

AD-A273 353



Technical Report 978

Micro Computer Feedback Report for the Strategic Leader Development Inventory

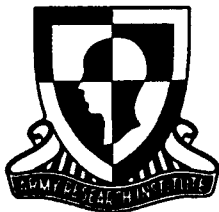
James E. Hopkins
U.S. Army Research Institute

May 1993

93-29316



17298



**United States Army Research Institute
for the Behavioral and Social Sciences**

Approved for public release; distribution is unlimited.

93 11 30004

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction
of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Acting Director

Research accomplished under contract
for the Department of the Army

Battelle, Inc.

DTIC QUALITY INSPECTED 5

Technical review by

Thomas O. Jacobs
Edgar Johnson

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

NOTICES

DISTRIBUTION: Primary distribution of this report has been made by ARI. Please address correspondence concerning distribution of reports to: U.S. Army Research Institute for the Behavioral and Social Sciences, ATTN: PERI-POX, 5001 Eisenhower Ave., Alexandria, Virginia 22337-5600.

FINAL DISPOSITION: This report may be destroyed when it is no longer needed. Please do not return it to the U.S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 1993, May	3. REPORT TYPE AND DATES COVERED Final Jun 92 - Aug 92		
4. TITLE AND SUBTITLE Micro Computer Feedback Report for the Strategic Leader Development Inventory		5. FUNDING NUMBERS DAAL03-91-C-0034 62785A 791 2405 TCN 92-272		
6. AUTHOR(S) Hopkins, James E. (ARI)				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Institute for the Behavioral and Social Sciences ATTN: PERI-RO 5001 Eisenhower Avenue Alexandria, VA 22333-5600		8. PERFORMING ORGANIZATION REPORT NUMBER ARI Technical Report 978		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709-2211		10. SPONSORING / MONITORING AGENCY REPORT NUMBER --		
11. SUPPLEMENTARY NOTES Task was performed under a Scientific Services Agreement issued by Battelle, Research Triangle Park Office, NC 27709. Contracting Officer's Representative, Thomas O. Jacobs.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE --		
13. ABSTRACT (Maximum 200 words) 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.				
14. SUBJECT TERMS Strategic Leader Development Inventory Computer Printer Control Language Assembler Leadership ASM		15. NUMBER OF PAGES 172		
		16. PRICE CODE --		
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited	

Technical Report 978

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.

FOREWORD

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

	Page
INTRODUCTION	1
SLDI DATA FILES	1
FEEDBACK DOCUMENTATION	3
APPENDIX A. SAMPLE FEEDBACK GRAPHS	A-1
B. DESCRIPTION OF SLDI	B-1
C. CONVERTING THE POS FILE	C-1
D. FORMAT FOR FEEDBACK DATA FILES	D-1
E. LOCATING WORDPERFECT'S PRINTER PORT	E-1
F. FORMAT FOR FEEDBACK VARIABLE STRINGS	F-1
G. SOURCE CODE FOR THE FEEDBACK PROGRAM	G-1

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

ID Number: 101

Success Factors

Scored: 08/25/92

DIMENSIONS:

	Below Average	Better Than Most	The Best
	1.....2.....3.....4.....5		
Raw Scores:			

Better Than Host

Better Than Host

The Best of

CONCEPTUAL EFFECTIVENESS

	Self	Peers	Superiors	Subordinates
1. I am satisfied with my work.	4.0	4.0	4.0	4.0
2. I like my job.	4.0	4.0	4.0	4.0
3. I feel good about myself.	4.0	4.0	4.0	4.0
4. I am proud of my work.	4.0	4.0	4.0	4.0
5. I am satisfied with my life.	4.0	4.0	4.0	4.0
6. I like my supervisor.	4.0	4.0	4.0	4.0
7. I like my coworkers.	4.0	4.0	4.0	4.0
8. I feel good about my future.	4.0	4.0	4.0	4.0
9. I am satisfied with my salary.	4.0	4.0	4.0	4.0
10. I am satisfied with my benefits.	4.0	4.0	4.0	4.0
11. I am satisfied with my company.	4.0	4.0	4.0	4.0
12. I am satisfied with my industry.	4.0	4.0	4.0	4.0
13. I am satisfied with my location.	4.0	4.0	4.0	4.0
14. I am satisfied with my schedule.	4.0	4.0	4.0	4.0
15. I am satisfied with my training.	4.0	4.0	4.0	4.0
16. I am satisfied with my growth opportunities.	4.0	4.0	4.0	4.0
17. I am satisfied with my work-life balance.	4.0	4.0	4.0	4.0
18. I am satisfied with my overall experience.	4.0	4.0	4.0	4.0

Raw Scores: 1.....2.....3.....4.....5

TEAM BUILDING

Self	Peers	Superiors	Subordinates
[REDACTED]	[REDACTED]	[REDACTED]	[REDACTED]

Raw scores: 1.....2.....3.....4.....5

STRATEGIC POTENTIAL

	Self	Peers	Superiors	Subordinates
1. <i>Self</i>	100	100	100	100
2. <i>Peers</i>	100	100	100	100
3. <i>Superiors</i>	100	100	100	100
4. <i>Subordinates</i>	100	100	100	100

[illegible]

PERFORMANCE UNDER STRESS

Self
Peers
cognitive

DATE	DESCRIPTION	AMOUNT	BALANCE
1900			
1901			
1902			
1903			
1904			
1905			
1906			
1907			
1908			
1909			
1910			
1911			
1912			
1913			
1914			
1915			
1916			
1917			
1918			
1919			
1920			
1921			
1922			
1923			
1924			
1925			
1926			
1927			
1928			
1929			
1930			
1931			
1932			
1933			
1934			
1935			
1936			
1937			
1938			
1939			
1940			
1941			
1942			
1943			
1944			
1945			
1946			
1947			
1948			
1949			
1950			
1951			
1952			
1953			
1954			
1955			
1956			
1957			
1958			
1959			
1960			
1961			
1962			
1963			
1964			
1965			
1966			
1967			
1968			
1969			
1970			
1971			
1972			
1973			
1974			
1975			
1976			
1977			
1978			
1979			
1980			
1981			
1982			
1983			
1984			
1985			
1986			
1987			
1988			
1989			
1990			
1991			
1992			
1993			
1994			
1995			
1996			
1997			
1998			
1999			
2000			
2001			
2002			
2003			
2004			
2005			
2006			
2007			
2008			
2009			
2010			
2011			
2012			
2013			
2014			
2015			
2016			
2017			
2018			
2019			
2020			
2021			
2022			
2023			
2024			
2025			
2026			

STRATEGIC LEADER DEVELOPMENT INVENTORY

ID Number: 101

Failure Factors

Scored: 08/24/92

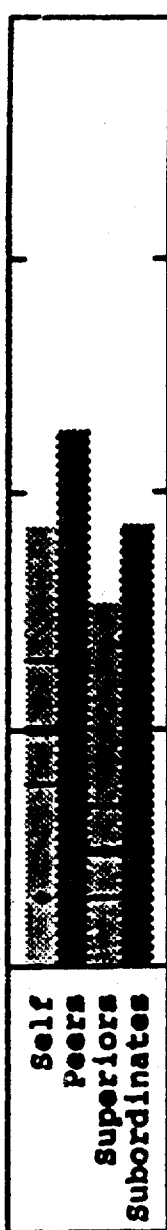
DIMENSIONS:

