line! Check power or LPT assignment.'
  db 'Press Any Key .0
  MOV [Color],CL ;restore original color
  CALL HIDE_CUR
  CALL ERR_SOUND
  CALL GET_CHAR
  RET
ENDP LPT_ERR

PROC RANK_ERR
  CALL 'CLEAR_MESSAGE
  MOV AL,' (Warning) ;'warning color
  MOV CL,'(Color) ;save original color
  MOV [Color],AL ;set color
  MOV AX,0207h ;row 3/Col 8
  CALL GOTOYX ;set cursor
  CALL CSTR_OUT ;display warning
  db 'Use the 'Rank' command to compute the percentiles.'
  db 'Press Any Key .0
  MOV [Color],CL ;restore original color
  CALL HIDE_CUR
  CALL ERR_SOUND
  CALL GET_CHAR
  RET
ENDP RANK_ERR

PROC FILE_ERR
  CALL 'CLEAR_MESSAGE
  MOV AL,' (Warning) ;'warning color
  MOV CL,'(Color) ;save original color
  MOV [Color],AL ;set color
  MOV AX,0207h ;row 3/Col 8
  CALL GOTOYX ;set cursor
  CALL CSTR_OUT ;display warning
  db 'Use the 'File' command to select a SDLI data file.'
  db 'Press Any Key .0
  MOV [Color],CL ;restore original color
  CALL HIDE_CUR
  RET

G-41
CALL ERR_SOUND
CALL GET_CHAR
RET

ENDP FILE_ERR

Input = none
Output = none

PROC PROGRESS_MESSAGE
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV AL,[Menu] ;menu color
MOV CL,[Color] ;save original color
MOV [Color],AL ;set color
MOV AX,0207h ;row 3/Col 8
CALL GOTOYX ;set cursor
CALL CSTR_OUT ;display string
db ' Please wait .......... Printing report for ID Number: ',0
MOV AX, Offset ID
CALL DSTR_OUT
CALL CSTR_OUT
db '.,0 ;endOfString marker
MOV [Color],CL ;restore original color
CALL HIDE_CUR
POP DX
POP CX
POP BX
POP AX
RET

ENDP PROGRESS_MESSAGE

Input = ASCII number string in [ID]
Output = none

PROC LOCATE_MESSAGE
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL CLEAR_MESSAGE ;menu color
MOV AL,[Menu] ;save original color
MOV CL,[Color] ;set color
MOV AX,0207h ;row 3/Col 8
CALL GOTOYX ;set cursor
CALL CSTR_OUT ;display string
db ' Please wait .......... Searching file for ID Number: ',0
MOV AX, Offset ID
CALL DSTR_OUT
CALL CSTR_OUT ;display period

G-42
db '.0
MOV [Color],CL ;restore original color
CALL HIDE_CUR
POP DX
POP CX
POP BX
POP AX
RET
ENDP LOCATE_MESSAGE

;---Request Printer Port Status
; Input = Assign port in [LPT] 0 - 2
; Output = Carry Flag = port not ready
PROC ON_LINE
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,0200h ;get status function no
MOV DX,[LPT] ;ptr to ptr
INT 17h ;request status
AND AH,10h ;is printer ready?
JNZ ISR1 ;0 means printer error
STC ;set carry flag
ISR1: POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP ON_LINE

;---Eject paper on HP laser.
; Input = None
; Output = None
PROC EJECT
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,Offset FFeed
CALL PRINT_STRING
POP AX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP EJECT

;---Initialize the HP laser.
; Input = None
; Output = None
PROC INITIALIZE_HP
PUSH AX  ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,Offset Init
CALL PRINT_STRING
POP DX
POP CX
POP BX
POP AX
RET
ENDP INITIALIZE_HP

;---restore default setting to the HP laser.
; Input = None
; Output = None
PROC RESTORE_HP
PUSH AX  ;save registers
PUSH BX
PUSH CX
PUSH DX
CALL RESTORE_MESS  ;inform user
MOV SI,Offset Rest  ;ptr to ASCII string
;---get character to be sent to LPT port
RR1: MOV AL,[SI]     ;load Char to send
   CMP AL,0       ;is this end of string?
   JZ RR4        ;if yes normal exit.
   XOR AH,AH     ;0 = BIOS function No.
   MOV DX,[LPT]  ;get LPT port assign.
   INT 17h       ;send char to printer
;---test bit 5 of 8. If bit 5 = 0 then no power.
   TEST AH,10h   ;is printer powered up?
   JZ RR4        ;exit if NO
;---test bit 1 of 8. If bit 1 = 1 then printer time-out
   TEST AH,01    ;is printer time-out?
   JNZ RR4      ;if YES then exit
;---pause 1/3 second or up to 1/6 second once each hour.
RR2: MOV AH,0       ;function number
   INT 1Ah       ;get DOS clock ticks
   MOV BX,DX     ;save ticks in CX
   MOV AX,3      ;18.2 ticks per second
   ADD BX,AX     ;add 15 seconds
   JC RR2       ;loop if over flow
RR3: MOV AH,0       ;function number
   INT 1Ah       ;get DOS clock ticks
   CMP DX,BX     ;has time run out?
   JC RR3       ;if not loop until done
   INC SI       ;point to next char
   JMP SHORT RR1 ;loop until finished
RR4: CLC

G-44
POP DX ; restore registers
POP CX
POP BX
POP AX
RET

Input = none
Output = none

PROC RESTORE MESS
CALL CLEAR MESSAGE ; menu color
MOV AL,[Menu]
MOV CL,[Color] ; save original color
MOV [Color],AL ; set color
MOV AX,020Ah ; row 3/Col 8
CALL GOTOYX ; set cursor
CALL CSTR_OUT ; display warning
db 'Please wait while resetting the'
db ' HP Printer.',0
MOV [Color],CL ; restore original color
CALL HIDE_CUR
RET
ENDP RESTORE MESS
ENDP RESTORE HP

--- Place printer cursor in row/col position
Input = AX = row/col in hex
Output = None

PROC HPGOTOYX
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV BX,AX ; save row/col
CMP AL,100 ; is col < 100 ?
JC GOT1 ; if yes Ok continue
XOR AL,AL ; if NO column = 0
GOT1: XOR AH,AH ; zero to high byte
MOV CL,10 ; divisor to CL
DIV CL ; convert to decimal
OR AX,3030h ; convert to ASCII digit
MOV [Col],AX ; save digit
MOV AL,BH ; move row to AL
CMP AL,100 ; is row < 100 ?
JC GOT2 ; if yes Ok continue
XOR AL,AL ; if NO row = 0
GOT2: XOR AH,AH ; zero to high byte
MOV CL,10 ; divisor to CL
DIV CL ; convert to decimal
OR AX,3030h ; convert to ASCII digit
MOV [Row],AX ; save digit
MOV AX,Offset GoTo
CALL PRINT_STRING
POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP HPGOTOYX

Show the contents of the AH register to screen.
Used for showing feedback from LPT port using INT 17h calls
Input = None
Output = None
Called from: PRINT_STRING if [Debug] is ON

PROC SHOW_AH
PUSH AX
PUSH BX
PUSH CX
PUSH DX
;---display contents of AH register in binary
MOV BL,AH ;save input in BX
MOV AX,0734h ;row/colm
call HPGOTOYX
call CSTR_Out
db 'low',0
MOV CX,8 ;loop counter
AHO: MOV AX,BX
AND AX,1 ;zero all bit but first
CALL BIN_OUT
CMP CL,5
JNZ AH1
CALL CSTR_OUT
db ' to ',0
AH1: SHR BL,1 ;loop 8 times
LOOP AHO
CALL CSTR_OUT
db 'high',0
CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP SHOW_AH

---Instructions for the Print command.
Input = None
Output = None

PROC PRINT_INSTRU
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,1500h ;row 21, column 0
CALL MENU_BOX ;draw menu box
MOV CL,[Color] ;get assigned color
MOV AL,[Menu] ;get menu color
MOV [Color],AL ;set menu color
MOV AX,180Bh ;row 22, column 12
CALL GOTOYX
CALL CSTR_OUT
db 'Press the <Esc> key to pause or cancel the'
db 'printing of reports.',0
CALL HIDE_CUR
MOV [Color],CL ;restore assigned color
POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP PRINT_INSTRU

;-----Find ID number in current open data file.
; Input = three digit ASCII number in [ID]
; Output = Carry flag = not found

PROC LOCATE
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,DS ;assign ES = DS
MOV ES,AX
CALL GOTO_TOP ;file ptr to BegOfFile
JC SEAl ;exit on error
;-----locate ID number in the data file
CALL LOCATE_MESSAGE ;inform user of search
CALL READ_LINE ;1 line from data file
JNC SEAl ;not EndOfFile
MOV AL,0FFH ;mark EndOfFile true
MOV [EOF].AL ;<> 0 = True
SEAl: CALL CLEAR_FILBUF ;hex 0's to file buffer
CALL CLEAR_LINE ;mark ID line
JC SE2 ;not EndOfFile
;-----locate ID number in the data file
MOF CX,3 ;loop counter
MOV DI,Offset ID ;ptr to ID number
MOV SI,Offset FilBuf ;ptr to data file line
CLD
REP CMPSB ;are the bytes = ?
JZ SE7 ;if YES exit found
XOR AL,AL ;zero AX register
CMP AL,[EOF] ;is EndOfFile TRUE?
JZ SE1 ;False = get next line
SE5: CALL NOT_FOUND ;inform user not found
G-47
; carry flag = not found
; restore registers

Input = none
Output = none

PROC NOT_FOUND
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV CL,[Color] ; store original Color
MOV AL,[Warning] ; warning color
MOV [Color].AL ; set color
MOV AX,0107h ; row 3/Col 12
CALL GOTOXY ; set cursor
CALL CSTR OUT ; display warning
db 'The ID# is not in current data file. ',0
db 'Press any key to continue. ',0
MOV [Color].CL ; restore original color
CALL HIDE CUR
CALL ERR SOUND
CALL GET_CHAR
CLC ; clear cf = continue
POP DX
POP CX
POP BX
POP AX
RET
ENDP NOT_FOUND
ENDP LOCATE

PROC ERR_SOUND
PUSH AX
MOV AX,Offset Beep
CALL SOUND
POP AX
RET
ENDP ERR_SOUND

Print a data point in the chart.
Input: AX = starting row/col of value
BX = Offset of data value in FIBuf
Output: Carry flag = abort printing
Note: this procedure is used to plot all data points "D"
DX = store hex value of point
CX = loop counter
Note: points are loaded in with the units digit in AH and tens in AL.

G-48
PROC CHART POINT
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX

;---set cursor at value "1" on the chart
  CALL HPGOTOXY
  JC CP8

;---get value from file data buffer
  MOV AX,[BX]
  CMP AX,2020h
  JZ CP7
  CMP AX,0
  JZ CP7

;---does it need rounding
  CMP BYTE PTR [BX + 2], '5'
  JC CP2

;---round the number
  CMP AH,'9'
  JZ CP1
  INC AH
  JMP SHORT CP2

CP1: MOV AH,'0'
  INC AL
  CMP AL,9
  JZ CP
  JMP SHORT CP2

CP2: AND AX,0CFCFh
  MOV DX,AX
  XOR CX,CX
  ADD CL,DL
  JZ CP7
  DEC CX
  JZ CP4
  CMP CX,5
  JNC CP7

CP3: CALL PRINT-STRING
  JC CP8
  LOOP CP3

CP4: XOR CX,CX
  ADD CL,DL
  JZ CP6
  CMP CX,10
  JNC CP7

CP5: CALL PRINT-STRING
  JC CP8
  LOOP CP5

CP6: MOV AX,Offset Point
  CALL PRINT-STRING
  JC CP8

CP7: CLC
CP8: POP DX
POP CX
POP BX
POP AX
RET
ENDP CHART_POINT

Place the cursor in the desired position in the chart.

Input = AX = Data value to position cursor in ASCII form
Output = Carry flag = abort printing
Note: this procedure moves the cursor for plotting percentiles
DX = store hex value of point
CX = loop counter
Note: points are loaded in with the units digit in AH and tens in AL

PROC POSITION_YX
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
AND AX,0CFCFh ;convert to hex
MOV DX,AX ;save number in DX
XOR CX,CX ;zero to loop counter
ADD CL,DL ;get tens digit
JZ PX5 ;if zero out of bounds
DEC CX ;beg. scale at 1 not 0
JZ PX2 ;if zero skip tens
CMP CX,5 ;is data in bounds?
JNC PX5 ;exit if out of bounds
MOV AX,Offset NextNo ;ptr to next tens str
PX1: CALL PRINT_STRING ;advance to next tens
JC PX6 ;exit on printer error
LOOP PX1 ;loop until tens = 0
PX2: XOR CX,CX ;zero loop counter
ADD CL,DH ;get units digit
JZ PX6 ;if zero OK! exit
CMP CX,10 ;is data in bounds?
JNC PX5 ;exit if out of bounds
MOV AX,Offset NextUn ;ptr to next units str
PX3: CALL PRINT_STRING ;advance to next unit
JC PX6 ;exit on printer error
LOOP PX3 ;loop until units = 0
CLC ;normal exit
JMP SHORT PX6
PX5: CALL DATA_ERROR ;mark data error
PX6: POP DX ;restore registers
POP CX
POP BX
POP AX
RET

Input = none
Output = none
PROC DATA_ERROR

G-50
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL CLEAR_MESSAGE
MOV AL,[Warning] ; warning color
MOV CL,[Color] ; save original color
MOV [Color],AL ; set color
MOV AX,0207h ; row 3/Col 8
CALL GOTOYX ; set cursor
CALL CSTR ; display warning
db ' The POSITION_YX procedure has OutOfRange data.'
db ' Press Any Key: ',0
MOV [Color],CL ; restore original color
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
CLC
POP AX
POP BX
POP CX
POP DX
RET
ENDP DATA_ERROR
ENDP POSITION_YX

--- shaded area for lowest to highest score.
Input = AX = starting row/col of value
BX = Offset of data value in PerCnt
Output = Carry flag = abort printing
Note: this procedure is used to plot percentiles
DX = store hex value of length of shading
CX = loop counter
Note: points are loaded in with the units digit in AH and tens in AL

--- place percentiles in the chart.
Input = AX = starting row/col of value
BX = Offset of data value in PerCnt
Output = Carry flag = abort printing

PROC CHART_PERCENTILES
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV DX,AX ; save row/col position
; set cursor at value "1" on the chart
CALL HPGOTOYX ; set position
JC CHIM2 ; exit on printer error
; chart the median
ADD BX,4 ; ptr median in buffer
MOV AX,[BX] ; get value from chart
G-51
CMP AX, 2020h  ; is it <spaces>?
JZ CHM1  ; if yes exit no error
CMP AX, 0  ; is it EndOfLine?
JZ CHM1  ; if yes exit no error
CALL POSITION_YX  ; set cursor in chart
JC CHM2  ; exit if error
MOV AX, Offset Median  ; ptr to median string
CALL PRINT_STRING  ; print the median
JC CHM2  ; exit if error

-----set cursor at value "1" on the chart-----
MOV AX, DX  ; restore row/col
CALL HPGOTO_YX  ; set position
JC CHM2  ; exit on printer error

-----chart the 25%-----
INC BX  ; ptr 25 percentile
INC BX
MOV AX, [BX]  ; get value from chart
CMP AX, 2020h  ; is it <spaces>?
JZ CHM1  ; if yes exit no error
CMP AX, 0  ; is it EndOfLine?
JZ CHM1  ; if yes exit no error
CALL POSITION_YX  ; set cursor in chart
JC CHM2  ; exit if error
MOV AX, Offset Left  ; ptr to Left string
CALL PRINT_STRING  ; print the median
JC CHM2  ; exit if error

-----set cursor at value "1" on the chart-----
MOV AX, DX  ; restore row/col
CALL HPGOTO_YX  ; set position
JC CHM2  ; exit on printer error

-----chart the 75%-----
INC BX  ; ptr 75 percentile
INC BX
MOV AX, [BX]  ; get value from chart
CMP AX, 2020h  ; is it <spaces>?
JZ CHM1  ; if yes exit no error
CMP AX, 0  ; is it EndOfLine?
JZ CHM1  ; if yes exit no error
CALL POSITION_YX  ; set cursor in chart
JC CHM2  ; exit if error
MOV AX, Offset Right  ; ptr to right string
CALL PRINT_STRING  ; print the median
JC CHM2  ; exit if error

CHM1: CLC
CHM2: POP DX  ; restore registers
POP CX
POP BX
POP AX
RET

ENDP CHART_PERCENTILES

G-52
PROC CHART RANGE_L
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX

;set cursor at value "1" on the chart
CALL HPGOTOYX ;set position
JNC CPT1 ;if no error continue
JMP CPT15 ;error exit

;get lowest values from file PerCnt buffer
CPT1: MOV AX,[BX] ;get lowest score
CMP AX,' ' ;is it <spaces>?  
JNZ CPT2 ;if no goto next test
JMP CPT14 ;if yes exit no error

CPT2: CMP AX,0 ;is it EndOfLine?  
JNZ CPT3 ;if no goto next test
JMP CPT14 ;if yes exit no error

CPT3: CMP AX,'01' ;is the score 1.0?  
JNZ CPT4 ;if NO jump to CPT4
MOV DX,AX ;save value in DX
MOV AX,Offset HalfSp ;ptr to 1/2 space str
CALL PRINT_STRING ;move cursor 1/2 space
JC CPT5 ;exit on printer error
MOV AX,DX ;restore value to AX

CPT4: CALL POSITION_YX ;set cursor at 25%
CPT5: JC CPT15 ;exit on printer error
AND AX,0CFCFh ;convert to hex
MOV DX,AX ;save in DX

;set high values from file PerCnt buffer
INC BX ;ptr to highest score
INC BX
MOV AX,[BX] ;get high score
CMP AX,2020h ;is it <spaces>?
JZ CPT14 ;if yes exit no error
CMP AX,0 ;is it EndOfLine?
JZ CPT14 ;if yes exit no error
CMP AX,'05' ;is It a 5.0?
JNZ CPT8 ;If no continue
MOV CX,AX ;save value in CX
CMP DH,0 ;is starting unit = 0?
JNZ CPT6 ;if not back 1/2 space
MOV AX,Offset FullBk ;ptr to full space str
JMP CPT7 ;goto print_string

CPT6: MOV AX,Offset BackSp ;ptr to 1/2 space str
CPT7: CALL PRINT_STRING ;move cursor 1/2 space
JC CPT15 ;exit on printer error
MOV AX,CX ;restore value to AX
CPT8: AND AX,0CFCFh ;convert to hex

;sub low from high (remember tens unit is in low register)
CMP AH,DH ;do I have to borrow?
JNC CPT9 ;if no ready to sub
ADD AH,10 ;add borrow to units

G-53
DEC AL
CPT9: SUB AH,DH
SUB AL,DL
JC CPT14
MOV DX,AX
XOR CX,CX
ADD CL,DL
JZ CPT11
CMP CX,5
JNC CPT14
MOV AX,offset TenLit
CPT10: CALL PRINT_STRING
JC CPT15
LOOP CPT10
CPT11: XOR CX,CX
ADD CL,DH
JZ CPT14
CMP CX,5
JNC CPT14
CMP CL,4
JC CPT12
INC CL
CMP CL,7
JC CPT12
INC CL
CPT12: MOV AX,offset UnitLit
CPT13: CALL PRINT_STRING
JC CPT15
LOOP CPT13
CPT14: CLC
CPT15: POP DX
POP CX
POP BX
POP AX
RET
ENDP CHART_RANGE_L
:
:
PROC CHART_RANGE_D
PUSH AX
PUSH BX
PUSH CX
PUSH DX
;—— set cursor at value “1” on the chart
CALL HPGOTOXY
JNC DPT1
JMP DPT15
;—— get lowest values from file PerCnt buffer
DPT1: MOV AX,[BX]
CMP AX,"
JNZ DPT2
JMP DPT14
G-54
DPT2: CMP AX,0
JNZ DPT3
JMP DPT14

DPT3: CMP AX,01'
JNZ DPT4
MOV DX,AX
MOV AX,Offset HalfSp
CALL PRINT_STRING
JC DPT5
MOV AX,DX

DPT4: CALL POSITION_YX
DPT5: JC DPT15
AND AX,0CFCFh
MOV DX,AX

DPT6: MOV AX,Offset BackSp
DPT7: CALL PRINT_STRING
JC CPT15
MOV AX,DX
XOR CX,CX
ADO CLDL
JZ DPT11
CMP CX,5
JNC DPT14
MOV AX,Offset TenDock
DPT8: AND AX,0CFCFh
DPT9: SUB AH,DH
SUB AL,DL
JC DPT10
MOV DX,AX
XOR CX,CX
ADD CL,DL
JZ DPT11
MOV AX,Offset TenDock
DPT10: CALL PRINT_STRING
JC DPT15
LOOP DPT10

is it EndOfLine?
; if no goto next test
; if yes exit no error
; is the score 1.0?
; if NO jump to DPT4
; save value in DX
; ptr to 1/2 space str
; move cursor 1/2 space
; exit on printer error
; restore value to AX
; set cursor low range
; exit on printer error
; convert to hex
; save in DX
; --- get high values from file PerCrt buffer
; ptr to highest score
INC BX
INC BX
MOV AX,[BX]
CMP AX,2020h
JZ DPT14
CMP AX,0
JZ DPT14
CMP AX,'05'
JNZ DPT8
MOV CX,AX
CMP DH,0
JNZ DPT6
MOV AX,Offset FullBk
JMP DPT7
; goto move cursor
DPT8: MOV AX,Offset TensSp
CALL PRINT_STRING
JC CPT15
; convert to hex
; --- sub low from high (remember unit is in low register)
; do I have to borrow?
; if no ready to sub
; add borrow to units
; tens = tens - 1
DPT9: SUB AH,DH
SUB AL,DL
JC DPT10
MOV DX,AX
XOR CX,CX
ADD CL,DL
JZ DPT11
MOV AX,Offset TenDock
DPT10: CALL PRINT_STRING
JC DPT15
LOOP DPT10

G-55
DPT11: XOR CX,CX    ;zero CX register
        ADD CL,DH    ;get units digit
        JZ DPT14    ;if zero skip units
        CMP CX,10    ;is data in bounds?
        JNC DPT14    ;exit if out of bounds
        CMP CL,4    ;is units 1, 2 or 3?
        JC DPT12    ;if yes draw units
        INC CL    ;add 1 to units
        CMP CL,7    ;orig unit 4, 5 or 6?
        JC DPT12    ;if yes draw units
        INC CL    ;add 1 to units
DPT12: MOV AX,Offset Unidk    ;ptr to next units str
DPT13: CALL PRINT_STRING    ;advance to next unit
        JC DPT15    ;exit on printer error
        LOOP DPT13    ;loop until units = 0
DPT14: CLC
DPT15: POP DX    ;restore registers
        POP CX
        POP BX
        POP AX
        RET
ENDP CHART_RANGE_D

;Data used by the Menu System
.DATA
;Menu0data structure
menu0 dw Offset menu0HK    ;ptr to menu HotKeys
        db 'FRPLQ',0    ;Hotkey ASCII string.
        db 05,6    ;lightbar:position in string and number of bytes
        dw Offset m01    ;pointer to lightbar message
        db 18,6
        dw Offset m02
        db 30,7
        dw Offset m03
        db 44,5
        dw Offset m04
        db 56,6
        dw Offset m05

;the menu menu0HK string contains the hot keys that will activate the Choice.
;the letters in the string should include the first letters of each menu item.
;these letters must be in the same order as the menu items. Additional
;hot keys maybe added to the string if needed. The calling program must
;be able to filter these additional HotKeys. The hot key string must
;end with a zero.
menu0HK db 'FRPLQ',0

;messages can be up to 72 character in length. The length does not have to
;be the same. The previous message is cleared before the new message is
;written. The messages can be anywhere in the data section. The numbering
;system for messages: "m01" stands for menu0 message0

G-56
Select a SLDI data file for printing.
Compute the 25th and 75th percentiles for the data.
Print SLDI Feedback reports on a HP Laser Printer.
Select the parallel port assigned to the HP Laser.
Exit the program and return to DOS.

:Menu1 data structure

<table>
<thead>
<tr>
<th>dw Offset menu1HK</th>
<th>;ptr to menu HotKeys</th>
</tr>
</thead>
<tbody>
<tr>
<td>db 'First LPT'</td>
<td>Second LPT</td>
</tr>
<tr>
<td>db 'LPT '</td>
<td>Third</td>
</tr>
<tr>
<td>db 9,11</td>
<td>;lightbar:position in string and number of bytes</td>
</tr>
<tr>
<td>dw Offset m11</td>
<td>;pointer to lightbar message</td>
</tr>
<tr>
<td>db 28,12</td>
<td></td>
</tr>
<tr>
<td>dw Offset m12</td>
<td></td>
</tr>
<tr>
<td>db 48,11</td>
<td></td>
</tr>
<tr>
<td>dw Offset m13</td>
<td></td>
</tr>
</tbody>
</table>

:Menu2 data structure

<table>
<thead>
<tr>
<th>dw Offset menu2HK</th>
<th>;ptr to menu HotKeys</th>
</tr>
</thead>
<tbody>
<tr>
<td>db 'Success Factors Failure Factors'</td>
<td>Main Menu</td>
</tr>
<tr>
<td>db 04,17</td>
<td>;lightbar:position in string and number of bytes</td>
</tr>
<tr>
<td>dw Offset m21</td>
<td>;pointer to lightbar message</td>
</tr>
<tr>
<td>db 29,17</td>
<td></td>
</tr>
<tr>
<td>dw Offset m22</td>
<td></td>
</tr>
<tr>
<td>db 54,11</td>
<td></td>
</tr>
<tr>
<td>dw Offset m23</td>
<td></td>
</tr>
</tbody>
</table>
the menu menu2HK string contains the hot keys that will activate the Choice.
the letters in the string should include the first letters of each menu item.
these letters must be in the same order as the menu items. Additional
hot keys maybe added to the string if needed. The calling program must
be able to filter these additional HotKeys. The hot key string must
end with a zero.

```
menu2HK db 'SF',0 ;Hotkey ASCII string.
```

messages can be up to 72 character in length. The length does not have to
tbe the same. The previous message is cleared before the new message is
written. The messages can be anywhere in the data section. The numbering
system for messages: "m21" stands for menu2 message1
m21 db 'Display the names of data files for positive',0
m22 db 'factors',0
m23 db 'Display the names of data files for negative',0
db 'factors',0
m23 db 'Return to the Main Menu without selecting a file.',0
end of menu2 structure

```
Menu3 data structure
```

```
menu3 dw Offset menu3HK ;ptr to menu HotKeys
db 'Single Report All Reports',0
db 11,15 ;lightbar:position in string and number of bytes
db Offset m31 ;pointer to lightbar message
db 44,13
```

;the menu menu3HK string contains the hot keys that will activate the Choice.
;the letters in the string should include the first letters of each menu item.
;these letters must be in the same order as the menu items. Additional
;hot keys maybe added to the string if needed. The calling program must
;be able to filter these additional HotKeys. The hot key string must
;end with a zero.

```
menu3HK db 'SA',0 ;Hotkey ASCII string.
```

;messages can be up to 72 character in length. The length does not have to
tbe the same. The previous message is cleared before the new message is
written. The messages can be anywhere in the data section. The numbering
system for messages: "m31" stands for menu1 message1
m31 db 'Use the three digit ID number to select a report',0
db 'to print',0
m32 db 'Print reports for all the ID numbers in the file',0
db 'data file',0
```
end of menu3 structure

```
.CODE
```

Present the Main Menu and Title screen
Input = None

G-58
Output = If critical DOS error. Error number is in AL

CH = last choice CL = max number of choices

PROC MAIN MENU
PUSH AX
PUSH BX
PUSH CX
PUSH DX

;---draw playing screen
XOR AX,AX
CALL MENU_BOX
CALL DRAW_TITLE
CALL DISPLAY_STATUS
CALL MENU_INSTRU
MOV CH,01

MA1: MOV AX, Offset Menu0
MOV CL,5
CALL GET_CHOICE

;---Is it Esc key ?
CMP AH,0h
JNZ MA2
CALL EXIT_YN
JNC MA1
JMP MA8

MA2: MOV CH,AH

;---Is Select a data file ?
DEC AH
JNZ MA3
CALL GET_PATH
JC MA2B
CALL REPORT_MENU

MA2B: XOR AX,AX
CALL MENU_BOX
JMP SHORT MA1

;---Is it Rank the variables?
MA3: DEC AH
JNZ MA4
CALL RANK_DATA
CALL DISPLAY_STATUS
JMP SHORT MA1

;---Is Print the data file?
MA4: DEC AH
JNZ MA5
CALL PRINT_MENU
JMP SHORT MA1

;---Is it choose a Laser port?
MA5: DEC AH
JNZ MA6
CALL PORT_MENU
JMP SHORT MA7

;---Is it the Exit command?
MA6: DEC AH

G-59
JNZ MA7 ;go get another choice
CALL EXIT_YN ;exit to Dos Y/N ?
JC MA8 ;if yes exit else
MA7: CALL DISPLAY_STATUS ;display status report
JMP SHORT MA1 ;get next choice
MA8: CLC ;clear cf = normal exit
MA9: POP DX
POP CX
POP BX
POP AX
RET
ENDP MAIN MENU

Print Menu for selecting a single or all reports.
Input = None
Output = If critical DOS error Error number is in AL
CH = last choice CL = max number of choices

PROC PRINT MENU
PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL IS_PRINT ;is data ready to print?
JC P14 ;exit if not ready.
MOV CH,1 ;starting menu selection
P11: MOV AX, Offset Menu3
MOV CL,2 ;max choice for menu1
CALL GET_CHOICE ;get a menu selections
;---Is it Esc key ?
CMP AH, 0h
JZ P14 ;no ready.
MOV CH,AH ;save current choice
;---Is a Single Report ?
DEC AH ;is this the choice?
JNZ P12 ;if not goto next test
CALL PRINT_ONE_REPORT ;for a single ID number
CALL MENU_INSTRU ;restore bottom box
JMP SHORT P14 ;exit procedure
;---Is it Print them All ?
P12: DEC AH
JNZ P11 ;NO = get next choice
CALL PRINT_ALL_REPORTS ;for a all ID numbers
CALL MENU_INSTRU ;restore bottom box
P14: CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP PRINT MENU

Menu for selecting the LPT output port.
Input = None
; Output = If critical DOS error  Error number is in AL  
;       CH = last choice  CL = max number of choices

PROC PORT MENU
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  MOV CX,[LPT]
  MOV CH,CL
  INC CH
  MOV AX, Offset Menu1
  CALL GET_CHOICE
  MOV CL3 ;max choice for menu1
  CALL GET_CHOICE
  MOV AX, Offset Menu2
  CALL GET_CHOICE
  MOV AX, Offset Menu3
  CALL GET_CHOICE
  MOV AX, Offset Menu4
  CALL GET_CHOICE

PROC REPORT MENU
  PUSH AX
  CALL GET_CHOICE
  MOV AX, Offset Menu5
  CALL GET_CHOICE
  MOV AX, Offset Menu6
  CALL GET_CHOICE
  MOV AX, Offset Menu7
  CALL GET_CHOICE
  MOV AX, Offset Menu8
  CALL GET_CHOICE

ENDP PORT MENU

... Menu for selecting the type of report to print.
  ... Input = None
  ... Output = If critical DOS error  Error number is in AL  
  ... CH = last choice  CL = max number of choices

PROC REPORT MENU
  PUSH AX
  CALL GET_CHOICE
  JMP SHORT M14

CLC
POP AX
RET

G-61
PUSH CX
PUSH DX
XOR AX,AX
CALL MENU BOX ; draw top menu box
CALL MENU_INSTRU ; draw bottom box
MOV CH,1 ; starting menu selection
RE1: MOV AX, Offset Menu2
MOV CL,3 ; max choice for menu2
CALL GET_CHOICE ; get a menu selection
; is it a Positive Report?
CMP AH,0h ; was <ESC> key pressed?
JZ RE4 ; if yes exit menu
MOV CH,AH ; save current choice
; is It a Dermaliment Report?
DEC AH ; is this the choice?
JNZ RE2 ; if not goto next test
MOV [Report],AH ; assign 0 to [Report]
JMP SHORT RE5 ; display files
; is It Main Menu?
RE2: DEC AH ; is this the choice?
JNZ RE3 ; if not goto next test
DEC AH ; 0 - 1 = FFh
MOV [Report],AH ; assign -1 to [Report]
JMP SHORT RE5 ; display files
RE3: DEC AH ; is this the choice?
JNZ RE1 ; if not display menu
RE4: STC ; cf = goto Main Menu
JMP SHORT RE7
RE5: CALL SELECT_FILE ; pick a file to open
CALL MENU_INSTRU
CALL DRAW TITLE ; redraw title screen
CALL DISPLAY STATUS ; display name & LPT port
CMP WORD PTR [FileHd],0 ; is a file open?
JZ RE1 ; loop if NO exit if YES
RE6: CLC ; clear cf = normal exit
RE7: POPDX
POP CX
POP BX
POP AX
RET
ENDP REPORT_MENU

---Get a Choice from the Keyboard from the menu system pointed to by AX.
Input = AX points the desired Menu data structure
CH = Starting Choice (menu item to highlight)
CL = max number of choices in this menu.
Output = AH = Choice is returned to the calling program
AL = Char from the keyboard. (Return key or Esc key)
Calls 'Display_Menu' to display the menu on the screen.
`Hot_keys` to see if Char is a HotKey for this menu

```
PROC GET_CHOICE
    PUSH BX
    PUSH CX
    PUSH DX
    MOV DX, AX
    GE0: CALL DISPLAY_MENU  ;menu to the screen
        MOV BL, CH         ;save old Choice in BL
    GE1: CALL GET_CHAR      ;get keyboard input.
        CMP AL, '1'       ;is Char < '1'
        JC GE4            ;if yes goto next test
        CMP AL, ','       ;is Char a digit ?
        JNC GE4           ;if not goto next test
        CALL NUM_LOCK     ;convert NumLock pad
    GE4: CMP AL, '4'        ;is it a right arrow?
        JNZ GE5          ;jump if not
    GE5: CMP AL, '13h'      ;is it a left arrow?
        JNZ GE6          ;jump if not
        DEC CH           ;Choice = Choice -1
    GE6: MOV CH, CL        ;set CH = maxmenu.
        JNC GE7          ;if yes then
    GE7: CMP AL, '0Dh'      ;is it a return key?
        JZ GE10        ;if yes return
        CMP AL, '1Bh'    ;is it an escape key?
        JNZ GE8         ;if no goto next test
        SUB CH, CH      ;if yes, choice = 0
        JMP SHORT GE10  ;exit; save new Choice
    GE8: CALL HOT.Keys    ;is Char a hot key?
        JC GE9           ;carry = No; next char
        CMP AH, CH      ;has Choice changed?
        JZ GE11         ;if not then exit
        MOV CH, AH      ;save new Choice
        CALL DISPLAY_MENU  ;display menu on Exit
        JMP SHORT GE11  ;exit; save new choice
    GE9: CMP CH, BL        ;new Choice - old Choice ?
        JZ GE1           ;yes = no menu display
        JMP SHORT GE0    ;no = call display menu
    GE10: MOV AH, CH       ;save choice on Exit
    GE11: POP DX
        POP CX
        POP BX
        RET