← Never

Occasionally

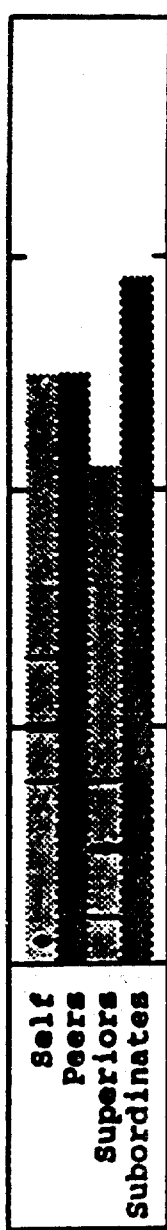
Always →

Raw Scores: 1.....2.....3.....4.....5



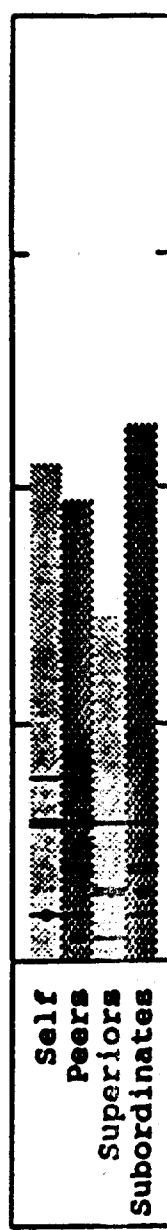
LIMITED PERSPECTIVE

Raw Scores: 1.....2.....3.....4.....5



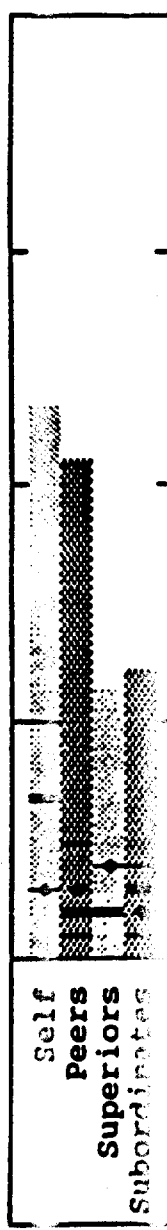
EGOCENTRIC

Raw Scores: 1.....2.....3.....4.....5



CAREERIST

Raw Scores: 1.....2.....3.....4.....5



UNPROFESSIONAL

♦ = Score | = 25% | = 50% | = 75% ♦ and 2 = average

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.

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

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:

O 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.

O 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.

O 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:

O 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.

O 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 judgements 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.

O 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.

O 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 Columns	New Columns	ID number	range: '001' to '499'
1-3	1-3		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.)
35-36	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:

DOS File name: *.POS = Positive Factors data file.

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 Subordinates)	
10-12	Self on Strategic Potential (Self vs Peers)	
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

Data File Columns: (3 ascii bytes)		[FileBuf] (3 ascii bytes)	Low	[PerCnt]			25%	75%
1-3	ID number	0 - 2	(ascii words)					
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	

APPENDIX G:
Source Code for the FeedBack Program

```

;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 FBB.ASM           ;printer subroutines
INCLUDE FBM.ASM           ;menu procedures
INCLUDE FBT.ASM           ;title procedures
INCLUDE FBU.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 diminisions
BarPos dw 0101h           ;position of hllite bar

```

```

;
;Data file variables
PosTyp db    '.POS',0           ;Positive dim data type
NegTyp db    '.NEG',0           ;Derailment dim data
FITTyp db    '.SLD',0           ;file type
FileNa db    '???????.SLD',0    ;ASCIIZ file name
SearNa db    '???????.SLD',0    ;ASCIIZ 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:
Filbuf 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,I,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 db    82 DUP (0h)         ;input ASCIIZ string.
Search db   82 DUP (0h)
Digit db    1                   ;0 = OFF Other = ON
Insert db   0                   ;0 = OFF Other = ON
EndFld 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
Exit: CALL RESTORE_VIDEO                        ;restore users settings

```

```

MOV     AL,[ErrCode]           ;load errorlevel number
MOV     AH,4Ch                 ;Exit function number
INT     21h                    ;return to DOS
;-----End of Main porcedure for FeedBack
Error:  CALL  DISPLAY_ERROR      ;show DOS extend error
        JMP   SHORT Exit
;-----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,'&k0S',0 ;space + 19 comp. spaces
NextUn db 1Bh,'&k2S',1Bh,'&a+2C',1Bh,'&k0S',0 ;2 compressed spaces
HalfSp db 1Bh,'&k2S',1Bh,'&a+1C',1Bh,'&k0S',0 ;1 compressed spaces
BackSp db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S',0 ;1 compressed spaces
FullBk db 1Bh,'&k2S',1Bh,'&a-2C',1Bh,'&k0S',0 ;1 compressed spaces

```

```

;--- HP PCL strings used by printing procedures
Heder db 1Bh,'(s3BSTRATEGIC LEADER DEVELOPMENT INVENTORY',0

```

```

IDstr db 'ID Number: '
ID db '123',0 ;3 digit ASCII number

```

```

PosT db 1Bh,'&a+23C',1Bh,'&f0SSuccess Factors',1Bh,'&f1S',1Bh,'&a+2R'
db 1Bh,'&a+3C',1Bh,'(s0B' ;post.row/col & unbold
db '|',1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S~Q'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 'Below Average Better Than Most The Best '
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S~P'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S|'
db 1Bh,'(s3B',0 ;bold and EndOfString

```

```

NegT db 1Bh,'&a+23C',1Bh,'&f0SFailure Factors',1Bh,'&f1S',1Bh,'&a+2R'
db 1Bh,'&a+3C',1Bh,'(s0B' ;post.row/col & unbold
db '|',1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S~Q'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 'Never Occasionally Always '
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S-'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S~P'
db 1Bh,'&k2S',1Bh,'&a-1C',1Bh,'&k0S|'
db 1Bh,'(s3B',0 ;bold and EndOfString

```

```

DTstr db 'Scored: '
Date db '07/24/92',0 ;file date

```

```

Distr db 'DIMENSIONS:',1Bh,'(s0B',0 ;unBold, EndStMarker

```

```

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 = "| "
UnlLt db 177,0 ;177 = " "
UnlDk db 178,0 ;178 = " "
TenLT db 13 DUP (177),0
TenDk db 13 DUP (178),0
;---An HP PCL string used by Initialize_HP procedure
Init db 1Bh,'E' ;reset printer
db 1Bh,'&J1O' ;landscape
db 1Bh,'&k0S' ;10.0 cpi
db 1Bh,'(10U' ;PC-8 symbol set
db 1Bh,'(s0P',0 ;Fixed spacing
;---An HP PCL string used by Restore_HP procedure
Rest db 1Bh,'&J0O' ;portrait
db 1Bh,'(8U' ;Roman-8 symbol set
db 1Bh,'(s1P' ;Proportional spacing
db 1Bh,'E',0 ;reset printer
;---An HP PCL string used by HPGOTOYX procedure
GoTo db 1Bh,'&a' ;set hp laser to
col dw 0 ;ASC II column number
db 'C',1Bh,'&a' ;set hp laser to
row dw 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,'&f0S Raw Scores: 1' ;starting Push
;ticks and "2"
db 1Bh,'&k2S - - - - - ',1Bh,'&k0S' ;compressed mode
db 1Bh,'&f0S',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&f0S',1Bh,'&a+6R',193,1Bh,'&f1S2' ;bottom tick and "2"
;ticks and "3"
db 1Bh,'&k2S - - - - - ',1Bh,'&k0S' ;compressed mode
db 1Bh,'&f0S',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&f0S',1Bh,'&a+6R',193,1Bh,'&f1S3' ;bottom tick and "3"
;ticks and "4"
db 1Bh,'&k2S - - - - - ',1Bh,'&k0S' ;compressed mode
db 1Bh,'&f0S',0Ah,194,1Bh,'&f1S' ;top tick mark
db 1Bh,'&f0S',1Bh,'&a+6R',193,1Bh,'&f1S4' ;bottom tick and "4"
;ticks and "5"
db 1Bh,'&k2S - - - - - ',1Bh,'&k0S5' ;compress and "5"
db 1Bh,'&f1S',0Ah ;ending Pop + line feed = next line
;top line
db 1Bh,'&f0S',218,13 DUP (196),194,49 DUP (196),191
db 1Bh,'&f1S',0Ah ;next line
;self line
db 1Bh,'&f0S',179,' Self ',179,1Bh,'&a+49C',179
db 1Bh,'&f1S',0Ah ;next line
;peers line
db 1Bh,'&f0S',179,' Peers ',179,1Bh,'&a+49C',179
db 1Bh,'&f1S',0Ah ;next line

```

```

;superiors line
db 1Bh,'&f0S',179,' Superiors ',179,1Bh,'&a + 49C',179
db 1Bh,'&f1S',0Ah ;next line
;subordinates line
db 1Bh,'&f0S',179,'Subordinates ',179,1Bh,'&a + 49C',179
db 1Bh,'&f1S',0Ah ;next line
;bottom line
db 192,13 DUP (196),193, 49 DUP (196),217,0 ;EndOfString marker
;---An HP PCL string to draw a graphing box and the present cursor position
Inform db 1Bh,'&f0S',218,63 DUP (196),191
db 1Bh,'&f1S',0Ah ;next line
;self line
db 1Bh,'&f0S',179
db ' ^D = Score | = 25% || = 50% | = 75% and = Range '
db 179,1Bh,'&f1S',0Ah ;next line
;bottom line
db 192, 63 DUP (196),217,0 ;EndOfString marker
;---An HP PCL string to label Postive 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 Postive 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 Postive 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 Postive Dimension #4
Po. 4 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

```

```

;---An HP PCL string to label Derailment Dimension #4
Neg4 db 1Bh,'(s3B' ;bold ON
      db 'UNPROFESSIONAL'
      db 1Bh,'(s0B' ;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 [DataEr] = 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, unranded
      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 [DataEr],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 ;is 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 bock
      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 ;read char from key-
      INT 21h ;board buffer.
      JZ RD3 ;NO key pressed continue
      CMP AL,1Bh ;was it the <ESC> key?
      JNZ RD3 ;if NO continue
      CALL ESC_YN ;if YES inform user
      JC RD4 ;carry flag = abort
;---loop until each column is ranked.
RD3: LOOP RD0 ;loop until cx = 0
      MOV AL,0FFh ;[ranked] < > 0 = TRUE
      MOV [Ranked],AL ;mark file ranked
;---If no errors write data line to file
      CMP BYTE PTR [DataEr],0 ;any errors found?
      JNZ RD4 ;if Yes exit else

```

```

        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   RK5
      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
      MOV   AL,[Warning]                  ;warning color
      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,020Bh ;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]              ;get memory segment
    MOV     ES,AX                    ;place in ES register
    MOV     AX,4900h                 ;release function no
    INT     21h                      ;release memory block
    JC     REL1                      ;if No error continue
;---initialize variable
    XOR     AX,AX                    ;zero to register
    MOV     [VarSeg],AX              ;set memory back to 0
    CLC                                ;clear carry flag
REL1: POP     ES
    POP     DX                      ;restore registers
    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                      ;continue if no error
    JMP     CRV9                      ;exit on DOS error
CRV0: MOV     AX,[MaxNo]              ;get number of ID's
    MOV     CL,3                      ;no. bits to shift
    SHR     AX,CL                     ;paragraph = ID/8 + 2
    MOV     BX,AX                     ;paragraph count to BX
    INC     BX                        ;get an extra paragraph
    INC     BX                        ;get an extra paragraph
    MOV     AH,48h                    ;allocate mem function
    INT     21h                      ;request memory block
    JC     CRV2                      ;jump if memory error.
    MOV     [VarSeg],AX              ;base address of seg
    JMP     SHORT CRV8                ;normal exit of proc.
CRV2: MOV     CL,[Color]              ;save original color
    MOV     AL,[Warning]              ;warning color
    MOV     [Color],AL               ;set color
    MOV     AX,0207h                 ;row/Col
    CALL    GOTOYX                    ;position cursor
    CALL    CSTR_OUT                  ;send string to screen
    db      ' Not enough memory to rank the variables. '
    db      'Press Any Key to Continue. ', 0

```

```

MOV     [Color],CL           ;restore original color
CALL    HIDE_CUR             ;hide cursor off screen
CALL    ERR_SOUND
CALL    GET_CHAR             ;wait for key is pressed
STC                                           ;set carry flag = error
JMP     SHORT CRV9
CRV8:   CLC                   ;clear carry flag
CRV9:   POP     ES
        POP     DX             ;restore registers
        POP     CX
        POP     BX
        RET
ENDP    GET /AR_BLK

```

```

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

```

```

PROC    CLEAR_FILBUF
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    ES
:-----fill [FilBuf] with 192 hex 0's
        MOV     AX,DS           ;Make ES = DS
        MOV     ES,AX
        MOV     CX,95           ;max number of words
        MOV     BX,Offset FilBuf ;pointer to ASCIIZ str
        XOR     AX,AX           ;hex 0's to AX register
        MOV     [BX],AX        ;0 to first word of str
        MOV     DI,BX           ;DI = pointer to next
        INC     DI              ;ptr to next word
        INC     DI
        MOV     SI,BX           ;SI = ptr to hex 0's
        CLD                     ;auto inc DI and SI
        REP     MOVSW           ;fill string with 0's
        CLC                     ;clear carry flag
        POP     ES
        POP     DX             ;restore registers
        POP     CX
        POP     BX
        POP     AX
        RET
ENDP    CLEAR_FILBUF

```

```

:
: Clear Percentile variables.
:   Input = None
:   Output = None 192 hex 0 to [PerCnt]
:

```

PROC CLEAR PERCNT

```

PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH ES

```

-----fill [FilBuf] with 192 hex 0's

```

MOV AX,DS           ;Make ES = DS
MOV ES,AX
MOV CX,96           ;max number of words
MOV BX,Offset PerCnt ;pointer to ASCIIZ str
XOR AX,AX           ;hex 0's to AX register
MOV [BX],AX         ;0 to first word of str
MOV DI,BX           ;DI = pointer to next
INC DI              ;ptr to next word
INC DI
MOV SI,BX           ;SI = ptr to hex 0's
CLD                 ;auto inc DI and SI
REP MOVSW           ;fill string with 0's
CLC                 ;clear carry flag
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 # 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 [FilBuf] (loop count x 3)