PROC NUM_LOCK
    AND AL, 0Fh          ;convert to hex
    DEC AL
    JNZ NUM_LOCK         ;if not = 1 continue
```

G-63
MOV  AL,6          ;convert to End
JMP  SHORT NML7    ;exit
NML0: DEC  AL
JNZ  NML1          ;If not = 2 continue
MOV  AL,24         ;convert to DnArrow
JMP  SHORT NML7    ;exit
NML1: DEC  AL
JNZ  NML2          ;If not = 3 continue
MOV  AL,3          ;convert to PageDn
JMP  SHORT NML7    ;exit
NML2: DEC  AL
JNZ  NML3          ;If not = 4 continue
MOV  AL,19         ;convert to LeftArrow
JMP  SHORT NML7    ;exit
NML3: DEC  AL
JNZ  NML4          ;If not = 6 continue
MOV  AL,4          ;convert to RtArrow
JMP  SHORT NML7    ;exit
NML4: DEC  AL
JNZ  NML5          ;If not = 7 continue
MOV  AL,1          ;convert to Home Key
JMP  SHORT NML7    ;exit
NML5: DEC  AL
JNZ  NML6          ;If not = 8 continue
MOV  AL,5          ;convert to UpArrow
JMP  SHORT NML7    ;exit
NML6: DEC  AL
JNZ  NML7          ;If not = 9 continue
MOV  AL,18         ;convert to PageUp
NML7: RET
ENDP NUM_LOCK

Examine the Hot Key ASCII string to find out if Char is a Hot Key.

Input = AL = Char
   CH = Choice
   CL = MaxChoice
   DX = pointer to the menu structure in data segment.
       the first word in the data structure is a pointer to
       the Hot Key ASCII string.
Output = Carry Flag  If Char in AL is not a HotKey
       AH = Choice
       AL = menu match  AL = 0Dh
           nonmenu match AL = Char

Notes: Called by GET_CHOICE. Menu data must be in an exact format.
See Menu1 data structure for an example of the correct format.

PROC HOT KEYS
PUSH BX              ;save registers
PUSH CX
PUSH DX
MOV SI,DX            ;ptr to HotKey string pointer
MOV BX,[SI]          ;load ptr to ASCII HotKey str.
G-64
AND AL, 7Fh  ;make 0 - 127 ASCII char.
CMP AL, 'a' ;is char a small letter?
JC HOT1  ;if not, Ok continue.
AND AL, 0DFh ;change to capital char
HOT1: MOV DX, AX ;save Char in DX
XOR AX, AX ;zero to AX
MOV SI, AX ;new Choice counter
MOV AL, DL ;Char returns to AL
HOT2: CMP [BX+SI], AH ;is this the End of String?
JZ HOT5 ;exit; no match found
CMP [BX+SI], AL ;is Char a Hot Key?
JZ HOT3 ;0 = found a Hot Key
INC SI ;choice = choice + 1
JMP SHORT HOT2 ;check the next Char in string.
JMP HOT4
HOT3: INC SI ;choice = choice + 1
MOV BX, SI ;choice counter to BL
MOV AH, CH ;original Choice to AH
CMP BL, CL ;is choice a menu Item?
JC HOT4 ;carry = not a menu Item
MOV AH, BL ;set new Choice
HOT4: CLC ;clear carry = HotKey found
JMP SHORT HOT6 ;Exit (found)
HOT5: MOV AH, CH ;restore original Choice
STC ;set carry flag = not HotKey
HOT6: POP DX ;restore registers
POP CX
POP BX
RET

ENDP HOT KEYS

Display menu string; highlight one menu Item; and write message string.
Input = DX pointer to the menu structure in data segment.
CH = Choice
Output = None
Notes: Called by GET CHOICE. Menu data must be in an exact format.
See Men1 data structure for an example of the correct format.

PROC DISPLAY MENU
PUSH AX  ;save registers
PUSH BX
PUSH CX
PUSH DX
INC DX  ;skip HotKey string offset
INC DX  ;ptr to beginning menu string
MOV AL, [Menu] ;menu color attribute
MOV [Color], AL  ;change color attribute
MOV AX, 0107h ;starting position for cursor
CALL GOTOYX ;place cursor
MOV AX, DX ;Offset menu to AX
CALL DSTR, OUT ;Display menu
MOV AL, [Warning] ;color for lightbar
MOV [Color], AL  ;change color attribute

G–65
SUB    AX, AX        ; zero AX register
ADD    AL, CH        ; Choice to AL
JZ     DIP1          ; abort if Choice = 0
DEC    AL            ; (Choice - 1) * 4 = offset
MOV    CL, 2         ; number of shifts to CL
SHL    AX, CL        ; shift twice = ax*4
INC    SI            ; SI points to base of table
ADD    SI, AX        ; add offset
MOV    BX, [SI]      ; get 2 bytes from table
MOV    CL, BH        ; number of char to copy
SUB    BH, BH        ; zero BH
DEC    BL            ; BX = position in menu string
MOV    AX, 0107h      ; starting cursor pos
ADD    AX, BX        ; add offset to choice
PUSH   SI            ; save SI register
CALL   GOTOYX        ; position cursor
POP    SI            ; restore SI register
MOV    AX, DX        ; start of menu string to AX
ADD    AX, BX        ; add offset to choice
MOV    DX, [SI + 2]  ; ptr to message from table.
CALL   SUB, DSTR, OUT ; highlight choice
MOV    AL, [Menu]    ; set menu color
MOV    [Color], AL   ; change color attribute
MOV    AX, 0207h      ; position cursor in 2nd row
CALL   GOTOYX        ; column 7 for message string.
MOV    BX, 024Eh      ; row 2 and column 78
CALL   CLEAR_WNDOW   ; clear out old message.
MOV    AL, [MenuMes] ; set message color
MOV    [Color], AL   ; change color attribute
MOV    AX, DX        ; pointer to message string
CALL   DSTR, OUT     ; display message string
MOV    AL, [Normal]  ; set normal color
MOV    [Color], AL   ; change color attribute
CALL   HIDE_CUR      ; hide the cursor.
DIP1:   POP   DX       ; restore registers
POP    CX
POP    BX
POP    AX
RET

ENDP   DISPLAY_MENU
ENDP   GET_CHOICE

--- Instructions for use of the menu system highlight bar.  
Input = None
Output = None

PROC   MENU INSTRU
PUSH   AX            ; save registers
PUSH   BX
PUSH   CX

G-66
PUSH DX
MOV AX,1500h ;row 21, column 0
CALL MENU BOX ;draw menu box
MOV AL,[Menu]
MOV [Color],AL
MOV AX,180Ch ;row 22, column 12
CALL GOTOYX
CALL CSTR OUT
db 'Use the ',205,' or ',205,16, 'arrow keys to '
db 'position the Highlight Bar.',0
MOV AX,1709h ;row 23, column 12
CALL GOTOYX
CALL CSTR OUT
db 'Press the <Enter> key or the first letter of the '
db 'word to proceed.',0
MOV AL,[Normal]
MOV [Color],AL
POP DX ;restore registers
POP CX
POP BX
POP AX
RET

ENDP MENU_INSTRU

;----------Display the users selections.
; Input = AX = none
; Output = none
; AX-DX register saved.

PROC DISPLAY_STATUS
PUSH AX-
PUSH BX ;save registers
PUSH CX
PUSH DX
MOV CL,[Color] ;save original color
MOV AL,[Normal]
MOV [Color],AL
MOV AX,0506h ;row/col
CALL GOTOYX
CALL CSTR OUT
db 'Data File = ',0
XOR AX,AX ;zero to AX
MOV BX,[FileHd] ;file handle
CMP BX,AX ;is the file open?
JZ FLE1 ;If no clear line
MOV AX,Offset FileNa ;ptr name of open file
CALL DSTR OUT ;send to the screen
JMP SHORT FLE2

FLE1: CALL CSTR OUT
db 'NOT Selected',0

FLE2: MOV AX,0534h
CALL GOTOYX

G-67
CALL CSTR_OUT
db "Number of ID's = ".
MOV AX,[MaxNo] ;report type
CALL BIN_OUT
MOV AX,0606h ;row/col
CALL GOTOYX
CALL CSTR_OUT
db 'Percentiles are\',0
XOR AX,AX ;zero to AX
MOV AL,[Ranked] ;report type
CMP AH,AL ;is it not selected?
JZ FLE3 ;if not goto next text
CALL CSTR_OUT ;send string to screen
db 'computed\',0
JMP SHORT FLE7 ;exit routine
FLE3: CALL CSTR_OUT ;send string to screen
db 'NOT computed\',0
FLE7: CALL ON_LINE ;is printer on line?
JNC FLE8 ;carry flag means NO
MOV AX,0634h ;row/col
CALL GOTOYX ;set cursor
CALL CSTR_OUT
db 'LPT\',0
XOR AX,AX
MOV AX,[LPT]
INC AX
CALL BIN_OUT
CALL CSTR_OUT
db 'is NOT Ready\',0
JMP SHORT FLE9
FLE8: MOV AX,0634h ;row/col
CALL GOTOYX ;set cursor
CALL CSTR_OUT
db 'LPT\',0
XOR AX,AX
MOV AX,[LPT]
INC AX
CALL BIN_OUT
CALL CSTR_OUT
db 'is Ready\',0
JMP SHORT FLE9
FLE9: MOV [Color],CL ;restore color value
POP DX
POP CX
POP BX
POP AX
RET
ENDP DISPLAY_STATUS
.DATA

.CODE

; Display the program's title on the screen
; Input = None
; Output = None
; Video mode: text row = 25 Col = 80

PROC DRAW TITLE
  PUSH AX ; save registers
  PUSH BX
  PUSH CX
  PUSH DX
  CALL CLEAR TITLE ; clear main window
  MOV AL, [Wimling]
  MOV [Color], AL
  MOV AX, 0800h
  MOV BX, 134Fh
  CALL CLEAR WINDOW
  MOV AL, [Menu]
  MOV [Color], AL
  MOV AX, 0902h
  MOV BX, 124Dh
  CALL CLEAR WINDOW
  ; draw shading
  MOV AL, [Border]
  MOV [Color], AL
  MOV CX, 8 ; line counter
  MOV BX, 0A05h ; row/col
MOV AX,Offset Shade

TIT2: XCHG AX,BX
CALL GOTOYX
XCHG AX,BX
CALL DSTR_OUT
MOV AX,SI
INC AX
INC BH
LOOP TIT2

--draw title
MOV AL,[Normal]
MOV [Color],AL
MOV CX,7
MOV BX,0A40h
MOV AX,Offset TKey
TIT1: XCHG AX,BX
CALL GOTOYX
XCHG AX,BX
CALL TITLE_OUT
MOV AX,SI
INC AX
INC BH
LOOP TIT1

TIT3: POP DX
POP CX
POP BX
POP AX
RET

ENDP DRAW_TITLE

;--Clear the main display window in the EditKey view program.
; Input = None
; Output = None
; 16 colors row = 25 Col = 80

PROC CLEAR_TITLE
PUSH AX
PUSH BX
PUSH CX
MOV CL,[Color]
MOV AL,[Normal]
MOV [Color],AL
MOV AX,0400h
MOV BX,144Fh
CALL CLEAR_WINDOW
MOV [Color],CL
POP CX
POP BX
POP AX
RET

ENDP CLEAR_TITLE

G-70
Send an ASCII string to screen and skip all <space> but advance the cursor for each space.

Input = AX must point to the string. The string must end with a hex zero. The desired color attribute must be defined in the data segment.

Output = None. All registers are saved except SI.

PROC TITLE_OUT
    PUSH AX                      ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV SI,AX                    ;pointer to string.
    MOV BH,0                     ;page 0 assumed
    MOV CX,1                     ;from data segment.
    MOV BL,[Color]               ;load color attribute
    TITL1: MOV AL,[SI]           ;get char from string
        CMP AL,0                   ;is it the end?
        JZ TITL2                  ;exit if end of string.
        CMP AL,20h                ;is it a space?
        JNZ TITL3                 ;if space skip?
        CALL INC CURSOR
        JMP SHORT TITL4
    TITL3: MOV AL,20h             ;write a space
        MOV DL,AL                 ;store char in DL
        MOV AX,0920h              ;write 1 space to
        PUSH SI                    ;save SI register
        INT 10h                    ;set color attribute
        MOV AH,0Eh                 ;fun.no. teletype mode
        MOV AL,DL                  ;get char for DL reg.
        INT 10h                    ;char to the console
        POP SI                     ;restore SI register
    TITL4: INC SI                 ;point to next char
        JMP SHORT TITL1           ;get next character.
    TITL2: POP DX                 ;restore registers.
        POP CX
        POP BX
        POP AX
    RET
ENDP TITLE_OUT

.CODE

Set the colors variables for the video mode.

Input: ES is a ptr to the (PSP) Program Segment Prefix
       (when DOS programs are loaded the ES register points to PSP)
Output: None

G-71
PROC COLOR MODE
  MOV DI, 80h ; offset to len COM tail
  XOR AX, AX ; zero register
  ADD AL, [ES: DI] ; get len Com tail
  JZ VID2 ; jump if no parameters
  MOV CX, AX ; loop counter
  VID0: INC DI ; ptr to next byte
  MOV AL, [ES: DI] ; is it the marker?
  AND AL, 5Fh ; make a capital letter
  CMP AL, 'M' ; is it the mono par?
  JZ VID1 ; if Yes jump to Mono
  LOOP VID0 ; look through COM tail
  JMP SHORT VID2 ; Not found get dis mode
  VID1: CALL MONO VIDEO ; set color variables
  JMP SHORT VID5 ; exit
  VID2: XOR AX, AX ; get display mode
  INT 10h ; BIOS function.
  CMP AL, 7 ; is it Text-Mono?
  JNZ VID3 ; No = jmp next test
  CALL MONO VIDEO ; set color variables
  JMP SHORT VID5 ; exit
  VID3: CMP AL, 0Fh ; is it Graph-Mono?
  JNZ VID4 ; No = jmp next test
  CALL MONO VIDEO ; set color variables
  JMP SHORT VID5 ; exit
  VID4: CALL COLOR_VIDEO ; set color variables
  VID5: RET

PROC COLOR_VIDEO
  MOV BX, Offset Menu
  MOV AL, 71h
  MOV [BX], AL
  INC BX
  MOV AL, 1Fh
  MOV [BX], AL
  INC BX
  MOV AL, 1Eh
  MOV [BX], AL
  INC BX
  MOV AL, 7Fh
  MOV [BX], AL
  INC BX
  MOV AL, 4Fh
  MOV [BX], AL
  INC BX
  MOV AL, 070h
  MOV [BX], AL
  RET

ENDP COLOR_VIDEO
PROC MONO_VIDEO
  MOV BX, Offset Menu

G-72
MOV AL,70h ;Menu = Black/White
MOV [BX],AL
INC BX
MOV AL,0Fh ;Normal = White/Black
MOV [BX],AL
INC BX
MOV AL,0Fh ;HiLite = White/Black
MOV [BX],AL
INC BX
MOV AL,70h ;MenuMes = Black/White
MOV [BX],AL
INC BX
MOV AL,0Fh ;Warning = White/Black
MOV [BX],AL
INC BX
MOV AL,7Fh ;Border = White/Lt White
MOV [BX],AL
RET
ENDP MONO_VIDEO
ENDP COLOR_MODE

Save the current user's video information to be restored by RESTORE_VIDEO
set text video mode for this program.

Input = None
Output = set variables: [vidmode],[vidpage],[vidcurs],[vidfont]
[vidattr] and [vidbord]
Note: has no effect if the DOS version is less than 3.30.

PROC TEXT_VIDEO
;test for DOS 3.3 or greater
MOV AH,30h ;get DOS ver number
INT 21h
XCHG AH,AL ;high byte to ah
CMP AX,031Eh ;is DOS > = 3.30 ?
JNC SV0 ;if yes continue else
JMP SV5 ;exit assume text mode

;get video mode
SV0: MOV AH,0Fh ;get video mode
INT 10h
MOV [vidmode],AL ;save videomode
MOV [vidpage],BH ;save videopage

;get cursor information
MOV AH,03h ;get cursor status
INT 10h
MOV [vidcurs],CX ;save cursor shape

;get font size
MOV AX,1130h ;get font information
XOR BH,BH ;0 = current font
INT 10h
MOV AX,1112h ;assume small font
CMP CX,8 ;is it 8x8 font ?
JZ SV1
MOV AX,1114h
CMP CX,16
JZ SV1
MOV AX,1111h

SV1: MOV [vidfont],AX

;get current color attributes
MOV AH,08h
MOV BH,[vidpage]
INT 10h
MOV [.v.attr],AH
MOV CL,4
SHR AH,CL
MOV [vidfont],AH

;is this a VGA system ?
MOV AX,1A00h
INT 10h
CMP AL,1Ah
JNZ SV2
MOV AX,1008h
INT 10h
MOV [vidfont],AH

;set font type
MOV AX,1114h
XOR BL,08h
INT 10h

;set text video mode
SV2: MOV AX,0003
MOV BL,[vidmode]
INT 10h
CMP BL,7
JZ SV5
INT 10h

SV5: MOV AL,[Normal]
MOV [Color],AL
CALL CLEAR_SCREEN
CLC
RET

ENDP TEXT_VIDEO

; Restore the users video information which was save by SAVE_VIDEO
; when the program began.

; Input = None
; Output = Clears the screen
; Note: uses variables: [vidmode],[vidpage],[vidcursor],[vidfont]
; [vidattr] and [vidbord]

PROC RESTORE_VIDEO
PUSH ES

; test for DOS 3.3 or greater
MOV AH,30h
;get dos ver number

G-74
INT 21h
XCHG AH,AL     ;high byte to ah
CMP AX,031Eh    ;is dos >= 3.30 ?
JC REV2         ;default to w/b

;--- restore original video mode
MOV AL,[vidmode] ;get video mode no.
XOR AH,AH
INT 10h

;--- return display page to 0
MOV AX,0500h
INT 10h

;--- restore original font size
MOV AX,[vidfont]
XOR BL,BL
INT 10h

;--- read cursor configuration
MOV AH,03h
XOR BH,BH
INT 10h

;--- restore original video page
MOV AL,[vidpage]
MOV AH,5h
INT 10h

;--- restore cursor shape
MOV CX,[vidcurs]
MOV AH,01h
INT 10h

;--- set border color
MOV AX,1001h
MOV BH,[vidbord]
INT 10h

;--- clear the screen if dos 3.3 or greater
MOV AX,0600h     ;scroll & clear window
MOV BH,[vidattr]  ;get color attribute
XOR CX,CX        ;start row 0, col 0
MOV DX,40h       ;bios data area
MOV ES:4Ah        ;get number of columns
DEC DX           ;convert to 0 start
MOV ES:84h        ;get number of rows
INT 10h          ;clear whole screen
JMP SHORT REV3

;--- clear screen if not dos 3.3 or greater
REV2: MOV AL,[vidattr] ;system color W/B
MOV [Color],AL   ;set color
CALL CLEAR_SCREEN ;blank the screen

REV3: CLC          ;clear carry flag
POP ES
RET

ENDP RESTORE VIDEO
Open a Disk File using the file Handle method.

Input = AX = ptr ASCII name of the file.
shared/read/write access assumed.
Output = Carry flag set if an opening error or
file size in AX and DX; File Handle in BX.
is set to beginning of the file.
Note: Registers are not saved.

PROC OPEN
MOV DX,AX
;ptr to file name string
MOV AH,3Dh
;open file with handle
MOV AL,42h
;shared/read/write mode
INT 21h
;try to open file.
JC OP1
;carry = opening error
MOV BX,AX
;file handle in BX
XOR AX,AX
;zero AX
MOV CX,AX
;off set from EOF
MOV DX,AX
; = 0 in CX AND DX.
MOV AX,4202h
;position at EOF
INT 21h
;size of file in
PUSH AX
;bytes returns in
PUSH DX
;AX and DX.
XOR AX,AX
;zero AX
MOV CX,AX
;off set from BOF
MOV DX,AX
; = 0 in CX AND DX.
MOV AX,4200h
;position file pointer
INT 21h
;file ptr to BOF
POP DX
;size of file returns
POP AX
;in AX and DX regs.
CLC
;clear carry flag
JMP SHORT OP4
;return to calling prg.
OP1:
CMP AX,0Ch
;is access code wrong?
JNZ OP2
;if not skip.
XOR AX,AX
;zero AX if wrong code
OP2:
CMP AX,6
;is error code > 5?
JC OP3
;if not skip.
MOV AX,6
;end of error table
OP3:
MOV [ErrCode],AL
;save error code
SHL AX,1
;multi err code by 2
MOV BX,Offset OpenErr
;open error table base
ADD BX,AX
;add err. offset to base
CALL CLEAR_MESSAGE
;warning color
MOV CH,[Warning]
;save original color
MOV CL,[Color]
;set color
MOV [Color],CH
MOV AX,0207h
;row 3/Col 8
CALL GOTOXY
;set cursor
CALL CSTR OUT
;output this line to
db 'Error opening ',0
;the screen.
MOV AX,DX
;File name pointer.
CALL DSTR OUT
;output file name.
MOV AX,[BX]
;ptr msg string to AX
ENDP
CALL DSTR_OUT ;display type of error
MOV [Color].CL ;restore original color
CALL HIDE CUR
CALL ERR "SOUND" ;to stop program
STC ;set carry flag
OP4: RET
ENDP OPEN

.DATA
OpE1 db ': Invalid access code. ',0
OpE2 db ': Invalid function. ',0
OpE3 db ': File not found. ',0
OpE4 db ': Path not found. ',0
OpE5 db ': No handles available. ',0
OpE6 db ': Access denied. ',0
OpE7 db ': Error code beyond table. ',0

.CODE

;---Send a ASCII String of a given length in the Data Seg. to the console.
; Input = AX must points to the first character to send in the string.
; CL = number of bytes to send
; Output = None. AX-DX registers saved.

PROC SUB DSTR_OUT

PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV SI,AX ;pointer to string.
MOV DL,CL ;DL = number of chars
MOV BH,0 ;page 0 assumed
MOV BL,[Color] ;load color attribute
MOV CX,1 ;from data segment.

DSTR3: MOV AL,[SI] ;get char from string
CMP AL,0 ;Is it the end ?
JZ DSTR4 ;exit if end of string.
MOV DH,AL ;store char in DH
MOV AX,0920h ;write 1 space to
PUSH SI ;save SI register
INT 10h ;set color attribute
MOV AH,0Eh ;fun. no. teletype mode
MOV AL,DH ;get char for DH reg.
INT 10h ;char to the console
POP SI ;restore SI register
INC SI ;point to next char
DEC DL ;character counter
JNZ SHORT DSTR3 ;get next character.

DSTR4: POP DX ;restore registers.

G-77
POP BX
POP AX
RET
ENDP SUB_DISTR_OUT

;---Position the cursor on the screen
; Input = AH (row) AL (column) position in binary numbers.
; Output = none. All registers restored.
; Notes: upper left hand corner = 0,0
; page 0, 25 rows and 80 columns screen assumed.
; Calling with DH = 25 will hide the cursor off screen!!!
PROC GOTOXY
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV DX,AX
CMP DH,25 ;is row > 25 ?
JC @@LOC1 ;if yes default to 0
MOV DH,0 ;set row to top line
@@LOC1: CMP DL,80 ;is column > 79 ?
JC @@LOC2 ;if yes default to 0
MOV DL,0 ;column to far right
@@LOC2: MOV AH,02h ;set cursor funct, no.
     MOV BH,0 ;page 0 assumed
     INT 10h ;position cursor
     POP DX ;restore registers
     POP CX
     POP BX
     POP AX
     RET

;---Advance cursor one column on the screen
; Input = none
; Output = none. All registers restored.
; page 0, 25 rows and 80 columns screen assumed.
PROC INC_CURSOR
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,0300h
MOV BH,AL
INT 10h ;position cursor
CMP DL,79
JZ INC1
INC DL
JMP SHORT INC2
INC1: INC DH
INC2: MOV AX,0200h
INT 10h

G-78
```assembly
POP DX ; restore registers
POP CX
POP BX
POP AX
RET
ENDP INC_CURSOR

--- Hide cursor at row 25, column 0 below the last line of the screen.
Input = None
Output = None
Calls GOTOYX
Notes: Page 0 and 25 line text screen assumed.