```

PROC READ VAR

```

PUSH BX
PUSH CX
PUSH DX
PUSH DS
XOR AX,AX           ;zero AX
MOV BX,AX           ;destination pointer
CMP [VarSeg],AX     ;is VarSeg assigned?
JZ RVR9             ;if not assigned go on

```

-----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

```

MOV CX,Offset FilBuf	;start of file buffer
ADD AX,CX	;add buffer to start
MOV DX,AX	;save offset in [FilBuf]
CALL GOTO_TOP	;reset file pointer
JC RVR9	;exit on DOS error
;-----set default ID string to ASCII zeros	
MOV DI,Offset ID	;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	
RVR1: CALL CLEAR_FILBUF	;hex 0's to file buffer
CALL READ_LINE	;1 line from data file
JNC RVR2	;not EndOfFile
MOV AL,0FFh	;mark EndOfFile true
MOV [EOF],AL	; < > 0 = True
;-----is this a '000' data line?	
RVR2: MOV CX,3	;loop counter
MOV DI,Offset ID	;ptr to ID number
MOV SI,Offset FilBuf	;ptr to data file line
CLD	;auto inc DI and SI
REPZ CMPSB	;are the bytes = ?
JZ RVR5	;skip if ID = '000'
;-----is this an ID number line?	
MOV SI,Offset FilBuf	;ptr to data file line
MOV CX,3	;loop counter
RVR4: MOV AL,[SI]	;get first byte
CMP AL,'0'	;is it < ASCII 0
JC RVR5	;if Yes read next line
CMP AL,':'	;is it a digit?
JNC RVR5	;if No read next line
INC SI	;point to next byte
LOOP RVR4	;check next byte
;-----is this the EndOfFile 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	;ptr to buffer section
MOV AX,2000h	;ASCII <space>, hex 0
CMP [SI],AH	;is it a space?
JZ RVR5	;is yes skip variable
CMP [SI],AL	;is it past EndOfFile?
JZ RVR5	;if yes skip variable
CALL VAR_TO_BLK	;move variable to block
INC BX	;ptr to next word
INC BX	
;-----is this the last line ?	
RVR5: XOR AL,AL	;zero AX register
CMP AL,[EOF]	;is EndOfFile TRUE?
JZ RVR1	;False = get next line

```

RVR9: MOV    AX,BX                ;return no of variables
      SHR    AX,1                ;number of var found
      CLC                                ;clear carry flag
      POP    DS
      POP    DX                ;restore registers
      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 [FilBuf]
:  Output = None
PROC  VAR_TO_BLK
      PUSH    AX
      PUSH    BX
      PUSH    CX
      PUSH    DX
      PUSH    ES
      MOV     AX,[VarSeg]        ;ptr to base of memblk
      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,'5'            ;check for rounding
      CMP     [SI+2],AL         ;round the number ?
      JC      VTB1              ;# < 5 = no round
:-----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                                ;clear carry flag
VTB2: POP     ES
      POP     DX                ;restore registers
      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

If A[J] > A[J+Gap] then

Begin

Temp := A[J];

A[J] := A[J+Gap];

A[J+Gap] := Temp;

J := J-Gap;

End

Else J := 0;

End;

End;

Gap := Gap DIV 2;

End;

End;

The follow 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                ;I is stored in CX
INC     CX                   ;I = Gap + 1
VARS2:  MOV     BX,CX         ;J in BX
SUB     BX,AX                ;J = I - Gap
JZ      VARS3                ;skip if J = 0
JC      VARS3                ;skip if J is < 0.
CALL    COMPARE_VAR          ;repeat until J = 0
VARS3:  INC     CX            ;I = I + 1
CMP     DX,CX                ;Is I < 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     DX
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 swaped in memory if needed
;

```

```

PROC    COMPARE_VAR
PUSH    AX                    ;save registers
PUSH    BX
PUSH    CX
PUSH    DX
MOV     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

```

```

MOV     DI,AX                ;offset of J + Gap
MOV     SI,BX                ;offset of J
MOV     AX,[SI]              ;read word each str.
MOV     BX,[DI]              ;write word each str.
MOV     [SI],BX
MOV     [DI],AX
MOV     AX,DX                ;restore gap to AX
MOV     BX,BP                ;restore J to BX
SUB     BX,AX                ;J = J - gap
JZ      COMV3                ;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 [PerCnt] 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)

$6(\text{counter}-1) + 5 = \text{position in PerCnt}$

Output = Median and 25% and 75% stored in [PerCnt]

PROC STORE VAR

```

PUSH    AX
PUSH    BX
PUSH    CX
PUSH    DX
PUSH    ES
MOV     DX,AX                ;save number found
;-----compute offset in PerCnt 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 PerCnt   ;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 [percent]
INC     BX                  ;word pointer.
;-----get the highest

```

```

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                   ;advance [percent]
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         ;50% ptr
        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                ;restore 50% ptr
SHR     AX,1                 ;50%/2 = ptr to 25%
TEST    AL,01h               ;is the number even?
JZ      STV2                  ;if Yes goto next test
INC     AX                   ;if NO make it even
STV2:   MOV     DI,AX         ;25% ptr in DI
        ADD     DX,AX         ;75% ptr in DX
        MOV     AX,[ES:DI]   ;get 25% value
        MOV     [BX],AX      ;store 25% value
        INC     BX           ;advance [percent]
        INC     BX           ;word pointer.
        MOV     DI,DX         ;75% ptr to DI
        MOV     AX,[ES:DI]   ;get 25% value
        MOV     [BX],AX      ;store 25% value
        CLC                    ;clear carry flag
        POP     ES            ;restore registers
        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                ;clear menu area
CALL    MENU_BOX
MOV     CL,[Color]           ;save original attri
```

```

MOV     AL,[Warning]                ;warning color
MOV     [Color],AL                  ;set color
MOV     AX,010Bh                    ;row 3/Col 12
CALL    GOTOYX                      ;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    GOTOYX                     ;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

```

```

    db      .,0
    MOV     [Color],DL          ;restore original attri
    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,'.'           ;is digit > 9
    JNC     CDK3                ;if Yes then error
;-----is beginofarray value less than "10" ?
CKD2:  MOV     DI,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
CKD3:  CALL    DATA_ERR        ;inform user of error

```

```

CDK4: POP     ES
      POP     DX                      ;restore registers
      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 maybe 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    GOTOYX                 ;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
      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