PROC HIDE_CUR
    MOV AX,1900h ; row = 25 col = 0
    CALL GOTOYX ; place cursor
    RET
ENDP HIDE_CUR
ENDP GOTOYX

--- Send an ASCII string in the Data Segment to the console.
Input = AX must point to the string. The string must end with a hex zero. The desired color attribute must be defined in the data segment.
Output = None. All register are saved except SI.

PROC DSTR_OUT
    PUSH AX ; save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV SI,AX ; pointer to string.
    MOV BH,0
    MOV AX,@DATA
    PUSH DS
    MOV DS,AX
    MOV BL,[Color] ; load color attribute
    POP DS
    MOV CX,1 ; from data segment.
    DSTR1: MOV AL,[SI] ; get char from string
        CMP AL,0 ; is it the end?
        JZ DSTR2 ; exit if end of string.
        MOV DL,AL ; store char in DL
        MOV AX,0820h ; write 1 space to
        PUSH SI ; save SI register
        INT 10h ; set color attribute
        MOV AH,0Eh ; ?fun. no. teletype mode
        MOV AL,DL ; get char for DL reg.
        INT 10h ; char to the console
        POP SI ; restore SI register
    DSTR2: POP AX
```
INC SI ;point to next char
JMP SHORT DSTR1 ;get next character.
DSTR2: POP DX
POP CX
POP BX
POP AX
RET
ENDP DSTR_OUT

---Send a two byte unsigned binary number to the screen in decimal form.
Input = binary number in AX
Output = decimal number to the screen. Registers restored on return.
Note: this recursive procedure could use up to 40 bytes of stack space.
leading zeros are suppressed and no space padding is used.
BIN_OUT and DIGIT_OUT must be NEAR procedures.
PROC BIN_OUT NEAR
PUSH AX ;save dividend
PUSH BX
PUSH CX ;save CX register
PUSH DX ;save remainder
SUB DX,DX ;zero DX register
MOV CX,0Ah ;divisor is 10
DIV CX ;AX/10; answer in AX
CMP AX,0 ;remainder digit in DL
JZ @@BIN ;if yes stop recursion
CALL BIN_OUT ;continue recursive call
@@BIN: CALL DIGIT_OUT ;display digit in DL
POP DX ;previous digit to DL
POP CX ;restore CX register
POP BX
POP AX ;restore AX register
RET ;NOTE: this RET will point to @@BIN: to display
;each digit of the recursions stored in DL register.
;After all digits are displayed it will return to
;the calling program.

---Send a digit (0 to 9) stored in DL register to the screen
PROC DIGIT_OUT NEAR
MOV BH,0 ;page 0 assumed
MOV BL,[Color] ;load color attribute
MOV CX,1 ;from data segment.
MOV AX,0920h ;write 1 space to
INT 10h ;set color attribute
MOV AH,0Eh ;fun.no. teletype mode
MOV AL,DL ;get char for DL reg.
OR AL,30h ;convert to ASCII digit
INT 10h ;char to the console
RET
ENDP DIGIT_OUT
ENDP BIN_OUT

;---Get a Char from the keyboard. (keyboard buffer not cleared before input)
**PROC GET_TEXT**

GET1: MOV AX, 0700h

INT 21h

CLC

CMP AL, 0

JNZ @@TEXT

MOV AH, 07h

INT 21h

CALL EXT_CHAR

JC GET1

STC

@@TEXT: MOV AH, 0

RET

ENDP GET_TEXT

---

Get a Char from the Standard Input device. (keyboard assumed)

Input = none

Output = binary ASCII keyboard code in AX

Carry flag = extended code.

**PROC GET_CHAR**

GET0: MOV AX, 0C07h

INT 21h

CLC

CMP AL, 0

JNZ @@CHAR

MOV AH, 07h

INT 21h

CALL EXT_CHAR

JC GET0

STC

@@CHAR: MOV AH, 0

RET

---

A subroutine to convert extended codes to control codes.

Input = extended code in AL

Output = converted code in AL

Carry flag if not one of the Keys listed below:

<table>
<thead>
<tr>
<th>Key</th>
<th>Extended Code</th>
<th>Converted to:</th>
<th>Ctrl-Char</th>
<th>Ctrl-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>47h</td>
<td>^A</td>
<td>1h</td>
<td></td>
</tr>
<tr>
<td>UpArr</td>
<td>48h</td>
<td>^E</td>
<td>5h</td>
<td></td>
</tr>
<tr>
<td>PgUp</td>
<td>49h</td>
<td>^R</td>
<td>12h</td>
<td></td>
</tr>
<tr>
<td>LArr</td>
<td>48h</td>
<td>~S</td>
<td>13h</td>
<td></td>
</tr>
<tr>
<td>RArr</td>
<td>49h</td>
<td>~D</td>
<td>4h</td>
<td></td>
</tr>
<tr>
<td>End</td>
<td>4Fh</td>
<td>^F</td>
<td>6h</td>
<td></td>
</tr>
<tr>
<td>DnArr</td>
<td>50h</td>
<td>~X</td>
<td>18h</td>
<td></td>
</tr>
<tr>
<td>PgDn</td>
<td>51h</td>
<td>~C</td>
<td>3h</td>
<td></td>
</tr>
<tr>
<td>Ins</td>
<td>52h</td>
<td>~V</td>
<td>16h</td>
<td></td>
</tr>
</tbody>
</table>

G-81
; Del 53h
PROC EXT_CHAR
    CMP AL,47h
    JNZ EXT0
    MOV AL,1
    JMP SHORT EXT10
EXT0: CMP AL,48h
    JNZ EXT1
    MOV AL,5
    JMP SHORT EXT10
EXT1: CMP AL,49h
    JNZ EXT2
    MOV AL,12h
    JMP SHORT EXT10
EXT2: CMP AL,4Ah
    JNZ EXT3
    MOV AL,13h
    JMP SHORT EXT10
EXT3: CMP AL,4Dh
    JNZ EXT4
    MOV AL,4
    JMP SHORT EXT10
EXT4: CMP AL,4Fh
    JNZ EXT5
    MOV AL,6
    JMP SHORT EXT10
EXT5: CMP AL,50h
    JNZ EXT6
    MOV AL,18h
    JMP SHORT EXT10
EXT6: CMP AL,51h
    JNZ EXT7
    MOV AL,3
    JMP SHORT EXT10
EXT7: CMP AL,52h
    JNZ EXT8
    MOV AL,16h
    JMP SHORT EXT10
EXT8: CMP AL,53h
    JNZ EXT9
    MOV AL,7
    JMP SHORT EXT10
EXT9: STC
    JMP SHORT EXT11 ; set carry flag
EXT10: CLC
    JMP SHORT EXT11 ; clear carry flag
EXT11: RET
ENDP EXT_CHAR
ENDP GET_CHAR

;———Send an ASCII string in the Code segment to the console.
The call must be right before the string. The string must end with a hex zero. This procedure must be called as a near procedure.
The desired [color] attribute must be stored in the Data segment.

Note: All registers save except SI.

PROC CSTR_OUT NEAR
POP SI
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV BH,0 ;page 0 assumed
MOV BL,[Color] ;load color attribute
MOV CX,1 ;from data segment.
CSTR1: MOV AL,[CS:SI] ;get char from code seg
CMP AL,0
JZ CSTR2 ;0 = end of string
MOV DL,AL ;store char in DL
MOV AX,0920h ;write 1 space to
PUSH SI ;save SI register
INT 10h ;set color attribute
MOV AH,0Eh ;func.no.teletype mode
MOV AL,DL ;get char for DL
INT 10h ;char to the console
POP SI ;restore SI register.
INC SI ;point to next char
JMP SHORT CSTR1 ;get next character.
CSTR2: INC SI ;SI points to next
POP DX ;instruction in code.
POP CX ;restore registers.
POP BX
POP AX
PUSH SI ;store the CS pointer.
RET ;restore CS register.
ENDP CSTR_OUT

---Clear a Window
Input = AX = upperleft corner row/col row 0 - 24
BX = lower right corner row/col col 0 - 79
[color] = current attribute from data section
page 0 assumed.
Output = abort if row or column are out of bounds.

PROC CLEAR_WINDOW
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
CMP BH,AH ;is starting row>ending?
J C WIN1 ;exit if yes.
CMP BL,AL ;is starting col>ending?
J C WIN1 ;exit if yes.
CMP BH,25 ;is row out of bounds?
J N C WIN1 ;exit if yes.

G-83
CMP BL,80 ; is col out of bounds?
JNC WIN1 ; exit if yes.
MOV CX,AX ; starting row/col to CX
MOV DX,BX ; ending row/col to DX
MOV AX,0800h ; window function no
MOV BH,[Color] ; get active color
INT 10h ; clear the window
WIN1: POP DX ; restore registers
    POP CX
    POP BX
    POP AX
    RET

ENDP CLEAR_WINDOW

---Draw a 17 line 80 column display box for the Restaurant program.
It can also be used to clear the display screen and title screen.
Input = None
Output = None
Calls CSTR_OUT procedure

PROC DISPLAY_BOX
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    MOV AL,[Border] ; change color attribute
    MOV [Color],AL ; for screen output
    MOV AX,0400h ; row 4, column 0
    CALL GOTOYX ; set cursor position
    CALL CSTR_OUT ; draw box
    db 201, 78 DUP (205),187
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 186, 78 DUP ('),186
    db 200, 78 DUP (206),186,0
    MOV AL,[Normal] ;
    MOV [Color],AL
    POP DX
    POP CX
    POP BX
    POP AX
Display an error message on the screen in row 5 column 3. Normal colors
Input = AX pointer to ASCIIZ string in Data segment
Output = Carry Flag set
Note: sends message to screen and wait for key to be pressed.

PROC ERROR_MESSAGE
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV CL,[Color] ;save assigned color
MOV AL,[Normal] ;set color to normal
MOV [Color],AL ;for string output
MOV AX,403h ;point row 4, col 3
CALL GOTOXY ;position cursor
MOV AX,BX ;load string pointer
CALL DSTR OUT ;display error message
MOV [Color],CL ;restore assigned color
CALL HIDE CUR ;hide the cursor
CALL GET_CHAR ;wait until key pressed
STC ;carry flag = error
POP DX
POP CX
POP BX
POP AX
RET
ENDP ERROR_MESSAGE

Play a series of notes using the 8253 timer chip to produce sound.
Input AX = pointer to 16 bit data string containing
    frequency and duration for each pitch.
    data string must end with a Hex 0
Output None

PROC SOUND
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
PUSH BP
MOV SI,AX ;place data ptr in SI
IN AL,61h ;get status port B
OR AL,3 ;enable speaker and
OUT 61h,AL ;timer channel 2.
MOV AL,0B6h ;initialize channel 2
OUT 43h,AL ;for mode 3
SOU1: MOV DX,[SI] ;load frequency
CMP DX,0 ;is it the end of str?
JZ SOU3 ;if yes exit else
G-85
INC SI ; advance ptr to
INC SI ; point to the duration
MOV AL,DL ; low lab of frequency
OUT 42h,AL ; send to latch2 port
MOV AL,DH ; low msb of frequency
OUT 42h,AL ; send to latch2 port
MOV AH,0 ; int function number
INT 1Ah ; get BIOS timer count
MOV BX,DX ; move leoword to BX
ADD BX,[SI] ; add duration to BX
INC SI ; advance ptr to
INC SI ; point to next frequency.

SOI2: INT 1Ah ; get BIOS timer count
CMP DX,BX ; is count > duration?
JC SOU1 ; if not check again else
JMP SHORT SOU1 ; jump to get next freq.

SOU3: IN AL,61h ; get byte from port B
AND AL,0FCh ; turn off speaker bits
OUT 61h,AL ; replace byte in port B
MOV DX,1282 ; default setting
MOV AL,DL ; get lab of count
OUT 42h,AL ; send to port 42h
MOV AL,DH ; get msb of count
OUT 42h,AL ; send to port 42h
POP BP ; restore registers
POP CX
POP BX
POP AX
RET

ENDP SOUND

Force the numlock key ON by turning on bit 5 in the BIOS data area

PROC NUM_LOCK_ON
PUSH AX
PUSH DS
XOR AX,AX
MOV DS,AX
MOV AL,20h
MOV SI,417h
OR [SI],AL
POP DS
POP AX
RET

ENDP NUM_LOCK_ON

--- Clear Display Message.
Input = None
Output = None

PROC CLEAR_MESSAGE

G-86
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AL,[Menu] ;Menu color
MOV [Color],AL ;set color
MOV AX,0200h ;row 3; col 8
CALL GOTOYX ;position cursor
CALL CSTR_OUT
db 73 DUP(2h),0
MOV AL,[Normal] ;normal color
MOV [Color],AL ;set color
CALL HIDE_CUR
POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP CLEAR-MESSAGE

--- Create a Disk File using the file Handle method.

Input = AX = ptr ASCIIZ name of the file.
shared/read/write access.
Output = Carry flag set if a creating error or
File Handle in BX and a file of 0 bytes is open

**** Caution: This procedure will erase and existing file. ********
If the two files have the same name.

PROC CREATE
MOV DX,AX ;ptr to file name string
MOV AH,3Ch ;create file with handle
XOR CX,CX ;normal attributes
INT 21h ;try to open file.
JC CT1 ;carry = opening error
MOV BX,AX ;file handle in BX
CLC
JMP SHORT CT4 ;return to calling prg.
CT1: CMP AX,0Ch ;is access code wrong?
JNZ CT2 ;if not skip.
XOR AX,AX ;zero AX if wrong code
CT2: CMP AX,6 ;is error code > 5 ?
JC CT3 ;if not skip.
MOV AX,8 ;end of error table
CT3: MOV [ErrCode],AL ;save error code
SHL AX,1 ;multi err code by 2
MOV BX,Offset OpenErr ;open error table base
ADD BX,AX ;add err. offset to base
MOV AX,0107h ;row 2, column 8
CALL GOTOYX ;position cursor

G-87
MOV CL,[Color]; save color attribute
MOV AL,[Warning]; get new attribute
MOV [Color].AL; assign attr to color
CALL CSTR_OUT ; output this line to screen
DB 'Error creating ',0 ; File name pointer.
MOV AX,DX; output file name.
CALL DSTR_OUT ; ptr mag string to AX
MOV AX,[BX]; display type of error
MOV [Color].CL ; restore original attr
STC ; set carry flag
CT4: RET
ENDP CREATE

;Send a 16 bit unsigned binary number to the screen in decimal form
; in EGA and VGA Graphics Mode 10h page 0
; Input = AX = binary number  CX = total number of digits
; the number is padded with leading zeros until CX digits
; [color] = current attribute from data section
; Output = None
; Video mode: 10h 640 x 350 16 colors  row = 25 Col = 80

Note: the calling procedure must make sure that the number is CX is
large enough to display all the digits of the number in AX. This
procedure can be used when leading zeros are needed.

PROC BIN_DIG_OUT NEAR
PUSH AX; save dividend
PUSH BX
PUSH CX; save CX register
PUSH DX; save remainder
SUB DX,DX; zero DX register
MOV BX,0Ah; divisor is 10
DIV BX; AX/10; answer in AX
DEC CX; remainder digit in DL
JZ @BIN; if yes stop recursion
CALL BIN_DIG_OUT; continue recursive call
@BIN: CALL DIGIT_OUT ; display digit in DL.
PUSH DX; previous digit to DL
POP DX
POPCX; restore CX register
POP BX
POP AX; restore AX register
RET; NOTE: this RET will point to @@BIN: to display
each digit of the recursions stored in DL register.
; After all digits are displayed it will return to
; the calling program.

; Ask a yes or no question.
; Input = None
; Output = Carry Flag = YES

G-88
PROC EXIT_YN
    PUSH AX ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLEAR MESSAGE ;warning color
    MOV AL,[Warning] ;row 3/Col 12
    MOV AX,0208h
    CALL GOTOYX ;display warning
db 'Exit to DOS? [Y/N]',0
    MOV AL,[Normal] ;normal color
    MOV [Color],AL
    CALL CLEAR MESSAGE ;set color

    EX1: CALL HIDE CUR
    CALL GET_CHAR
    AND AL,5Fh ;turn off bits 6 & 8
    CMP AL,'N' ;is it No?
    JZ EX4 ;if yes exit
    CMP AL,0Dh ;is it <Enter>?
    JNZ EX2 ;if not continue
    STC ;set carry flag
    JMP SHORT EX5 ;exit

    EX2: CMP AL,'Y' ;is it Yes?
    JNZ EX1 ;if not get another
    STC ;set carry flag
    JMP SHORT EX5 ;exit

    EX4: CLC ;clear carry flag
    EX5: POP DX ;restore registers
    POP CX
    POP BX
    POP AX
    RET

ENDP EXIT_YN

---Clear the screen and place the cursor in position 0,0
Input = None  Color = current [color] attribute from data section
Output = None  Border color is also set the same as the screen.
Notes: All registers saved. 25 rows and 80 columns page 0 assumed.

PROC CLEAR_SCREEN
    PUSH AX ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV BH,[Color] ;color attribute
    MOV AX,0700h ;initialize window func
    SUB CX,CX ;row/col = 0,0
    MOV DX,184Fh ;row/col = 24,79
    INT 10h ;clear screen window

G-89
MOV     BH, [Color] ; color attribute
MOV     CL, 4 ; shift background color
SHR     BH, CL ; to the lower 4 bytes.
MOV     AX, 1001h ; function number
INT     10h ; set screen border
MOV     AH, 2h ; set cursor position
MOV     BH, 0 ; page 0, row, col to DX
MOV     DX, CX ; position cursor at the
INT     10h ; the top left corner.
POP     DX ; restore registers
POP     CX
POP     BX
POP     AX
RET
ENDP   CLEAR_SCREEN

--- Draw a 4 line 80 column menu box. Starting at row 0-21, column 0.
Input = AX = Row, Columns cursor position. Column must be 0
Output = None
Calls CSTR_OUT procedures
PROC MENU_BOX
PUSH    AX
PUSH    BX
PUSH    CX
PUSH    DX
XOR     AL, AL ; set column to 0
CALL    GOTOXY ; set cursor position
MOV     DL, [Color] ; save original Color
MOV     AL, [Menu] ; change color attribute
MOV     [Color], AL ; for screen output.
CALL    CSTR_OUT ; draw menu box
db     201, 78 DUP (205), 187 ; all except last byte.
db     186, 78 DUP (" "), 186 ; CSTRU_OUT will cause
db     186, 78 DUP (" "), 186 ; the screen to scroll
db     200, 78 DUP (205), 0 ; in row 25, col 80.
MOV     BL, AL ; color to BL
MOV     AH, 09h ; write char func no.
MOV     AL, 188 ; last character of box
MOV     BH, 0 ; page 0 assumed
MOV     CX, 1 ; number of bytes
INT     10h ; write last byte
MOV     [Color], DL ; restore original Color