```

```

CALL BIN_OUT
CALL CSTR_OUT           ;display warning
db " Press Any Key. ",0
MOV [Color],DL          ;restore original color
CALL HIDE_CUR
CALL ERR_SOUND
CALL GET_CHAR
CMP AL,1Bh              ;was it the <ESC> key?
JNZ DAE2                ;if NO continue
CALL ESC_YN             ;if YES inform user
JC DAE3                 ;carry flag = abort
DAE2: CALL CLEAR_MESSAGE
CLC                     ;clear carry flag
DAE3: POP DX            ;restore registers
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           ;reset file pointer
JC FDW4                 ;exit on DOS error
;-----assign buffer offset
MOV BX, Offset FilBuf
DEC CX                  ;column to 0 - ? form
ADD BX,CX               ;bx = buffer pointer
XOR CX,CX               ;zero line counter
MOV BP,DX               ;store original value
DEC DH                  ;adjust for loop
FDW1: INC DH             ;dx = search word + 0
CALL CLEAR_FILBUF       ;hex 0's to file buffer
CALL READ_LINE          ;1 line from data file
JNC FDW2                ;not EndOfFile
MOV AL,0FFh             ;mark EndOfFile true
MOV [EOF],AL            ;<> 0 = True
;-----is this the correct line?
FDW2: INC CX             ;inc line counter
CMP [BX],DX             ;is this a match
JZ FDW3
DEC DH                  ;dx = search word - 1

```

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	
POP DX	;restore registers
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	
MOV CL,[Color]	;store original Color
MOV AL,[Warning]	;warning 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 ? "	
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
    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,160Ah                ;row 22,column 12
    CALL    GOTOYX
    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

```

```

    INC    BX
;---place and ID of '000' in data string
    MOV    AX,'00'
    MOV    [BX],AX
    INC    BX
    INC    BX
    MOV    [BX],AL
;---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
    POP    DX                    ;restore registers
    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

```

JC PRA5	;exit on DOS error
CALL PROGRESS MESSAGE	;inform user of progress
CALL PRINT_INSTRU	;display bottom box
CALL INITIALIZE_HP	;hp to portrait mode
JC PRA4	;exit if printer error
;---get DOS timer ticks	
PRA1: MOV AH,0	;function number
INT 1Ah	;get DOS clock ticks
MOV BX,DX	;save ticks in CX
MOV AX,455	;18.2 ticks per second
ADD BX,AX	;add 25 seconds
JC PRA1	;loop if over flow
;---locate next ID number	
CALL FIND_DATA_LINE	;locate Next ID number
JC PRA3	;Normal Exit EndOfFile
CALL PROGRESS MESSAGE	;inform user of progress
CALL PRINT_TITLE	;print report header
JC PRA4	;exit if printer error
;---print each dimension	
CALL PRINT_DIM1	;print dimension 1
JC PRA4	;exit if printer error
CALL PRINT_DIM2	;print dimension 2
JC PRA4	;exit if printer error
CALL PRINT_DIM3	;print dimension 3
JC PRA4	;exit if printer error
CALL PRINT_DIM4	;print dimension 4
JC PRA4	;exit if printer error
CALL EJECT	;eject chart from HP
JC PRA4	;exit if printer error
;---is 25 seconds up yet?	
PRA2: MOV AH,0	;function number
INT 1Ah	;get DOS clock ticks
CMP DX,BX	;has time run out ?
JC PRA2	;if not loop until done
;---loop until all graphs are printed	
JMP SHORT PRA1	;go print next graph
;---normal exit point	
PRA3: CALL RESTORE_HP	;reset to normal defaults
CLC	;normal exit
JMP SHORT PRA5	
;---abort exit point	
PRA4: CALL RESTORE_HP	;abort or error exit
STC	;restore registers
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                      ;save registers
    PUSH  BX
    PUSH  CX
    PUSH  DX
:-----get ID number
    CALL  GET_ID
    JC    PRI5                    ;exit on <Esc> key
:-----locate ID number in the data file
    CALL  LOCATE                  ;locate ID Number
    JC    PRI5                    ;exit if not found
:-----printing of a report begins here
    CALL  PROGRESS_MESSAGE        ;inform user of progress
    CALL  PRINT_INSTRU           ;display bottom box
    CALL  INITIALIZE_HP          ;hp to portrait mode
    JC    PRI4                    ;exit if printer error
    CALL  PRINT_TITLE            ;print report header
    JC    PRI4                    ;exit if printer error
:-----print each dimension
    CALL  PRINT_DIM1             ;print dimension 1
    JC    PRI4                    ;exit if printer error
    CALL  PRINT_DIM2             ;print dimension 2
    JC    PRI4                    ;exit if printer error
    CALL  PRINT_DIM3             ;print dimension 3
    JC    PRI4                    ;exit if printer error
    CALL  PRINT_DIM4             ;print dimension 4
    JC    PRI4                    ;exit if printer error
    CALL  EJECT                  ;eject chart from HP
    JC    PRI4                    ;exit if printer error
PRI3: CALL  RESTORE_HP            ;reset to normal defaults
    CLC                          ;normal exit
    JMP  SHORT PRI5
PRI4: CALL  RESTORE_HP
    STC                          ;abort or error exit
PRI5: POP   DX                    ;restore registers
    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                      ;save registers
    PUSH  BX
    PUSH  CX
    PUSH  DX
:-----print dimension name

```

MOV	AX,0B08h	;row/col hex
CALL	HPGOTOYX	;set position
JNC	P10	;if on error continue
JMP	P13	;exit on printer error
;-----is this a POS or NEG dimensions?		
P10:	CMP BYTE PTR [Report],0	;get type of report
JZ	P11	;jump if positive
MOV	AX,Offset Neg1	;ptr to derallment str
JMP	SHORT P12	;jump to print the str
P11:	MOV AX,Offset Pos1	;ptr to positive str
P12:	CALL PRINT_STRING	;print the string
JC	P13	;exit on printer error
;-----draw chart outline		
MOV	AX,0B22h	;row/col hex
CALL	HPGOTOYX	;set position
JC	P13	;exit on error
MOV	AX,Offset Box	;ptr to draw box string
CALL	PRINT_STRING	;draw the dim box
JC	P13	;exit on printer error
;-----chart percentile for selfs, peers, superiors and subordinates		
MOV	AX,0A30h	;row/col of '1' hex
MOV	BX,Offset PerCnt + 5	;ptr to self data
CALL	CHART_RANGE_L	;draw shaded area
JC	P13	;exit on error
CALL	CHART_PERCENTILES	;draw the data
JC	P13	;exit on error
INC	AH	;row to peers
MOV	BX,Offset PerCnt + 45	;ptr to peers data
CALL	CHART_RANGE_D	;draw shaded area
JC	P13	;exit on error
CALL	CHART_PERCENTILES	;draw the data
JC	P13	;exit on error
INC	AH	;row to superiors
MOV	BX,Offset PerCnt + 85	;ptr to superiors data
CALL	CHART_RANGE_L	;draw shaded area
JC	P13	;exit on error
CALL	CHART_PERCENTILES	;draw the data
JC	P13	;exit on error
INC	AH	;row to subordinates
MOV	BX,Offset PerCnt + 125	;ptr to subordinates
CALL	CHART_RANGE_D	;draw shaded area
JC	P13	;exit on error
CALL	CHART_PERCENTILES	;draw the data
JC	P13	;exit on error
;-----chart points for self, peers, superiors and subordinates		
MOV	AX,0A30h	;restore row/col of '1'
MOV	BX,Offset FilBuf + 3	;ptr to self point
CALL	CHART_POINT	;ax=starting row/col
JC	P13	;exit on error
INC	AH	;row to peers
MOV	BX,Offset FilBuf + 15	;ptr to self point
CALL	CHART_POINT	;ax=starting row/col

```

JC      P13                      ;exit on error
INC     AH                      ;row to superiors
MOV     BX,Offset FilBuf + 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 FilBuf + 39   ;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,1308h        ;row/col hex
        CALL    HPGOTOYX        ;set position
        JNC     P20             ;if on error continue
        JMP     P23             ;exit on printer 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 deralliment 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 superiors
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 subordinates
MOV      BX,Offset PerCnt + 135   ;ptr to subordinates
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 FilBuf + 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 FilBuf + 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 FilBuf + 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 FilBuf + 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