POP     DX
POP     CX
POP     BX
POP     AX
RET
ENDP   MENU_BOX

--- Draw a 4 line 80 column menu box. Starting at row 0-21, column 0.

G-90
Input = AX = Row, Columns cursor position. Column must be 0
Output = None
Calls CSTR OUT procedures

PROC CLEAR_BOX
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    MOV DL,[Color] ; save original Color
    MOV AL,[Normal] ; change color attribute
    MOV [Color],AL ; for screen output.
    XOR AL,AL ; set column to 0
    MOV BL,79
    MOV BH,AH
    ADD BH,4
    CALL CLEAR_WINDOW
    MOV [Color],DL ; restore original Color
    POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP CLEAR_BOX

; Display the DOS extended error message return
; by calling Int 21h function 59h - Get extended error information. If the
; error code is less than 36 the error string is presented. If the error
; number is 36 or larger the number is print to the screen.
; Input = None   data bytes [ErrCode],[Color], and [Normal] assumed.
; Output = Error number in [ErrCode]

PROC DISPLAY_ERROR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH ES
    MOV [ErrCode],AL ; save AL register
    CALL CLEAR_MESSAGE
    MOV CH,[Warning] ; warning color
    MOV CL,[Color] ; save original color
    MOV [Color],CH ; set color
    MOV AX,0207h ; row 3/Col 8
    CALL GOTOYX ; set cursor
    ; request extended error information
    XOR BX,BX ; get extended error
    MOV AX,5900h ; information from
    INT 21h ; DOS system
    XOR AH,AH ; zero hi byte
    CMP AL,0 ; was an error found?
    JZ ERROR1 ; if NO display message
CMP AL, 37 ; is code < 37 ?
JC ERROR1 ; if YES display mess
CALL CSTR OUT
db 'DOS Error Number: ', 0
CALL BIN OUT
MOV AX, 37 ; 37 = unknown DOS error
ERROR1: SHL AX, 1 ; error number x 2
MOV BX, Offset ErrStr
ADD BX, AX
CALL CSTR_OUT
db 'DOS Error: ', 0
MOV AX, [BX] ; load ptr to error str.
CALL DSTR_OUT
CALL CSTR_OUT
db ' Press Any Key. ', 0 ; send string to screen
MOV [Color], CL ; restore original color
CALL HIDE CUR
CALL ERR SOUND
CALL GET_CHAR
POP ES
POP DX
POP CX
POP BX
POP AX
RET
ENDP DISPLAY_ERROR

.DATA
ErrStr dw Err00, Err01, Err02, Err03, Err04, Err05, Err06, Err07, Err08, Err09
   dw Err10, Err11, Err12, Err13, Err14, Err15, Err16, Err17, Err18, Err19
   dw Err20, Err21, Err22, Err23, Err24, Err25, Err26, Err27, Err28, Err29
   dw Err30, Err31, Err32, Err33, Err34, Err35, Err36, Err37
Err00 db 'no error found', 0
Err01 db 'function number invalid', 0
Err02 db 'file not found', 0
Err03 db 'path not found', 0
Err04 db 'to many open files', 0
Err05 db 'access denied', 0
Err06 db 'handle invalid', 0
Err07 db 'memory control blocks destroyed', 0
Err08 db 'insufficient memory', 0
Err09 db 'memory block address invalid', 0
Err10 db 'environment invalid', 0
Err11 db 'format invalid', 0
Err12 db 'access code invalid', 0
Err13 db 'data invalid', 0
Err14 db 'unknown unit', 0
Err15 db 'disk drive invalid', 0
Err16 db 'attempted to remove current directory', 0
Err17 db 'not same device', 0
Err18 db 'no more files', 0
Err19 db 'disk write-protected', 0
Err20 db 'unknown unit', 0
En21 db 'drive not ready',0
En22 db 'unknown command',0
En23 db 'data error (crc)',0
En24 db 'bad request structure length',0
En25 db 'seek error',0
En26 db 'unknown media type',0
En27 db 'sector not found',0
En28 db 'printer out of paper',0
En29 db 'write fault',0
En30 db 'read fault',0
En31 db 'general failure',0
En32 db 'sharing violation',0
En33 db 'lock violation',0
En34 db 'disk change invalid',0
En35 db 'FCB unavailable',0
En36 db 'sharing buffer exceeded',0
En37 db 'check DOS documentation',0

.CODE

; INT24h Substitute critical-error handler to tell DOS to Retry or Fail errors and
; return to the calling program. This subroutine will redirect DOS's
; attempt back to the calling program.
; Note: The Abort is converted to what DOS calls a Fail and will return
; control back to the calling program with an error code in AL.
; INT23h Ignore the control C break command from the keyboard.

PROC INTERRUPT Handler
;----Install critical-error handler
    PUSH DS
    MOV DX,Seg INT24
    MOV DS,DX
    MOV DX,Offset INT24
    MOV AX,2524h
    INT 21h
;----Install ^C error handler (ignore ^C breaks)
    MOV DX,Seg INT23
    MOV DS,DX
    MOV DX,Offset INT23
    MOV AX,2523h
    INT 21h
    POP DS
    RET
;----Substitute interrupt 23h
PROC INT23 FAR
    XOR AX,AX
    IRET
ENDP INT23
;----Substitute interrupt 23h
PROC INT24 FAR
    PUSH BX

G-93
PUSH E
PUSH SI
PUSH DI
PUSH BP
PUSH DS
PUSH ES
MOV DX,AX
MOV AX,DATA
MOV DS,AX
MOV CL,[Color]
MOV AL,[Warning]
MOV [Color],AL
MOV AX,0229h
CALL GOTOXY
CALL CSTR_OUT
db 7,7,‘ Error: Press R to Retry or A to Abort ’,
CALL HIDE_CUR
CRT1: MOV AH,6
MOV DL,OFFh
INT 21h
JZ CRT1
AND AL,5Fh
MOV AH,AL
MOV AL,3
CMP AH,’A’
JZ CRT2
MOV AL,1
CMP AH,’R’
JNZ CRT1
CRT2: MOV DL,AL
MOV AL,[Menu]
MOV [Color],AL
MOV AX,0220h
CALL GOTOXY
CALL CSTR_OUT
db 0
MOV [Color],CL
MOV AX,DX
POP ES
POP DS
POP BP
POP DI
POP SI
POP DX
POP CX
POP BX
IRET
ENDP INT24
ENDP INTERRUPT_HANDLER
;—-The Pause for set for 1/2 second.
; Input = None
; Output = None

PROC PAUSE
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    XOR AX,AX
    ;Get ticks function no.
    INT 1Ah
    ;Get Dos timer ticks
    MOV BX,DX
    ;Low byte ticks to BX
    ADD BX,9
    ;9 = 1/2 second
    PA1: XOR AX,AX
    ;Get ticks function no.
    INT 1Ah
    ;Get Dos timer ticks
    CMP BX,DX
    ;Is time run out?
    JNC PA1
    ;If not loop again
    CLC
    ;Clear carry flag
    POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP PAUSE

;—-Select a Select a Key file.
;—-Input = None
;—-Output = Carry Flag if DOS Error
;—-Local variables:
; BH = highlight bar position # 1 to 14
; BL = starting directory number # 1 to Max File

PROC SELECT FILE
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLOSE_FILE
    ;Close any open file
    CALL SET_TYPE
    ;POS or NEG variables
    CALL CREATE_MEM_DIR
    ;Make a memory directory
    JC SEL6
    ;Exit if error
    CALL SELECT_SCREEN
    ;Display select screen
    JC SEL4
    ;No files in directory

;—-Display the files
    MOV BX,[BarPos]
    ;Starting position
    SEL0: MOV AX,BX
    ;File variables to AX
    CALL DISPLAY_FILES
    ;Files to screen
    SEL1: CALL GETCHAR
    CMP AL,1Bh
    ;Is it an Esc key?
    JNZ SEL2
    ;If not goto next test
JMP SEL4 ;Exit
SEL2: CMP AL,0Dh ;Is it a pick?
JZ SEL3 ;If YES exit loop

;--see if an active control key was pressed
CALL CONTROL KEYS
JC SEL1 ;no change get char
JMP SHORT SEL0 ;redraw file window

;--Open file, count ID's, and look for ranked data line.
SEL3: CALL MOVE_NAME ;file name to data seg
CALL OPEN_SLDI ;open SLDI file
JC SEL5 ;Esc = main menu
MOV [BarPos],BX ;save current position
CALL READ_DATE ;get DOS date of file
CALL FINDZERO ;locate 000 in data file
JC SEL5

SEL4: CALL RELEASE_MEM_DIR ;release mem block
CLC ;clear carry flag
JMP SHORT SEL6

SEL5: CALL MOVE_NAME ;file name to data seg
CALL OPEN_SLDI ;open SLDI file
JC SEL5 ;Esc = main menu
MOV [BarPos],BX ;save current position
CALL READ_DATE ;get DOS date of file
CALL FINDZERO ;locate 000 in data file
JC SEL5

SEL6: POP DX
POP CX
POP BX
POP AX
RET

Adjust the highlight bar position variables in BX register
Input = AL = Char for the keyboard
BH = hllite bar position # 1 to 14
BL = starting directory number # 1 to MaxFile
Output Carry Flag = no change in BX

PROC CONTROL KEYS ;--Is it a Down arrow?
CMP AL,24 ;Is it Down arrow?
JNZ KYS2 ;If not goto next test
MOV AX,BX ;get current variables
ADD AL,AH ;inc hllite bar
CMP [MaxFile],AH ;Is it end of file?
JC KYS10 ;If yes Exit no changes
CMP BH,14 ;Is bottom of window?
JZ KYS1 ;If yes Inc starting
INC BH ;else Inc bar number
JMP SHORT KYS9 ;exit

KYS1: INC BL ;inc starting number
JMP SHORT KYS9 ;display new directory

;--Is it an Up arrow?
KYS2: CMP AL,5 ;Is it Up arrow?
JNZ KYS4 ;If no goto next test
CMP BX,0101h ;Is it beginning of file?
STC ;set carry for ret
JZ SHORT KYS10 ;If yes Exit no changes

G-96
CUP

BH,1 ; Is top of window?
JZ KYSS
DEC BH
JMP SHORT KYSS9

KYSS: DEC BL
JMP SHORT KYSS9

;— Is it a Home Key?

KYSS: CMP AL,1
JNZ KYSS5
MOV BX,0101h
JMP SHORT KYSS9

;— Is it a End Key?

KYSS: CMP AL,6
JNZ KYSS7
MOV AH, [MaxFile]
CMP AH, 15
JNC KYSS6
MOV BL, 1
MOV BH, AH
JMP SHORT KYSS9

KYSS6: SUB AH, 13
MOV BLAH
MOV BH, 14
JMP SHORT KYSS9

;— Is it the PageUp Key?

KYSS: CMP AL, 18
JNZ KYSS8
MOV AH, BL
SUB AH, 14
MOV AL, 1
JLE KYSS4
MOV BLAH
MOV AL, AH
JMP SHORT KYSS9

;— Is it the PageDown Key?

KYSS: CMP AL, 3
STC
JNZ KYSS10
MOV AH, BL
MOV AL, [MaxFile]
SUB AL, 14
ADD AH, 14
CMP AL, AH
MOV AL, 6
JLE KYSS5
MOV BLAH

KYSS: CLC
KYSS10: RET
ENDP CONTROL_KEYS

;— Display Files in Memory Directory
; Input: AL = starting directory number (1 to MaxFile)
; AH = hllite bar number (1 to 14)
Output a 14 line of file names to the screen.

Note: local variables:
- AH = non-hilite color attribute
- BX = row/col
- DH = hilite bar color attribute
- CX = loop counter
- DL = reverse hilite bar number (14 to 1)

Note: the hilite bar counter stored in DL is reversed from 1 to 14
into 14 to 1 so it can be compared to the loop counter in
CX to select the correct row to hilite.

PROC DISPLAY FILES
    PUSH AX ; save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV DL,15
    SUB DL,AH ; convert 1 to 14 into
    MOV AH,[Menu] ; normal attribute
    MOV [Color].AH ; set default color
    MOV DH,[Warning] ; hilite bar attribute
    MOV BX,0520h ; row 6/col 32
    MOV CX,14 ; number of rows
    DISO: CMP DL,CL ; is this the hilite bar
           JNZ DIS1
           MOV [Color].DH ; if yes color = warning
    DIS1: CALL DIR STR ; display one file name
           CMP DL,CL ; is this the hilite bar
           JNZ DIS2
           MOV [Color].AH ; if yes color = menu
    DIS2: INC BH ; ptr to next row
           INC AL ; ptr to next dir entry
           LOOP DISO ; loop 14 times
           CALL HIDE CUR
           CLC ; clear carry flag
    DIS3: POP DX ; restore registers
           POP CX
           POP BX
           POP AX
    RET

ENDP DISPLAY FILES

PROC MOVE NAME
    PUSH AX ; save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV AX,BX ; memblk ptr to AX
    ADD AL,AH ; start dir no + hilite
    DEC AL ; minus 1 = memblk recno

G-98
XOR AH, AH ; convert 16 bit number
MOV CL, 4 ; shift counter
SHL AX, CL ; 2^4 = times 16
INC AX ; skip 2 leading spaces
INC AX
MOV SI, AX ; memblock offset to SI
MOV DI, Offset FilaName ; destination offset
MOV AX, DS ; assign ES to the
MOV ES, AX ; data section
MOV AX, [DirSeg] ; memblock base ptr
MOV DS, AX ; assign memblock to DS
CLD ; auto inc SI & DI
MOV: MOV AL, [SI] ; get first byte
CMP AL, '' ; is beginning of type?
JZ MOV1 ; exit loop if yes
MOVSB ; move byte
JMP SHORT MOV0 ; loop until end of type
MOV1: MOV AX, ES ; restore DS to
MOV DS, AX ; point to the data seg
MOV CX, 5 ; number byte to move
MOV SI, Offset FilaTyp ; point to 5 bytes string
REP MOVSB ; move type to FilaName
CLC ; clear carry flag
POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP MOVE_NAME

PROC SELECT SCREEN
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
XOR AX, AX ; row 0, column 0
CALL MENU BOX ; draw menu box
MOV AX, 010Bh ; row 1, column 4
CALL GOTOYX
MOV CL, [Color] ; save current color attr
MOV AL, [Menu] ; set color = menu
MOV [Color], AL
CALL CSTR_OUT
db 'Use the , 24, and , 25, arrow keys to highlight the '
db 'desired data file', 0
MOV AX, 020Fh ; row 2, column 7
CALL GOTOYX
CALL CSTR_OUT

G-99
MOV AL,[Normal] ;set Color
MOV [Color],AL
CALL CLEAR TITLE
MOV AX,0405h
CALL GOTOYX
CALL CSTR OUT
db 'Press the <Enter> key to select the highlighted file.',0
MOV AX,0
CALL DSTR OUT
MOV AX,150h ;row 21,column 0
CALL MENU_BOX ;draw menu box
MOV AL,[Menu] ;set color = menu
MOV [Color],AL
MOV AX,1606h ;row 23,column 5
CALL GOTOYX
CALL CSTR OUT
db 'Directory Path : ',0
MOV AX,Offset Path
CALL DSTR OUT
MOV AX,1709h ;row 23,column 5
CALL GOTOYX
CALL CSTR OUT
db 'Press the <Esc> key to return to the menu without selecting a file.',0
MOV AX,1709h ;row 23,column 5
CALL GOTOYX
CALL CSTR OUT
db 'The current directory contains: ',0
XOR AX,AX ;zero AX register
MOV AL,[Maxfile] ;load number FIL files
CMP AL,1 ;is it only one?
JZ CEE1 ;is yes singular text
CMP AL,0 ;is it zero?
JZ CEE2 ;if yes display error
CALL BIN OUT ;else display number
CALL CSTR OUT ;of files.
JMP CEE4
CEE1: CALL CSTR OUT ;singular text mess.
JMP CEE4
CEE2: CALL CSTR OUT ;zero files statement
JMP CEE4
db " files with a type of ",0
MOV AL,[Warning] ;warning color
MOV [Color],AL ;set color
MOV AX,0506h ;row 5 Col 7
CALL GOTOYX ;set cursor
CALL CSTR OUT ;display warning
db 'No ',0
MOV AX,Offset FilTyp
CALL DSTR OUT
CALL CSTR OUT
db ' files found in directory! Press Any Key for '
db 'previous Menu.',0
CALL HIDE_CUR
CALL GET_CHAR ;wait for keyboard key
MOV [Color].CL ;restore original Color
STC ;set carry flag
JMP SHORT CEE5 ;exit no files found

;draw background boxes and key descriptions
CEE4: MOV AX, Offset FITyp
CALL DSTR_OUT
CALL CSTR_OUT
db '.*',0
CALL SELECT_WINDOW
CEE5: POP DX ;restore registers
POP CX
POP BX
POP AX
RET

;draw file display windows and key instructions
: Input = None
: Output = None
PROC SELECT_WINDOW
MOV AL,[Normal] ;set Color
MOV [Color].AL
MOV AX,0804h ;row 8, column 4
CALL GOTOYX
CALL CSTR_OUT
db '<Up Arrow> = Move Up','
db '<Down Arrow> = Move Down',0
MOV AX,0A04h ;row 10, column 4
CALL GOTOYX
CALL CSTR_OUT
db '<PageUp> = Scroll Up','
db '<Home> = First File',0
MOV AX,0C04h ;row 12, column 4
CALL GOTOYX
CALL CSTR_OUT
db '<PageDown> = Scroll Down','
db '<End> = Last File',0

;draw display windows
MOV AL,[System]
MOV [Color].AL
MOV AX,0621h
MOV BX,1332h
CALL CLEAR_WINDOW
MOV AL,[Menu]
MOV [Color].AL
MOV AX,051Fh
MOV BX,1230h
CALL CLEAR_WINDOW
CALL HIDE_CUR
MOV [Color].CL ;restore original Color
CLC ;clear carry flag

G-101
Create a directory of the FeedBack files in memory.

Input = None
Output = Carry flag if DOS error. AL = FFh is to many files
[DirSeg] = Starting segment address of memory block.
[MaxFile] = total number of FeedBack files.

PROC CREATE_MEM_DIR
PUSH BX
PUSH CX
PUSH DX
PUSH ES
CALL RELEASE_MEM_DIR ;continue if no error
JNC CREATE_EXP1 ;exit on DOS error
JMP CREATE_EXP2
CREATE_EXP0: CALL COUNT_FILES ;how many SLDI files?
CMP AX,251 ;is files found < 251 ?
JNC CREATE_EXP1 ;if no display err mess
MOV [MaxFile],AL ;save number of files
JMP CREATE_EXP2 ;if yes OK! continue
CREATE_EXP1: MOV AX,030Bh ;row 3 / col 1
CALL GOTOXY ;position cursor
CALL CSTR_OUT ;string to screen
db 'There are to more than 250 SLD files in this directory.';0
MOV AX,0511h ;row 3 / col 1
CALL GOTOXY ;position cursor
CALL CSTR_OUT ;string to screen
db 'Please move some of them to another directory.';0
MOV AX,071Ah ;row 3 / col 1
CALL GOTOXY ;position cursor
CALL CSTR_OUT ;string to screen
db 'Press Any Key to Exit to DOS.';0
CALL HIDE_CUR ;hide cursor
CALL GET_CHAR ;wait for keypressed
MOV AL,0FFh ;to many files marker
JMP CREATE_EXP2 ;exit to many files
CREATE_EXP2: CMP AL,0 ;were any files found ?
JZ CREATE_EXP1 ;if no files Exit
CREATE_EXP3: CALL GET_DIR_BLK ;allocate memory blk
JC CREATE_EXP2
CALL MAKE_DIR
JC CREATE_EXP2
CALL SHELL_SORT
CREATE_EXP8: CLC ;clear carry flag
CREATE_EXP9: POP ES
G-102
```assembly
POP DX ; restore registers
POP CX
POP BX
RET
ENDP CREATE_MEM_DIR