```

```

PROC PRINT_DIM3
    PUSH    AX                      ;save registers
    PUSH    BX
    PUSH    CX
    PUSH    DX
;----print dimension name
    MOV     AX,1B08h                ;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 derailment 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'

```

```

MOV    BX,Offset FilBuf + 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 FilBuf + 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 FilBuf + 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 FilBuf + 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    HPGOTOYX                 ;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
P42:    CALL    PRINT_STRING              ;print the string
        JC      P43                       ;exit on printer error
;-----draw chart outline
        MOV     AX,2022h                 ;row/col hex
        CALL    HPGOTOYX                 ;set position
        JC      P43                       ;exit on error
        MOV     AX,Offset Box             ;ptr to draw box string

```

```

CALL PRINT_STRING                ;draw the dim box
JC P43                          ;exit on printer error
;-----chart percentile for self, 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_L              ;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 subordinates
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 FilBuf + 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 FilBuf + 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 FilBuf + 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 FilBuf + 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

```

```

RET
ENDP PRINT_DIM4

```

```

.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,0FFh           ;read char from key-

```

```

    INT     21h               ;board 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 ext.

```

```

;---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               ;OK! <> 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

```

```

;-----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,020Bh ;row 3/Col 12
      CALL GOTOYX ;set cursor
      CALL CSTR_OUT ;display warning
      db " Printer is off line. Do you want to try again ? "
      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

```

```

:
:   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  GOTOYX               ;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

```

```

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
MOV CL,[Color] ;store original Color
MOV AL,[Warning] ;warning color
MOV [Color],AL ;set color
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,1Bh ;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
      INC     DI
      INC     DI
      MOV     [DI],AX
;-----locate ID number in the data file
FID1:  CALL    CLEAR_FILBUF
      CALL    READ_LINE
      JNC     FID2
      MOV     AL,0FFh
      MOV     [EOF],AL
;-----is this a '000' data line?
FID2:  MOV     CX,3
      MOV     DI,Offset ID
      MOV     SI,Offset FilBuf
      CLD
      REPZ    CMPSB
      JZ      FID5
;-----is this an ID number line?
      MOV     SI,Offset FilBuf
      MOV     CX,3
FID4:  MOV     AL,[SI]
      CMP     AL,'0'
      JC      FID5
      CMP     AL,':'
      JNC     FID5
      INC     SI
      LOOP    FID4
;-----copy ID to [ID] string
      ;assign ES = DS
      ;ptr to string to fill
      ;ASCII zeros
      ;place 1st two bytes
      ;zero = end of string
      ;advance string ptr
      ;ASCII 0 and hex 0
      ;hex 0's to file buffer
      ;1 line from data file
      ;not EndOfFile
      ;mark EndOfFile true
      ;<> 0 = True
      ;loop counter
      ;ptr to ID number
      ;ptr to data file line
      ;auto inc DI and SI
      ;are the bytes = ?
      ;skip if ID = '000'
      ;ptr to data file line
      ;loop counter
      ;get first byte
      ;is it < ASCII 0
      ;if Yes read next line
      ;is it a digit?
      ;if No read next line
      ;point to next byte
      ;check next byte
      ;0=OK! Found Data Line

```

MOV	CX,3	;number of bytes to move
MOV	DI,Offset ID	;ptr to ID number
MOV	SI,Offset FIDBuf	;ptr to data file line
CLD		;auto inc DI and SI
REP	MOVSB	;copy three bytes to ID
JMP	SHORT FID6	;Exit found
;-----Is this the last line ?		
FID5:	XOR AL,AL	;zero AX register
	CMP AL,[EOF]	;Is EndOfFile TRUE?
	JZ FID1	;False = get next line
	STC	;carry flag = None
	JMP SHORT FID7	;retrun EndOfFile
FID6:	CLC	;clear carry flag
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	
	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 derailment str
	JMP SHORT PT2	;jump to print the str
PT1:	MOV AX,Offset PosT	;ptr to positive str
PT2:	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	
	CALL PRINT_STRING	

```

JC      PT5                      ;exit on printer error
MOV     AX,0708h                 ;row/col hex
CALL    HPGOTOYX
JC      PT5                      ;exit on printer error
MOV     AX,Offset DiStr         ;string holds Bold OFF
CALL    PRINT_STRING
JC      PT5                      ;exit on printer error
MOV     AX,2822h                 ;row/col hex
CALL    HPGOTOYX
JC      PT5                      ;exit on printer error
MOV     AX,Offset Inform        ;bottom information box
CALL    PRINT_STRING
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 a 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 on line?
PR2:    CALL    ON_LINE
        JNC     PR3
        CALL    LPT_ERR
PR4:    STC                      ;set error flag
PR5:    POP     DX                ;restore registers
        POP     CX
        POP     BX
        POP     AX
        RET
ENDP    IS_PRINT
;

```

```

:   Input = none
:   Output = none
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

```

```

:   Input = none
:   Output = none
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

```

```

:   Input = none
:   Output = none
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 SLDI data file.'
    db      ' Press Any Key. ',0
    MOV     [Color],CL           ;restore original color
    CALL    HIDE_CUR

```

```

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 ;display period
db ',0 ;endOfSting 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
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 ..... Searching file for ID Number: ',0
MOV AX, Offset ID
CALL DSTR_OUT
CALL CSTR_OUT ;display period

```

```

    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 port
    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                ;restore registers
    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.
;-----send character to assigned LPT port
      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

```

```

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

:
:   Input = none
:   Output = none
PROC   RESTORE_MESS
    CALL   CLEAR_MESSAGE
    MOV    AL,[Menu]              ;menu color
    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
                                ;if NO column = 0
    XOR    AL,AL                ;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
                                ;if NO row = 0
    XOR    AL,AL                ;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    GOTOYX
        CALL    CSTR_OUT
        db      'low',0
        MOV     CX,8                   ;loop counter
AH0:    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    AH0                    ;loop 8 times
        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,1608h         ;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      SEA6              ;exit on error
:-----locate ID number in the data file
CALL    LOCATE_MESSAGE    ;inform user of search
SEA1:   CALL    CLEAR_FILBUF ;hex 0's to file buffer
CALL    READ_LINE        ;1 line from data file
JNC     SEA2              ;not EndOfFile
MOV     AL,0FFh           ;mark EndOfFile true
MOV     [EOF],AL         ;< > 0 = True
SEA2:   MOV     CX,3        ;loop counter
MOV     DI,Offset ID      ;ptr to ID number
MOV     SI,Offset FilBuf  ;ptr to data file line
CLD                       ;auto inc DI and SI
REP     CMPSB             ;are the bytes = ?
JZ      SEA7              ;if YES exit found
XOR     AL,AL             ;zero AX register
CMP     AL,[EOF]          ;is EndOfFile TRUE?
JZ      SEA1              ;False = get next line
SEA5:   CALL    NOT_FOUND  ;inform user not found