Make a directory of Feedback files in memory block [DirSeg]
Each entry is 16 bytes. Format: 2 spaces + File Name + padding spaces = 16
Input = [DirSeg] and [Search] in data section
Output = None

PROC MAKE_DIR
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH DS
PUSH ES
MOV AX, DS
MOV ES, AX
CMP WORD [DirSeg], 0
STC set carry if error
JZ COP4 ; if no memblk exit
----- find first match
XOR BX, BX zero file counter
MOV AX, 4E00h ; find a first file
XOR CX, CX ; ordinary files only
MOV DX, Offset Search ; ptr file name ASCII
INT 21h ; do the first search
JC COP4 ; if no match exit
----- set up ES and DS segment registers
MOV AH, 62h ; get the current PSP
INT 21h ; segment address.
JC COP4 ; exit on error
MOV AX, [DirSeg] ; ptr to base of memblk
MOV ES, AX ; ES set to memory blk
MOV DS, AX ; DS set to memory blk
----- set directory entry 0 = a blank ASCII string (16 spaces)
MOV AX, 2020h ; two spaces in ASCII
MOV DI, 2 ; destination ptr
MOV SI, 0 ; source ptr
MOV CX, 7 ; loop counter
MOV [SI], AX ; place 1st 2 bytes
CLD ; auto inc SI & DI
REP MOVSW ; place next 14 bytes
----- copy directory entries loop
MOV DS, BX ; DS set to PSP
----- place leading 2 spaces
COP0: MOV AX, 2020h ; two ASCII spaces
      MOV [ES:DI], AX ; place in directory
      INC DI ; advance directory ptr
      INC DI
```

G-103
;-----move one file name
MOV SI,9Eh
;max length of Name
MOV CX,12 ;max file name length

COP1: MOV AL,[SI] ;load byte to be moved
CMP AL,0 ;is it end of string?
JZ COP2 ;if end exit loop
CLD ;auto inc SI & DI
MOVSB
LOOP COP1 ;copy file name

;-----pad end of file name with spaces.
COP2: ADD CX,2 ;number of bytes
MOV AL,20h ;space to AL

COP3: MOV [ES:DI],AL ;place space in dir
INC DI ;ptr to next byte
LOOP COP3 ;loop until CX = 0

;-----find next match
MOV AX,4F00h ;fine next file function
INT 21h ;do next search
JNC COP0 ;loop until all found
CLC ;clear carry flag

COP4: POP ES ;restore registers
POP DS
POP DX
POP CX
POP BX
POP AX
RET

ENDP MAKE_DIR

;---- Allocate memory block for the Director of files ([MaxFile] + 2 paragraphs)
; Input = None
; Output = Carry flag set if memory block is not available.
; Index file seg address stored in [DirSeg]
; Note: The binary SEARCH procedure needs a blank record before
; the memory index records. The number of paragraphs
; needed is [MaxFile] + 1.

PROC GET_DIR_BLK
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX

MEM1: MOV BL,[MaxFile] ;get number of files
XOR BH,0h ;zero high byte
INC BX ;get an extra paragraph
MOV AH,48h ;allocate men function
INT 21h ;request memory block
JC MEM2 ;jump if memory error.
MOV [DirSeg],AX ;base address of seg
JMP SHORT MEM3 ;normal exit of proc.

MEM2: MOV CL,[Color] ;save original color

G–104
MOV AL, [Warning] ; warning color
MOV [Color], AL ; set color
MOV AX, 0101h ; row 1/Col 1
CALL GOTOXY ; position cursor
CALL CSRTOUT ; send string to screen

db 'Not enough memory for the directory of files.'

db 'Press Any Key to Continue.', 0
MOV [Color], CL ; restore original color
CALL HIDE CUR ; hide cursor off screen
CALL GET CHAR ; wait for key is pressed
STC ; set carry flag = error

MEM3: POP DX
PUSH CX
PUSH BX
PUSH AX
RET

ENDP GET_DIR_BLK

--- Adjust the DOS memory block size allocation to the minimum amount.
Input = None
Output = Carry flag set if memory block error.
Note: Assumes the programs memory is in a single block
and the stack segment is at the end of the program.

PROC RELEASE_MEM
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
PUSH ES
MOV AX, STACKSIZE ; current stack size
MOV CL, 4 ; convert to paragraphs
SHR AX, CL ; divide by 2^4 or 16
INC AX ; round up 2 paragraphs
INC AX ; to protect top of stack
MOV CX, AX ; save in register CX
MOV AX, SS ; get stack seg address
ADD CX, AX ; ptr to end of stack
MOV AH, 62h ; get the current PSP
INT 21h ; segment address.
JC RELD ; edit on error
MOV ES, BX ; ptr to current PSP
SUB CX, BX ; program size in
MOV BX, CX ; paragraphs to BX.
MOV AH, 4Ah ; release mem function
INT 21h ; release previous block.
RELD: POP ES ; restore registers
PUSH DX
PUSH CX
PUSH BX
PUSH AX

G-105
Count the number of Feedback data files in the current directory.

Input = None

Output = AX = total number of Feedback files found.

assumed any file ending with a .SLD ext is a Feedback data file.

When the file is opened the data will be validated.

PROC COUNT FILES
PUSH BX
PUSH CX
PUSH DX
MOV AX, DS
MOV ES, AX

;--- copy Path to [Search]
MOV SI, Offset Path
MOV DI, Offset Search
CLD

COU1: MOVSB
CMP BYTE PTR [SI], 0
JNZ COU1

; auto Inc DI and SI

COU2: MOV SI, Offset Path
MOV CX, 13
CLD
REP MOVSB

; find first file
XOR BX, BX
MOV AX, 4E00h
XOR CX, CX
MOV DX, Offset Search
INT 21h

; do the first search
JC COU6

; if no match exit

COU5: INC BX
MOV AX, 4F00h
INT 21h
JNC COU5

; do next search

COU6: MOV AX, BX
CLC
POP DX
POP CX
POP BX
RET

ENDP COUNT_FILES

;--- Fill the name field with 13 spaces in the data section.
; Input = AX = pointer to field

G-106
Output = None

PROC CLEAR_FIELD
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV BX,AX ;ptr to field.
MOV AX,DS ;Make ES = DS
MOV ES,AX
MOV CX,12 ;restore length of str
MOV AL,' ' ;place a space in first
MOV [BX],AL ;byte of [Input] string.
MOV DI,BX ;DI = pointer to next
INC DI ;byte of string
MOV SI,BX ;SI = pointer to str
CLD ;auto inc DI and SI
REP MOVSB ;fill str with spaces
POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP CLEAR_FIELD

Sort the Memory Index Records.

Input = expects the 16 byte index records to be located at address
pointer [IdxSeg] and the number of record to be [MaxRec]
Output = None
Note: this routine reassigns the DS and ES registers to point to the
Index File in memory. Record 0 is not sorted. The sort is
from record 1 to MaxRec. A blank record in record 0 is needed
for an ASCII string when performing a binary search.
The memory index record length is 16 bytes.
The sort is based on the first 10 bytes.

This sort is based on the following TPASCAL procedure:
PROCEDURE Sort; {A Shell Sort}
VAR
    Gap,J : Integer;
    Temp : string[13];
    TempNo : Integer;
Begin
    Gap := MaxRec Div 2;
    While gap > 0 Do
    Begin
        For I := (Gap + 1) to MaxRec Do
        Begin
            J := I-Gap;
            ...
While \( J > 0 \) Do

Begin

If \( A[J] > A[J+\text{Gap}] \) then

Begin

Temp := A[J];
A[J+\text{Gap}] := Temp;
\( J := J - \text{Gap} \);
End

Else \( J := 0 \);
End;

End;

\( \text{Gap} := \text{Gap} \div 2 \);
End;

End;

The following registers hold the above variables:

\( AX = \text{Gap}; \ BX = J; \ CX = I; \ DX = \text{MaxRec}; \) and \( BP = \text{temp storage} \)

PROC SHELL_SORT

PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
PUSH DS
PUSH ES
PUSH SP
MOV DL, [MaxFile] ; store MaxRec in DX
XOR DH, DH ; zero high byte
MOV AX, [DirSeg] ; get index base segment
MOV DS, AX ; reassign the DS & ES
MOV ES, AX ; to ptr to the index
MOV AX, DX ; \( \text{Gap} = \text{MaxRec} \)
SHR AX, 1 ; \( \text{Gap} = \text{Gap} \div 2 \)
SHEL1: CMP AX, 0 ; when \( \text{Gap} = 0 \) exit.
JLE SHEL4 ; exit if \( \text{Gap} < 0 \)
MOV CX, AX ; \( I \) is stored in CX
INC CX ; \( I = \text{Gap} + 1 \)
SHEL2: MOV BX, CX ; \( J \) in BX
SUB BX, AX ; \( J = I - \text{Gap} \)
JZ SHEL3 ; skip if \( J = 0 \)
JC SHEL3 ; skip if \( J < 0 \)
CALL COMPARE_SWAP ; repeat until \( J = 0 \)
SHEL3: INC CX ; \( I = I + 1 \)
CMP DX, CX ; is \( I \) < or = MaxRec
JNC SHEL2 ; if yes then loop.
SHR AX, 1 ; \( \text{Gap} = \text{Gap} \div 2 \)
JMP SHORT SHEL1
SHEL4: POP BP ; restore registers
POP ES
POP DS
POP DX
POP CX

G-108
POP BX
POP AX
RET ; sort is complete.

--- Compare and swap index strings if needed.
Note: This is a subroutine of SHELL_SORT. The index file record length is 16 bytes. The sort is made on the first 8 bytes.

Input = AX = Gap; BX = J; DS & ES point to the base of index file.
Output = AX = Gap; CX = I; and DX = MaxRec are returned on changed.
BX = J is discarded.

PROC COMPARE_SWAP
PUSH AX ; save registers
PUSH CX
PUSH DX
MOV DX,AX ; save Gap in DX

; Compare the first six bytes of each index record
COMP1: MOV BP,BX ; save J in BP
ADD AX,BX ; AX = J + Gap
MOV CL,4 ; shift counter
SHL AX,CL ; ptr to J + Gap in mem
SHL BX,CL ; ptr to J in mem
CLD ; auto-inc SI, DI
MOV DI,AX ; offset of J + Gap
MOV SI,BX ; offset of J
MOV CX,10 ; byte counter
REPE CMPSB ; compare strings
JLE COMP3 ; exit if < or =.

; Swap the 16 bytes of index record if string A > string A + Gap
MOV DI,AX ; offset of J + Gap
MOV SI,BX ; offset of J
MOV CX,8 ; word counter

COMP2: MOV AX,[SI] ; read word each str.
MOV BX,[DI]
MOV [SI],BX ; write word each str.
MOV [DI],AX
INC DI ; point to next word
INC DI
INC SI ; point to next word
INC SI
LOOP COMP2 ; loop five times
MOV AX,DX ; restore gap to AX
MOV BX,BP ; restore J to BX
SUB BX,AX ; J = J - gap
JZ COMP3 ; exit if J = 0.
JNC COMP1 ; continue if J > 0.

COMP3: POP DX ; restore registers
POP CX
POP AX
RET ; return to Shell_Sort

ENDP COMPARE_SWAP

G-109
SEND a 16 byte memory directory entry to the Screen.

Input = AL = DirFile number (0 to MaxFile) 0 = blank directory entry
BX = row /col

[MaxFile] = the number of directory entries in the memory dir
[DirSeg] = segment address of the base of the memory directory
Output = ASCII string sent to the screen

PROC DIR STR
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX

;--compute dirfile offset
XOR AH, AH ;AX = DirFile number
CMP [MaxFile], AL ;is DirFile # OK?
JNC DIO ;if in bounds jump
MOV AL, AH ;else make blank file

DIO: MOV CL, 4 ;shift 4 = times 16
SHL AX, CL ;mult by 16
MOV SI, AX ;ASCIIZ message ptr SI
MOV AX, BX ;row/col to AX
CALL GOTOYX ;position cursor
MOV AX, [DirSeg] ;place the memory blk
MOV DS, AX ;seg in DS.
MOV CX, 8 ;8 words = 16 bytes
CLD
REP MOVSW ;restore reg DS to
MOV AX, ES ;point to data segment.
XOR AL, AL ;place zero in string
MOV [DI], AL ;as EndOfString marker
MOV AX, Offset Input ;ptr to input string
CALL DSTROUT ;send name to the screen
POP DX
POP CX
POP BX
POP AX
RET
ENDP DIR STR

Input = none
Output = none

PROC PRINT_WAIT_MESS
PUSH AX
PUSH BX
PUSH CX
PUSH DX

G-110
;--please wait message to screen.
CALL CLEAR MESSAGE
MOV CL,[Color] ;save original attri
MOV AL,[Warning] ;warning color
MOV [Color],AL ;set color
MOV AX,0208h ;row 3/Col 12
CALL GOTOYX ;set cursor
CALL CSTR_OUT ;display warning
db 'Please wait ....... Reading data file: '0
MOV AX, Offset FileNa
CALL DSTR_OUT
CALL CSTR_OUT
db '0
MOV [Color],CL ;restore original attri
CALL HIDE_CUR
CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP PRINT_WAIT_MESS

.. Release the memory directory and variable blocks.
.. Input = None
.. Output = Carry flag if DOS error
.. [DirSeg] = Starting segment address of directory block.
.. [VarSeg] = starting segment address for variable block.
.. [MaxFile] = total number of Feedback files.

PROC RELEASE_MEM_DIR
PUSH BX
PUSH CX
PUSH DX
PUSH ES
XOR AX,AX ;zero AX
CMP [DirSeg],AX ;is DirSeg assigned?
JZ REL2 ;if not assigned go on
--- release assigned memory block
MOV AX,[DirSeg] ;get memory segment
MOV ES,AX ;place in ES register
MOV AX,4900h ;release function no
INT 21h ;release memory block
JC REL2 ;if No error continue
--- initialize variables
MOV AX,0101h ;set barposition to
MOV [BarPos],AX ;start = 1 hilight = 1
XOR AX,AX ;zero to register
MOV [DirSeg],AX ;set memory block to 0
MOV [MaxFile],AL ;set maxfiles to 0
CLC ;clear carry flag
REL2: POP ES

G-111
POP DX ;restore registers
POP CX
POP BX
RET

ENDP RELEASE_MEM_DIR

;---Read the file's DOS date to the [Date] string
; Input = None
; Output = file date to [Date]

PROC READ_DATE
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV BX,[FileHd] ;load file handle
CMP BX,0 ;is a file open
JZ DOS2 ;if not exit
MOV AX,5700h ;get date stamp funct.
INT 21h ;get stamp
JC DOS2 ;if DOS error Exit
MOV BX,DX ;composite to get day
AND BX,01Fh ;isolate day
MOV CL,5 ;shift counter
SHR DX,CL ;month to bits 0 to 3
MOV AX,DX ;composite to get month
AND AX,0Fh ;isolate month
MOV CL,4 ;shift counter
SHR DX,CL ;year to bits 0 to 5
AND DX,03Fh ;isolate year
ADD DX,80 ;add base year
MOV CX,BX ;store day in CX

;---convert to ASCII ;AX=Mon,CX=day,DX=year
MOV BX,Offset Date ;ptr to Date string
CALL CONVERT_ASCII ;place month in string
MOV AX,CX ;day of month to AX
MOV BX,Offset Date + 3 ;ptr to day section
CALL CONVERT_ASCII ;place day in Date str
MOV AX,DX ;place year in AX
MOV BX,Offset Date + 6 ;ptr to year section
CALL CONVERT_ASCII ;place year in Date str

DOS2: CLC ;clear carry flag
POP DX
POP CX
POP BX
POP AX
RET

;---Convert hex number into 2 digit ASCII number.
; Input = AX = hex number
; BX = ptr to [Date]
Output = two byte number into [Date] string

PROC CONVERT_ASCII
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  CMP AX, 100
  JC COVI ; is it a 2 digit number?
  XOR AX, AX ; set number to 00
  COVI: MOV CL, 10
  MOV CX, 0
  DIV CL ; AX/10
  JC COVI1 ; if yes continue else
  CLC ; clear carry flag
  MOV AX, AX
  MOV BX, 3030h ; convert to ASCII
  MOV [BX], AX
  MOV AX, 0
  POP AX
  POP DX
  POP CX
  POP BX
  POP AX
  RET
ENDP CONVERT_ASCII
ENDP READ_DATE

PROC SET_TYPE
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  MOV AX, DS
  MOV ES, AX
  MOV BX, Offset PPI
  CMP BYTE PTR [Report], 0 ; is it a POS report?
  JZ STT1 ; if YES goto STT1
  MOV BX, Offset Negty
  MOV CX, 0
  CMP BYTE PTR [Report], 0 ; is it a NEG report?
  JZ STT11 ; if YES goto STT11
  MOV CX, 2
  CLD ; clear carry flag
  MOV DI, Offset FileNa + 8 ; destination pointer
  MOV SI, SI
  MOV CX, CX
  REP MOVSW ; move two Words
  MOV DI, Offset FileNa + 8 ; destination pointer
  MOV SI, SI
  MOV CX, CX
  REP MOVSW ; move two Words
  MOV DI, Offset FileNa + 8 ; destination pointer
  MOV SI, SI
  MOV CX, CX
  REP MOVSW ; move two Words
G-113
MOV SI,BX ;source pointer
MOV CX,DX ;loop counter = 2
REP MOVSW ;move two Words
POP DX
POP CX
POP BX
POP AX
RET

ENDP SET_TYPE

;---Count number of ID's and find '000' ID number in data file.
; Input = assumes '000' ID number is after all ID data lines.
; Output = [Ranked] = True if '000' found.

PROC FIND_ZERO
    PUSH AX ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    MOV AX,DS
    MOV ES,AX
    XOR AX,AX
    MOV [MaxNo].AX ;set No of ID's = 0
    MOV [Ranked].AL ;set [Ranked] = FALSE
    MOV [EOF].AL ;set EndOfFile = FALSE
    CALL GOTO_TOP ;file ptr to BeginOfFile
    JC FZR8 ;exit on DOS error
;---set default ID string to ASCII zeros
    MOV BX,Offset ID ;ptr to string to edit
    MOV DI,BX ;ptr to string to fill
    MOV AX,3030h ;ASCII zeros
    MOV [DI].AX ;place 1st two bytes
    XOR AH,AH ;zero = end of string
    INC DI ;advance string ptr
    INC DI
    MOV [DI].AX ;ASCII 0 and hex 0
;---locate ID number in the data file
    CALL PRINT_WAIT_MESS ;inform user of search
    XOR BX,BX ;ID counter = 0
FZR1: CALL READ_LINE ;1 line from data file
    JNC FZR2 ;not EndOfFile
    MOV AL,OFFH ;mark EndOfFile true
    MOV [EOF].AL ;<> 0 = True
FZR2: MOV CX,3 ;loop counter
    MOV DI,Offset ID ;ptr to ID number
    MOV SI,Offset FillBuf ;ptr to data file line
    CLD
    REPZ CMPSB ;are the bytes = ?
    JNZ FZR3 ;If NO goto next test
    G-114
CALL COPY PERCNT
JMP SHORT FZR5

---Is this an ID data line?
FZR3: MOV SI, Offset FIBuf
MOV CX, 3
LOOP FZR4

---Is this an ID data line?
FZR4: MOV AL, [SI]
CMP AL, '0'
JC FZR5
CMP AL, ':'
JNC FZR5
INC SI
LOOP FZR4
INC BX

---Is this the last line?
FZR5: XOR AL, AL
MOV AL, [EOF]
JZ FZR1
MOV [MaxNo], BX
CLC

FZR6: POP DX
POP CX
POP BX
POP AX
RET

ENDP FIND_ZERO

---Copy PerCnt variables from data buffer to PerCnt variable string.
; Input = assumed 000 data in buffer in binary byte numbers
; Output = PerCnt variables set (60 bytes max)

PROC COPY PERCNT
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV AL, OFFH
MOV [Ranked], AL
MOV CX, 192
MOV BX, Offset FIBuf + 3
MOV SI, Offset PerCnt + 5
GES1: MOV AL, [BX]
CMP AL, 0
JZ GES5
CMP AL, ':'
JZ GES2

---Is value an ASCII 0-9?
CMP AL, '0'
JC GES3
CMP AL, ':'
JNC GES3

---Save digit in PerCnt variable

G-115
MOV [SI], AL ; save value
INC SI ; ptr to next variable
GES2: INC BX ; next byte in buffer
LOOP GES1 ; loop until CX = 0
DEC SI

; 20 var per dim and 4 dim = 80 two digit variables or 160 bytes
GES: MOV AX, SI
MOV BX, Offset PerCm + 5 ; starting position
SUB AX, BX ; AX = bytes found
CMP AX, 160 ; is the length correct
JZ GES4
Z = Normal exit else
GES3: CALL COPY_ERR ; error message
GES4: CLC
POP DX
POP CX
POP BX
POP AX
RET
ENDP COPY_PERCNT

; Input = none
; Output = none
PROC COPY_ERR
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV AL, [Warning] ; warning color
MOV CL, [Color] ; save original color
MOV [Color], AL ; set color
MOV AX, 020h ; row 3/Col 8
CALL GOTOYX ; set cursor
CALL CSTR_OUT ; display warning
db 'Line '000' is incorrect length for 4 * ',0
db 'Dimensions. Press Any Key. ',0
MOV [Color], CL ; restore original color
CALL ERR_SOUND
CALL HIDE_CUR
CALL GET_CHAR
CALL CLEAR_PERCNT ; zero percentile var.
XOR AL, AL
MOV [Ranked], AL ; mark file unranked
POP DX
POP CX
POP BX
POP AX
RET
ENDP COPY_ERR

G-116
.CODE

;--Open the answer SLD file for use by the Feedback program
; Input = name of file in [FileNa]
; Output = Carry flag set = critical DOS error.

PROC OPEN_SLD
PUSH BX
PUSH CX
PUSH DX
PUSH ES
CALL CLOSE_FILE ;close any open files

;--locate end of search string
MOV AX,DS
MOV ES,AX
MOV CX,68 ;max length of string
MOV BX,Offset Search ;ptr to first byte
MOV AL,0 ;looking for EndOfStr
KE1: INC BX ;ptr to next byte
CMP [BX],AL ;if this it ?
JZ KE2 ;if found
JNZ KE1 ;if no match look at next byte
STC
JMP SHORT KE9 ;if no match exit

;--backup until finding the last \
KE2: MOV CX,12
MOV AL,`
DEC BX
CMP [BX],AL
JZ KE3
LOOP KE2
STC
JMP SHORT KE9 ;if no match exit

;--copy file name to end of path
KE3: INC BX
MOV DI,BX ;destination ptr
MOV SI,Offset FileNa ;source ptr
MOV CX,13 ;number of bytes
CLD ;auto inc SI & DI
REP MOVSB ;copy all 13 bytes

;--open file and save file handle
MOV AX,Offset Search ;ptr to path + file name
CALL OPEN ;open key file
JC KE9 ;goto main menu on error
MOV [FileHd],BX ;save data file handle

;--set disk drive of open file
XOR AX,AX ;zero to [DiskDr] =
MOV [DiskDr],AL ;default drive
MOV BX,Offset Search ;ptr to path + filename
MOV AX,[BX] ;get first two bytes
CMP AH,;" ;is a drive given?
JNZ KE8 ;if NO will use default

G-117
SUB AL, 64 ; convert to hex value
JC KE8 ; if error continue
MOV [FileDr].AL ; save drive of file

KE8: CLC ; clear carry flag

KEG: POP ES
POP DX
POP CX
POP BX
RET

ENDP OPEN_SLDI

PROC CLOSE_FILE
PUSH BX
PUSH CX
PUSH DX
XOR AX, AX ; zero to AX
MOV BX, [FileHd] ; file handle
CMP BX, AX ; is the file open?
JZ CLO2 ; exit if file closed.
CALL CLEAR_PERCENT ; set percentiles = 0
MOV [FileHd], AX ; set file handle to 0
MOV [MaxNo], AX ; set total ID's to 0
MOV [Ranked], AL ; set ranked FALSE
MOV [FileDr], AL ; set file diskdrive = 0
MOV [EOF], AL ; set EOF = FALSE
MOV AH, 3Eh ; close file function no
INT 21h ; close data file
JNC CLO2 ; exit if successful.

CLO1: MOV AL, [Warning] ; warning color
MOV [Color], AL ; set color
MOV AX, 0207h ; row 2/Col 12
CALL GOTOXY
CALL CSTR_OUT ; display warning
db 'Error closing data file. Press Any Key to Continue.', 0
MOV AL, [Normal] ; normal color
MOV [Color], AL ; set color
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
STC ; set carry flag for ret

CLO2: POP DX ; restore registers
POP CX
POP BX
RET

ENDP CLOSE_FILE
Check to make sure a feedback file is in the directory.
Input = None
Output = Carry Flag if no file is open.

PROC IS_SLD
  PUSH AX  ;save registers
  PUSH BX
  PUSH CX
  PUSH DX
  XOR AX,AX  ;zero to AX register
  CMP [MaxFile],AL  ;were data files found?
  JZ DT1  ;0 means NO files
  JMP DT2  ;exit if found
DT1:  MOV CL,[Color]
      MOV AL,[Warning]  ;warning color
      MOV [Color],AL  ;set color
      MOV AX,020Bh  ;row 3/Col 12
      CALL GOTOYX  ;set cursor
      CALL CSTR_OUT  ;display warning
      db ' No key files found in directory! Press Any Key '
      db ' to Continue, ','0
      MOV [Color],CL  ;restore original color
      CALL HIDE_CUR
      CALL GET_CHAR
      STC  ;set carry flag
DT2:  POP DX  ;restore registers
      POP CX
      POP BX
      POP AX
      RET
ENDP IS_SLD

Inform the user the file is being opened.
Input = None
Output = None

PROC READ_MESS
  PUSH AX
  PUSH CX
  MOV CL,[Color]  ;save orig. color attr
  MOV AL,[Warning]  ;warning color
  MOV [Color],AL  ;set color
  MOV AX,020Bh  ;row 3/Col 12
  CALL GOTOYX  ;set cursor
  CALL CSTR_OUT  ;display warning
  db ' Reading File ','0
  MOV [Color],CL  ;restore orig. color attr
  CALL HIDE_CUR
  POP CX
POP AX
RET
ENDP READ_MESS

---Clear the second line of the menu box
Input = None
Output = None

PROC CLEAR_MESS
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV CL,[Color] ;save orig. color attr
MOV AL,[Menu] ;set menu color
MOV [Color].AL ;change color attribute
MOV AX,0207h ;row 2 and column 7
MOV BX,024Eh ;row 2 and column 78
CALL CLEAR_WINDOW ;clear out old message.
MOV AX,0207h ;row 2, column 7
CALL GOTOXY
CALL CSTR OUT
db 'Press the <Enter> key to open the highlighted file:.'
MOV [Color].CL ;restore orig.color attr
POP DX
POP CX
POP BX
POP AX
RET
ENDP CLEAR_MESS

---Read a line from the data file into the 128 byte memory buffer.
Input = file handle in [FileHd]
Output = sets [EOF] <> 0 when EndOfFile is reached.
Carry flag = file closes or file ptr already at EndOfFile.

NOTES:
Carriage returns are convert to hex 0.
Only the lower set ASCII characters are placed in the buffer.
No control codes etc.
Only the first 192 bytes of the line are saved in the buffer but the
procedure will keep reading until EndOfFile or an 0Dh is reached.

PROC READ_LINE
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
XOR AX,AX
MOV SI,Offset FBuf ;mark position in buffer

G-120
MOV  BP, Offset FBuf + 191  ;mark end of buffer
MOV  BX, [FileHd]          ;file handle
CMP  BX, AX               ;is a file open?
JZ   REE2                 ;if not Exit
CMP  [EOF]. AL            ;is ptr at end of file
JNZ  REE2                 ;if yes Exit

;———read 1 byte from data file
MOV  CX, 1                 ;read 1 byte
REE1: MOV  AX, 3F00h       ;read file function no
       MOV  DX, SI           ;buffer ptr to DX
       INT  21h              ;get byte
       JC   REE3             ;end of file?
       CMP  AX, CX           ;did it read a byte?
       JNZ  REE3             ;if no then EndOfFile
       MOV  AL, [SI]         ;get char in AL
       CMP  AL, 0DH           ;is it the end line?
       JZ   REE4             ;if YES exit
       CMP  AL, 128           ;is it 8th bit on?
       JNC  REE1             ;if yes read next char
       CMP  AL, 32            ;is it a control char?
       JC   REE1             ;if yes read next char
       CMP  BX, SI            ;is buffer full?
       JC   REE1             ;if yes read until 0DH
       INC  SI                ;if no advance buffer
       JMP  SHORT REE1        ;ptr & get another char
REE2: STC                   ;set carry flag
       JMP  SHORT REE5        ;exit finished file.
REE3: MOV  AL, 0FFh         ;non zero = end of file
       MOV  [EOF]. AL         ;mark endOfFile true
REE4: XOR  AL, AL           ;place end line
       MOV  [SI]. AL          ;in data file buffer
       CLC                     ;clear carry flag
REE5: POP  SI
       POP  DX
       POP  CX
       POP  BX
       POP  AX
       RET

ENDP READ_LINE

;Place the file pointer at the beginning of the open file.
; Input = none
; Output = Carry flag = error
PROC GOTO TOP
PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX
XOR  AX, AX               ;zero register
MOV  BX, [FileHd]         ;is a file open?
CMP  AX, BX               ;if not then exit

G-121
JM
TOM
; open rom to STC
design log
JMP
SHOT
TOP2,e"
; on error
;--.plkm
fe
pointo
the
beginning of the Gs
TOP: MOV CX,AX
; set offset = 0
MOV DX,AX
; set offset = 0
MOV AX,4300h
; set file pointer no.
INT 21h
; exit if error.
JC TOP2
; jump to register
XOR AL,AL
; set EndOfFile = False
CLC

TOP2: POP DX
POP CX
POP BX
POP AX
RET
ENDP GOTO_TOP

; Checks [FileDr] to make sure there is room for number of bytes in AX
; Input = [AX] = number of bytes needed
; assumes [FileDr] is pointing the desired drive
; 0 = default, 1 = A, 2 = B, etc
; Output = Carry flag = If not enough room
PROC IS FULL
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH BP
MOV BP,AX
; save bytes needed
MOV DL,[FileDr]
; get file drive no.
MOV AX,3800h
; disk space function
INT 21h
; get disk space
CMP AX,0FFFFh
; is drive valid?
JZ ISF3
; if NO exit error
CMP BX,BP
; available cluster > bytes
JNC ISF4
; yes OK! lots of room
MUL BX
; get available sectors
CMP DX,BP
; available sectors/65000
JNC ISF4
; greater than bytes?
CMP AX,BP
; available sectors > bytes
JNC ISF4
; if YES lots of room
MUL CX
; get available bytes
CMP DX,BP
; available bytes/65000
JNC ISF4
; greater than bytes?
CMP AX,BP
; available bytes > bytes
JNC ISF4
; if YES exit OK!
CALL FULL_ERR
; else inform user
ISF3: STC
JMP SHORT ISF5
ISF4: CLC

G-122
; Input = none
; Output = none

PROC  FULL_ERR
    CALL  CLEAR_MESSAGE
    MOV  AL,[Warning] ; warning color
    MOV  CL,[Color]   ; save original color
    MOV  AL,[Color].AL ; set color
    MOV  AX,0209h   ; row 3/Col 8
    CALL  GOTOXY
    CALL  CSTR_OUT  ; display warning
    db  'Not enough Disk Space to save the rankings.'
    db  'Press Any Key. ',0
    MOV  AL,[Color].CL ; restore original color
    CALL  HIDE_CUR
    CALL  ERR_SOUND
    CALL  GET_CHAR
    RET

ENDP  FULL_ERR
ENDP  IS_FULL

.CODE
PROC  GET_PATH
    PUSH  AX
    PUSH  BX
    PUSH  CX
    PUSH  DX
    CMP  BYTE PTR [Path],0 ; is the Path empty
    JNZ  GETP
    ; if NO then display Path
    GETP:  push  [Path] ; get default drive
            MOV  AH,12h ; default function
            INT   21h   ; get default drive
            ADD  AL,65 ; convert to cap letter
            MOV  AH,'.' ; place '.' in path
            MOV  SI,Offset Path ; ptr to [Path] string
            MOV  [SI],AX ; place drive letter
            INC  SI      ; in path
            INC  SI      ; ptr to 3rd byte
            MOV  AL,'\' ; place backslash in
            MOV  [SI],AL ; 3rd byte of string
            INC  SI ; point to 4th byte
    --- get default path
    GETP1:  CALL  EDIT_PATH

G-123
POP DX
POP CX
POP BX
POP AX
RET

ENDP GET_PATH

Input = last path entered or default path in [Path]
Output = current path in Input

PROC EDIT_PATH
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH ES, BX
MOV BX, DS
MOV ES, BX
CALL PATH_TO_INPUT
; move Path str to Input
CALL PATH_MESS_TOP
; edit message
CALL PATH_MESS_BTM
; edit message

EDIT1: MOV AX, 0108h
CALL PATH_EDITOR
; edit this field
JC EDIT2
; if valid save path
JC EDIT1
; if Not valid loop
JMP SHORT EDT3
; edit path OK!

EDT2: CALL SHORT EDT3
; draw bottom box
STC
; carry flag = Esc key
POP DX
; restore registers
POP CX
POP BX
POP AX
RET

ENDP EDIT_PATH

PROC PATH_MESS_TOP
PUSH AX
PUSH CX
XOR AX, AX
CALL MENU_BOX
MOV CL, [Color]
MOV AX, 0208h
CALL GOTOXY
MOV AL, [Menu]
MOV [Color], AL
CALL CSTR_OUT
db 'Enter the directory path:', 0
MOV [Color], CL
POP CX
POP AX
RET
ENDP PATH_MESS_TOP

---remove all but letters from the field and convert into and ASCII string
Input = None
Output = None
Note: Fields are 14 bytes long but the last byte is always a hex 0
therefore the name fields can only have 13 letters.

PROC FILTER_FIELD
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV BX, Offset Input
MOV CX, 12
TRI1: MOV AL,'A'
CMP [BX], AL
JNC TRI3
TRI2: CALL DELETE_CHAR
DEC BX
JMP SHORT TRI4
TRI3: MOV AL,'Z'
CMP [BX], AL
JC TRI2
TRI4: INC BX
LOOP TRI1

---convert trailing spaces to hex 0.
MOV CX, 13
MOV BX, Offset Input + 12
MOV AX, 20h
TRI5: CMP AL, [BX]
JNZ TRI6
MOV [BX], AH
DEC BX
LOOP TRI5
TRI6: POP DX
POP CX
POP BX
POP AX
RET

--- delete a character at the cursor
PROC DELETE_CHAR
PUSH AX
PUSH BX
DEP1: MOV AX, [BX]
CMP AH, 0
JZ DEP2
MOV [BX], AH
INC BX
JMP SHORT DEP1
DEP2: MOV AH, '

G-125
MOV [BX], AH
POP BX
POP AX
RET

ENDP DELETE CHAR
ENDP FILTER FIELD

; Instructions for entering the path name.
; Input = None
; Output = None

PROC PATH MESS BTM
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX, 1500h ; row 21, column 0
CALL MENU BOX ; draw menu box
MOV AX, 166Ah ; row 22, column 13
CALL GOTOYX
MOV AL, [Color] ; get current color
MOV CL, AL ; store in CL
MOV AL, [Menu] ; set color = menu
MOV [Color], AL
CALL CSTR OUT
db 'Type the complete path name for the directory to be searched.
', 0
MOV AX, 1708h ; row 23, column 13
CALL GOTOYX
CALL CSTR OUT
db 'Press the <Enter> key to continue or the <Esc> key for 
', 0
db 'the Menu.
', 0
MOV [Color], CL ; restore orig. color
POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP PATH MESS BTM

; Copy path ASCII string in [Path] to [Input].
; Input = None
; Output = None
; AX-DX register saved.

PROC PATH TO INPUT
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
; fill [Input] with 68 spaces
MOV AX, DS
; Make ES = DS

G-126
MOV ES,AX
MOV CX,67
MOV BX,Offset Input
MOV AL, 0
MOV [BX],AL
MOV DI,BX
INC DI
MOV SI,BX
CLD
REP MOVSB

--- copy current [path] to [input]
MOV DI,Offset Input
MOV SI,Offset Path
CLD
EDTO: MOVSB
CMP BYTE PTR [SI],0
JNZ EDT0
CLC
POP DX
POP CX
POP BX
POP AX
RET

ENDP PATH_TO_INPUT

--- Get an ASCII string input from the keyboard.
Input = AX = Row/Column position on the screen
[Input] must contain the string to be edited before
calling this subroutine.
[Insert] < > 0 places the editor in the insert mode.
Output = AL = Exit 'Char'
ASCII string at [Input] in the data section.
BX-DX register saved
Note: the follow register hold the following local variables.
AL = Input Character
BX = ptr in [Input] string
CX = Row/Col cursor position
DX = Starting Row/Col position

PROC PATH_EDITOR
PUSH BX
PUSH CX
PUSH DX
MOV DX,AX
CALL GOTOYX
MOV BX,Offset Input
MOV AX,BX
CALL DSTR OUT

--- fine first space in string
MOV SI,0
PATA: INC SI
CMP SI,67
JNZ PATA

; save registers
;save row/column in DX
;set cursor position
;ptr to [Input] str.
;[Input] ptr to AX
;Display blanks
;zero to SI
;ptr to next char
;stop if no spaces