```

```

SEA6: STC                                ;carry flag = not found
SEA7: POP DX                            ;restore registers
      POP CX
      POP BX
      POP AX
      RET

:
:   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 GOTOYX                        ;set cursor
      CALL CSTR_OUT                      ;display warning
      db 'The ID# is not in current data file.'
      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 1
:           BX = Offset of data value in FilBuf
:   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

```

PROC CHART_POINT

```

PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX

;---set cursor at value "1" on the chart
CALL  HPGOTOYX
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

CP2:  AND   AX,0CFCFh
MOV   DX,AX
XOR   CX,CX
ADD   CL,DL
JZ    CP7
DEC   CX
JZ    CP4
CMP   CX,5
JNC   CP7
MOV   AX,Offset NextNo
CP3:  CALL  PRINT_STRING
JC    CP8
LOOP  CP3
CP4:  XOR   CX,CX
ADD   CL,DH
JZ    CP6
CMP   CX,10
JNC   CP7
MOV   AX,Offset NextUn
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

```

```

;save registers

;set position
;exit on printer error

;point value
;is it <spaces>?
;if yes exit no error
;is it EndOfLine?
;if yes exit no error

;round the number ?
;if < 5 = no round

;is unit digit = 9
;if Yes goto over flow
;else advance units
;goto next test
;adjust if over flow
;example 29 to 30
;convert to hex
;save number in DX
;zero to loop counter
;get tens digit
;exit data error
;scale starts at 1 not 0
;if zero skip tens
;is data in bounds?
;exit if out of bounds
;ptr to next tens str
;advance to next tens
;exit on printer error
;loop until tens = 0
;zero to loop counter
;get units digit
;if zero skip units
;is data in bounds?
;exit if out of bounds
;ptr to next units str
;advance to next unit
;exit on printer error
;loop until units = 0
;ptr to point string
;plot point in chart
;exit on printer error

;restore registers

```

```

    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,0CFCh          ;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

```

```

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_OUT                ;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 1
;        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 1
;        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    CHM2                   ;exit on printer error
;-----chart the median
ADD   BX,4                   ;ptr median in buffer
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 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 HPGOTOYX	;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,' '	;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 HPGOTOYX	;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	
:	
:	

```

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

;-----get 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

```

DEC AL	;tens = tens - 1
CPT9: SUB AH,DH	;subtract unit digit
SUB AL,DL	;subtract ten digit
JC CPT14	;exit if value < 0
MOV DX,AX	;save answer in DX
XOR CX,CX	;zero to loop counter
ADD CL,DL	;get tens digit
JZ CPT11	;if zero skip tens
CMP CX,5	;is data in bounds?
JNC CPT14	;exit if out of bounds
MOV AX,Offset TenLi	;ptr to next tens str
CPT10: CALL PRINT_STRING	;advance to next tens
JC CPT15	;exit on printer error
LOOP CPT10	;loop until tens = 0
CPT11: XOR CX,CX	;zero CX register
ADD CL,DH	;get units digit
JZ CPT14	;if zero skip units
CMP CX,10	;is data in bounds?
JNC CPT14	;exit if out of bounds
CMP CL,4	;is units 1,2 or 3 ?
JC CPT12	;if yes draw units
INC CL	;add 1 to units
CMP CL,7	;orig unit 4 5 or 6?
JC CPT12	;if yes draw units
INC CL	;add 1 to units
CPT12: MOV AX,Offset UntLi	;ptr to next units str
CPT13: CALL PRINT_STRING	;advance to next unit
JC CPT15	;exit on printer error
LOOP CPT13	;loop until units = 0
CPT14: CLC	
CPT15: POP DX	;restore registers
POP CX	
POP BX	
POP AX	
RET	
ENDP CHART_RANGE_L	
...	
PROC CHART_RANGE_D	
PUSH AX	;save registers
PUSH BX	
PUSH CX	
PUSH DX	
;---set cursor at value "1" on the chart	
CALL HPGOTOYX	;set position
JNC DPT1	;if no error continue
JMP DPT15	;error exit
;---get lowest values from file PerCnt buffer	
DPT1: MOV AX,[BX]	;get lowest score
CMP AX,' '	;is it <spaces>?
JNZ DPT2	;if no goto next test
JMP DPT14	;if yes exit no error

DPT2: CMP AX,0	;is it EndOfLine?
JNZ DPT3	;if no goto next test
JMP DPT14	;if yes exit no error
DPT3: CMP AX,'01'	;is the score 1.0 ?
JNZ DPT4	;if NO jump to DPT4
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 DPT5	;exit on printer error
MOV AX,DX	;restore value to AX
DPT4: CALL POSITION_YX	;set cursor low range
DPT5: JC DPT15	;exit on printer error
AND AX,0CFCFh	;convert to hex
MOV DX,AX	;save in DX
;---get 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 DPT14	;if yes exit no error
CMP AX,0	;is it EndOfLine?
JZ DPT14	;if yes exit no error
CMP AX,'05'	;is it a 5.0?
JNZ DPT8	;if no continue
MOV CX,AX	;save value in CX
CMP DH,0	;is starting unit = 0?
JNZ DPT6	;if not backup 1/2 sp
MOV AX,Offset FullBk	;ptr to full space str
JMP DPT7	;goto move cursor
DPT6: MOV AX,Offset BackSp	;ptr to 1/2 space str
DPT7: CALL PRINT_STRING	;move cursor 1/2 space
JC DPT15	;exit on printer error
MOV AX,CX	;restore value to AX
DPT8: 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 DPT9	;if no ready to sub
ADD AH,10	;add borrow to units
DEC AL	;tens = tens - 1
DPT9: SUB AH,DH	;subtarct unit digit
SUB AL,DL	;subtract ten digit
JC DPT14	;exit if value < 0
MOV DX,AX	;save answer in DX
XOR CX,CX	;zero to loop counter
ADD CL,DL	;get tens digit
JZ DPT11	;if zero skip tens
CMP CX,5	;is data in bounds?
JNC DPT14	;exit if out of bounds
MOV AX,Offset TenDk	;ptr to next tens str
DPT10: CALL PRINT_STRING	;advance to next tens
JC DPT15	;exit on printer error
LOOP DPT10	;loop until tens = 0

```

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 UnitDk     ;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      'File Rank Print LPT'
      db      'Quit',0
      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              ;Hotkey ASCIIZ string.

```

```

;
;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

```

```

m01 db 'Select a SLDI data file for printing.',0
m02 db 'Compute the 25th and 75th percentiles for the data'
db 'file.',0
m03 db 'Print SLDI FeedBack reports on a HP Laser Printer.',0
m04 db 'Select the parallel port assigned to the HP Laser '
db 'Printer.',0
m05 db 'Exit the program and return to DOS.',0

```

;end of menu0 structure

;

;Menu1data structure

```

menu1 dw Offset menu1HK ;ptr to menu HotKeys
db 'First LPT Second LPT Third '
db 'LPT ',0
db 9,11 ;lightbar:position in string and number of bytes
dw Offset m11 ;pointer to lightbar message
db 28,12
dw Offset m12
db 48,11
dw Offset m13

```

;the menu menu1HK 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.