JZ PATB ; safety value
CMP BYTE PTR [BX + SI], ' ' ; is this a space?
JNZ PATA ; if NO check next char
PATB: MOV CX,SI ; offset to CX
ADD CX,DX ; row/column ptr to CX
ADD BX,SI ; advance BX pointer

--- beginning of input loop
CALL PATH_INSERT ; display insert status
PAT0: MOV AX,CX ; cursor position to AX
CALL GOTO_YX ; set cursor position
CALL GET_TEXT

--- Return key
CMP AL,0Dh ; is it a <return> ?
JZ PAT10 ; if yes exit.

--- Is it any other control character?
CMP AL,1Bh ; is it a control char?
JNC PAT5 ; jmp = not control char
CALL PATH_CONTROL_CHAR ; handle control char
JMP SHORT PAT0 ; get next character

--- Is it the <Esc> key?
PAT5: STC ; set carry flag
JZ PAT11 ; exit on <Esc> key

--- Filter unwanted characters
CALL CHAR_FILTER ; carry flag = not char
JC PAT0

--- Check the insert mode
PAT7: MOV AH,[Insert] ; get insert flag
CMP AH,0 ; is insert OFF? = 0
JZ PAT8 ; skip if turned off
CALL SHIFT_STR_RT ; move rest of str right

--- Place the character in the [Input] string.
PAT8: MOV [BX],AL ; place char in [Input]
MOV AX,BX ; prt rest of string
CALL DSTR_OUT ; display string from

--- See if 'end of string' is true.
XOR AH,AH
CMP AH,[BX+1] ; is 'end of string'? 
JZ PAT0 ; yes = do not move cursor
INC BX ; advance [Input] ptr.
INC CX ; advance cursor
JMP PAT0 ; if not continue input.

PAT10: CLC
PAT11: POP DX
POP CX
POP BX
RET

--- Display the status of the [Insert] flag to screen.

Input = None
Output = None
AX - DX registers saved

PROC PATH_INSERT

G-128
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV DL,[Color] ;set color for insert
MOV AL,[HL+][Color] ;string.
MOV AX, 0420h ;row col
CALL GOTOYX ;set cursor
XOR AX, AX ;zero AX
ADD AL, [insert] ;get insert flag
JNZ PAH1 ;< > 0 = insert mode
CALL CSTR_OUT ;clear insert from
                ;the screen.
JMP SHORT PAH2 ;send the following
PAH1: CALL CSTR_OUT ;send the following
db '"<Insert On>' ;"string to the screen
PAH2: MOV [Color], DL ;restore current color.
POP DX ;restore registers
POP CX
POP BX
POP AX
RET
ENDP PATH_INSERT

--- Check Control Characters

Input AL = Control Character
BX = ptr in [Input] string
CX = Row/Col cursor position
DX = Starting Row/Col position
OutPut jumps back to get another character.

PROC PATH_CONTROL_CHAR

--- Backspace key
CMP AL, 06h ;is it a Backspace key
JNZ CNN0 ;if not continue.
CALL BACKSPACE ;del char left of cur.

--- Insert key
CNN0: CMP AL, 16h ;is it the insert key?
JNZ CNN1 ;if not continue.
PUSH AX ;save Char
XOR AX, AX ;zero AX
ADD AL, [insert] ;get insert flag
JZ CNN0A ;if zero jump
MOV AL, AH ;make flag = 0
JMP SHORT CNN0B ;replace flag
CNN0A: DEC AL ;make flag = FFh
CNN0B: MOV [insert], AL ;replace flag
CALL PATH_INSERT ;display insert status.
POP AX ;restore Char

--- Home key
CNN1: CMP AL, 1h ;is it the Home key?
JNZ CNN2 ;if not continue.

G-129
MOV BX, Offset Input
MOV CX,DX

;----End key
CNN2:  CMP AL, 6h
JNZ CNN3
CALL END_STR

;----Delete key
CNN3:  CMP AL, 07h
JNZ CNN4
CALL DELETE

;----left arrow key
CNN4:  CMP AL, 13h
JNZ CNN6
CMP CX,DX
JZ CNN6
DEC BX
DEC CX

;----right arrow key
CNN6:  CMP AL, 4
JNZ CNN8
XOR AH,AH
CMP BYTE PTR [BX+1],0
JZ CNN8
INC BX
INC CX

CNN8:  RET
ENDP PATH_CONTROL_CHAR

;----move cursor to end of string
PROC  END_STR
PUSH AX

CON2A: MOV AX,[BX]
        CMP AH,0
        JZ CON2B
        INC BX
        INC CX
        JMP SHORT CON2A

CON2B: POP AX
        RET
ENDP END_STR

;----Insert a character at the cursor.
PROC  SHIFT_STR_RT
PUSH AX
        ;save new character
PUSH BX
        ;save str pointer
MOV AL,[BX]
        ;load char to be moved
INC BX
        ;ptr to the next char
SHI1:  MOV AH,[BX]
        CMP AH,0
        JZ SHI2
        MOV [BX],AL
        MOV AL,AH
        INC BX
        MOV AL,[BX]
        INC BX
        MOV AL,[BX]
        INC BX
INC BX ;ptr for next new char
JMP SHORT SHI1 ;loop until end of str.
SHI2: POP BX ;restore str pointer
POP AX ;restore original ptr
RET
ENDP SHIFT_STR_RT

;----- delete a character at the cursor
PROC DELETE
PUSH AX
PUSH BX
DEL1: MOV AX,[BX] ;read ptr BX and BX + 1
CMP AH,0 ;is it the end of str?
JZ DEL2 ;if yes then done.
MOV [BX],AH ;place BX + 1 in BX
INC BX ;point to next byte
JMP SHORT DEL1 ;loop until end of str.
DEL2: MOV AH,' ' ;place a <space> at
MOV [BX],AH ;end of the string.
POP BX ;restore original ptr
MOV AX,BX ;str pointer to AX
CALL DSTR_OUT ;display string
POP AX
RET
ENDP DELETE

;----- delete a character to the left of the cursor
PROC BACKSPACE
PUSH AX
MOV AX,Offset Input ;is the cursor at the
CMP AX,BX ;beginning of the string?
JZ BA3 ;if yes ignore backspace
DEC BX ;line pointer left
DEC CX ;cursor left
PUSH BX ;save original str ptr
BA1: MOV AX,[BX] ;read ptr BX and BX + 1
CMP AH,0 ;is it the end of str?
JZ BA2 ;if yes then done.
MOV [BX],AH ;place BX + 1 in BX
INC BX ;point to next byte
JMP SHORT BA1 ;loop until end of str.
BA2: MOV AH,' ' ;move <space> to AH
MOV [BX],AH ;place in last position
POP BX ;restore original ptr
MOV AX,CX ;row/column to AX
CALL GOTOXY ;set cursor position
MOV AX,BX ;str pointer to AX
CALL DSTR_OUT ;display string
BA3: POP AX
RET
ENDP BACKSPACE

G-131
; Filter out unwanted ASCII characters and capitalize letters
; Input = Char in AL
; Output = Carry Flag = not a good character. get another!

PROC CHAR_FILTER
    AND AL,7FH ; make 0 - 127 ASCII.
    CMP AL,''
        ; is it a control char?
    JC CHAR1
        ; if yes, get next char.
    CMP AL,'a'
        ; is char a small letter
    JC CHAR0
        ; if not, Ok continue.
    AND AL,OFh
        ; change to capital char

CHAR0:  CLC
        ; clear carry flag

CHAR1:  RET
ENDP CHAR_FILTER
ENDP PATH_EDITOR

; Check [Input] to see if the path is Ok!
; Input = AX = Assumed [Input] hold a Path
; Output = Carry flag is not a valid path name
; AX-DX register saved.

PROC CHECK_PATH
    PUSH AX
    PUSH BX ; save registers
    PUSH CX
    PUSH DX
    MOV AX,DS ; Make ES = DS
    MOV ES,AX

    ; remove all leading spaces
    MOV BX,Offset Input ; ptr to Input string
    CHE0: CMP BYTE PTR [BX],''
        ; is leading space? 
    JNZ CHE1
        ; if NO continue
    MOV CX,68
    MOV DI,BX
    MOV SI,BX
    INC SI
    MOV BYTE PTR [SI],'
    CLD
    REP MOVSB
    JMP SHORT CHE0
    ; check for leading space

    ; convert first ASCII space to a hex zero EndofStr marker
    CHE1: MOV CX,68
        ; max string length
    MOV BX,Offset Input - 1

    CHE2: INC BX
    CMP BYTE PTR [BX],''
        ; is it a space?
    JC CHE4
        ; exit if char < '
    LOOPNZ CHE2
        ; is NO loop

    ; remove trailing back slash
    DEC BX
    CMP BYTE PTR [BX],'
        ; is it a back slash?
    JZ CHE3
        ; if Yes remove from str
    INC BX
        ; if NO leave in str

    ; place : after drive name?
CHES:  MOV AX,BX                        ;ptr in str to ax
        SUB AX,Offset Input                ;string length in AX
        JZ  CHE4                           ;if OK! goto next test
        CMP AX,3                           ;is less than 3?
        JNC CHE4                           ;if NO goto next test
        MOV AX,003Ah                        ;3Ah = ':'
        MOV BX,Offset Input + 1            ;ptr to 2nd byte
        INC BX

CHE4:  MOV [BX],CH                      ;mark EndOfStg = 0
        ---is the path valid
        CALL IS_PATH                       ;is path valid?
        JNC CHE5                           ;NOT carry = OK!
        CALL PATH_ERROR                    ;display error message
        STC                                 ;set cf = error
        JMP SHORT CHE7                     ;-ave valid path strn in [Path]

CHE5:  MOV SI,Offset Input              ;source offset
        MOV DI,Offset Path                 ;destination offset
        CLD                                ;auto inc DI and SI

CHE6:  MOVSB                            ;copy one byte
        CMP BYTE PTR [SI],0                ;is next char = 0
        JNZ CHE6                           ;copy bytes
        XOR AL,AL                          ;zero = EndOfString
        MOV [DI],AL                         ;clear carry flag

CHE7:  POP DX                           ;--is this a Valid path?
        POP CX                             ;Input = ASCIIZ drive/directory string in [Input]
        POP BX                             ;Output = carry flag in not a valid path
        POP AX                             ;AX - DX registers saved
        RET                                ;PROC IS_PATH

        ---copy string to [Search]
        MOV SI,Offset Input                ;source offset
        MOV DI,Offset Search               ;destination offset
        CLD                                ;auto inc DI and SI

ISP1:  MOVSB                            ;copy one byte
        CMP BYTE PTR [SI],0                ;is next char = 0
        JNZ ISP1                           ;copy bytes
        ---place "." at end of string
        MOV AX,092h4                      ;DX = AX,AL
        MOV [DI],AX                        ;INC DI
INC DI
MOV AX,".*"
MOV [DI],AX
INC DI
INC DI
XOR AX,AX
MOV [DI],AX

; see if path is OK!
MOV DX,Offset Search ; ptr to ASCII string
MOV AX,4E00h ; Find function no.
MOV CX,0010h ; directory search
INT 21h ; do search
CMP AL,3 ; is path BAD?
CLC ; clear carry flag
JNZ ISP2 ; OK! if not 3
STC ; set error flag
ISP2: POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP IS_PATH

;---Display Path Error message.
; Input = None
; Output = None
; AX - DX registers saved
PROC PATH_ERROR
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV AL,0h ; replace hex 0 with
MOV [BX],AL ; a space
MOV CL,Color ; save current color
MOV AL,[Warning] ; warning color
MOV [Color],AL ; set color
MOV AX,0222h ; row 5 Col 7
CALL GOTOYX ; set cursor
CALL CSTR_OUT ; display warning
db 'Error: Invalid path. Press Any Key.',0
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR ; wait for keyboard key
MOV AL,[Menu] ; menu color
MOV [Color],AL ; set color
MOV AX,0222h ; row 5 Col 7
CALL GOTOYX ; set cursor
CALL CSTR_OUT ; clear warning
db 'Press Any Key',0
MOV [Color],CL ; restore original Color
POP DX ; restore registers

G-134
Draw the title screen and input the users name.

Input = None
Output = None

PROC GET_ID
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    MOV BX,DS
    MOV ES,BX
    XOR AX,AX
    CALL MENU_BOX
    CALL ID MESSAGE

    ID0: MOV AL,[Menu]
    MOV [Color],AL
    MOV AX,0115h
    CALL GOTOXY
    CALL CSTR_OUT
    db 'Please enter the ID number: ',0
    --- set color or Edit session
    MOV AL,[Normal]
    MOV [Color],AL
    --- set default ID string to ASCII zeros
    MOV BX,Offset ID
    MOV DI,BX
    MOV AX,3030h
    MOV [DI],AX
    XOR AH,AH
    INC DI
    INC DI
    MOV [DI],AX
    --- edit ID string
    MOV AX,0132h
    CALL NUMBER_EDITOR
    --- Get an ASCII string input from the keyboard.
    POP DX
    POP CX
    POP BX
    POP AX
    RET
ENDP GET_ID
Input = AX = Row/Column position on the screen
[ID] must contain the string to be edited before
calling this subroutine.
Output = ASCII string at [ID] in the data section.
BX-DX register saved
Note: the follow register hold the following local variables.
AL = Input Character
BX = ptr in [ID] string
CX = Row/Col cursor position
DX = Row/Col position
IMPORTANT: IS_ZERO traps searches for "000". This is not a valid ID#.
The 000 line contains the files constant variables.
[25%] [Mean] [75%] for each dimension.

PROC NUMBER EDITOR
PUSH BX ;save registers
PUSH CX
PUSH DX
PUSH DS
PUSH ES
MOV DX,AX ;row/column in DX
CALL GOTOYX ;set cursor position
MOV BX,Offset ID ;ptr to [ID] str.
MOV AX,BX ;[ID] ptr to AX
CALL DSTR_OUT ;Display zeros
MOV CX,DX ;row/column ptr to CX
:----beginning of input loop
NUE0: MOV AX,CX ;cursor position to AX
CALL GOTOYX ;set cursor position
CALL GET_CHAR ;wait for keybd input
:----Return key
CMP AL,0Dh ;is it a <return>?
JNZ NUE3 ;if NO goto next test
CALL IS ZERO ;is the ID number 0?
JMP SHORT NUE11 ;if = Yes DoNot search
:----Is it any other control character?
NUE3: CMP AL,18h ;is it a control char?
JNC NUE7 ;jmp = not control char
:----check for Backspace key
CMP AL,08h ;is it a Backspace key
JNZ NUE4 ;if not continue.
MOV AL,13h ;convert to left arrow.
:----left arrow key
NUE4: CMP AL,13h ;is it a left arrow key
JNZ NUE5 ;if not continue.
CMP CX,DX ;beginning of the string?
JZ NUE5 ;yes = beg. of line
DEC BX ;so loop will continue.
DEC CX
:----right arrow key
NUE5: CMP AL,4 ;is it Rt Arrow key?
JNZ NUE6 ;if not jump.

G-136
PUSH AX ; save char
MOV AX,[BX] ; check for end of str.
CMP AH,0 ; zero = end of string
POP AX ; restore char
JZ NUE6 ; if = 0 no right
INC BX ; advance pointer
INC CX
NUE6: JMP SHORT NUE0 ; get next character

----------

NUE7: STC ; set carry flag
JZ NUE11 ; edit if <Esc> key

----------

NUE8: MOV [BX],AL ; filter unwanted characters
MOV AX,BX ; row/coll to AX
CALL DSTR_OUT ; display string from

----------

NUE9: CMP AL,30h ; is it < ASCII 0 ?
JC NUE0 ; if YES get another.
CMP AL,3Ah ; is it an ASCII digit?
JNC NUE0 ; clear CF = not digit

----------

NUE10: MOV [BX],AL ; place char in [ID]
INC BX ; advance [ID] ptr.
INC CX ; advance cursor
JMP NUE0 ; if not continue input.

----------

NUE11: POP ES
POP DS
POP DX
POP CX
POP BX
RET

ENDP NUMBER_EDITOR

----------

IS ZERO
PROC IS ZERO

----------

PROC IS ZERO

----------

PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
MOV BX,Offset ID
MOV CX,3
MOV AL,30h

ISZ1: CMP [BX],AL ; is byte = 0?
JNZ ISZ2 ; not zero OK! exit
INC BX ; ptr to next byte
LOOP ISZ1 ; look at next byte

----------

G-137
CALL ZERO_MESS ;NO zero ID numbers
STC
JMP SHORT ISZ3 ;Exit on ID Error
CALL ZERO_MESS ;Enter error no search
CALL ZERO_MESS ;Inform user NO 0 ID's
ISZ2: CLC ;OK ID number
ISZ3: POP DX
POP CX
POP BX
POP AX
RET

; Input = none
; Output = none
PROC ZERO_MESS
PUSH AX
PUSH BX
PUSH CX
PUSH DX
MOV CL,[Color] ;Store original Color
MOV AL,[Warning] ;Warning color
MOV [Color].AL ;Set color
MOV AX,0109h ;Row 3/Col 12
CALL GOTOXY ;Set cursor
CALL CSTR_OUT ;Display warning
db "A valid ID numbers must be larger than '000'. Press Any "
db 'Key.',0
MOV [Color].CL ;Restore original color
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
CLC ;Clear CF = continue
POP DX
POP CX
POP BX
POP AX
RET
ENDP ZERO_MESS
ENDP IS_ZERO

; Instructions for entering the users ID number.
; Input = None
; Output = None
PROC ID_MESSAGE
PUSH AX ;Save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,1500h ;Row 21, column 0
CALL MENU_BOX ;Draw menu box
MOV AX,160Ch ;Row 22, column 13
CALL GOTOXY
MOV AL,[Color] ; get current color
MOV CL,AL ; store in CL
MOV AL,[Menu] ; set color = menu
MOV [Color],AL
CALL CSTR_OUT
db 'Type the ID Number and press the <Enter> key to continue',0
MOV AX,170Ch ;row 23,column 13
CALL GOTOXY
CALL CSTR_OUT
db 'or press the <Esc> key to return to the Main Menu.',0
MOV AX,0207h ;row 23,column 13
CALL GOTOXY
CALL CSTR_OUT
db 'A valid ID# must contains three numerical digits.'
db 'Example: 1 = 001',0
MOV [Color],CL ; restore orig. color
POP DX ; restore registers
POP CX
POP BX
POP AX
RET

ENDP ID_MESSAGE