```

menu1HK db 'FST',0 ;Hotkey ASCIIZ string.

```

;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: "m11" stands for menu1 message1

```

m11 db 'Send the SLDI data to the computer's LPT 1 '

```

```

db 'output port.',0

```

```

m12 db 'Send the SLDI data to the computer's LPT 2 '

```

```

db 'output port.',0

```

```

m13 db 'Send the SLDI data to the computer's LPT 3 '

```

```

db 'output port.',0

```

;end of menu1 structure

;

;Menu2data structure

```

menu2 dw Offset menu2HK ;ptr to menu HotKeys
db 'Success Factors Failure Factors'
db 'Main Menu ',0
db 04,17 ;lightbar:position in string and number of bytes
dw Offset m21 ;pointer to lightbar message
db 29,17
dw Offset m22
db 54,11
dw Offset m23

```

```

;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 'SFM',0 ;Hotkey ASCIIZ string.

```

```

;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: "m21" stands for menu2 message1

```

```

m21 db ' Display the names of data files for positive '
    db 'factors.',0
m22 db ' Display the names of data files for negative '
    db 'factors.',0
m23 db ' Return to the Main Menu without selecting a file.',0
;end of menu2 structure

```

```

;Menu3data structure

```

```

menu3 dw Offset menu3HK ;ptr to menu HotKeys
    db ' Single Report All Reports '
    db ',0
    db 11,15 ;lightbar:position in string and number of bytes
    dw Offset m31 ;pointer to lightbar message
    db 44,13
    dw Offset m32

```

```

;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 ASCIIZ string.

```

```

;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: "m11" stands for menu1 message1

```

```

m31 db ' Use the three digit ID number to select a report'
    db ' to print.',0
m32 db ' Print reports for all the ID numbers in the'
    db ' data file.',0
;end of menu3 structure

```

```

.CODE

```

```

Present the Main Menu and Title screen
Input = None

```

```

:      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                      ;row 0 and column 0
      CALL    MENU_BOX                  ;draw top menu box
      CALL    DRAW_TITLE
      CALL    DISPLAY_STATUS            ;display name & LPT port
      CALL    MENU_INSTRU               ;draw bottom box
      MOV     CH,01                     ;starting menu selection
MA1:   MOV     AX, Offset Menu0
      MOV     CL,5                       ;max choice for menu1
      CALL    GET_CHOICE                ;get a menu selections
:-----is it Esc key?
      CMP     AH,0h                     ;was <ESC> key pressed?
      JNZ     MA2                       ;if not goto next test
      CALL    EXIT_YN                   ;exit to Dos Y/N?
      JNC     MA1                       ;if no get next choice
      JMP     MA8                       ;Exit on <Esc> key
MA2:   MOV     CH,AH                     ;save current choice
:-----is it Select a data file?
      DEC     AH                         ;is this the choice?
      JNZ     MA3                       ;if not goto next test
      CALL    GET_PATH                  ;get file path
      JC      MA2B                      ;main menu if Esc key
      CALL    REPORT_MENU               ;Open an SLDI data file
MA2B:  XOR     AX,AX                     ;row/column
      CALL    MENU_BOX                  ;clear menu box
      JMP     SHORT MA1                 ;get another choice
:-----is it Rank the variables?
MA3:   DEC     AH                         ;is this the choice?
      JNZ     MA4                       ;if not goto next test
      CALL    RANK_DATA                 ;compute presentiles
      CALL    MENU_INSTRU               ;restore bottom box
      CALL    DISPLAY_STATUS
      JMP     SHORT MA1                 ;get another choice
:-----is it Print the data file?
MA4:   DEC     AH                         ;is this the choice?
      JNZ     MA5                       ;if not goto next test
      CALL    PRINT_MENU
      JMP     SHORT MA1                 ;get another choice
:-----is it choose a Laser port?
MA5:   DEC     AH
      JNZ     MA6
      CALL    PORT_MENU                 ;Select LPT port
      JMP     SHORT MA7                 ;return to main menu
:-----is it the Exit command?
MA6:   DEC     AH

```

```

        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        ;was <ESC> key pressed?
        JZ   P14          ;if yes exit menu
        MOV  CH,AH        ;save current choice
;-----is it 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           ;is this the choice?
        JNZ  P11          ;NO = get next choice
        CALL PRINT_ALL_REPORTS ;for a all ID numbers
        CALL MENU_INSTRU   ;restore bottom box
P14:    CLC              ;clear cf = normal exit
        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]                ;current port assignment
    MOV     CH,CL                  ;place in CH register
    INC     CH                      ;starting menu selection
MI1:   MOV     AX, Offset Menu1
    MOV     CL,3                   ;max choice for menu1
    CALL    GET_CHOICE             ;get a menu selections
:-----Is it Esc key ?
    CMP     AH,0h                 ;was <ESC> key pressed?
    JZ      MI4                   ;if yes exit menu
    XOR     AL,AL                 ;zero LPT choice
    MOV     CH,AH                 ;save current choice
:-----Is it Select LPT 1 ?
    DEC     AH                   ;is this the choice?
    JNZ     MI2                   ;if not goto next test
    XOR     AH,AH                 ;zero AH register
    MOV     [LPT],AX             ;assign 0 to [LPT]
    JMP     SHORT MI4            ;exit procedure
:-----Is it Select LPT 2 ?
MI2:   INC     AL                 ;assign 1 to AL
    DEC     AH                   ;is this the choice?
    JNZ     MI3                   ;if not goto next test
    XOR     AH,AH                 ;zero AH register
    MOV     [LPT],AX             ;assign 1 to [LPT]
    JMP     SHORT MI4            ;exit procedure
:-----Is it Select LPT 3 ?
MI3:   INC     AL                 ;assign 2 to AL
    DEC     AH                   ;is this the choice?
    JNZ     MI1                   ;NO = get next choice
    XOR     AH,AH                 ;zero AH register
    MOV     [LPT],AX             ;assign 2 to [LPT]
MI4:   CLC                      ;clear cf = normal exit
    POP     DX
    POP     CX
    POP     BX
    POP     AX
    RET
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
    PUSH    BX

```

```

        PUSH    CX
        PUSH    DX
        XOR     AX,AX                ;row 0 and column 0
        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 selections
;-----Is it Esc key ?
        CMP     AH,0h                ;was <ESC> key pressed?
        JZ      RE4                ;if yes exit menu
        MOV     CH,AH                ;save current choice
;-----Is it a Positive 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 a Derailment Report ?
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
;-----Is it Main Menu ?
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:    POP     DX
        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
    INC     CH                         ;Choice = Choice + 1
GE5:  CMP     AL,13h                   ;is it a left arrow?
    JNZ     GE6                         ;jump if not
    DEC     CH                         ;Choice = Choice -1
    JNZ     GE6                         ;if CH = 0 then
    MOV     CH,CL                      ;set CH = maxmenu.
GE6:  CMP     CL,CH                    ;is CH > maxmenu?
    JNC     GE7                         ;if yes then
    MOV     CH,1                       ;set CH = 1
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     NML0                       ;if not = 1 continue

```

```

        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
        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 ASCIIZ 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 ASCIIZ 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 Menu1data 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 ASCIIZ HotKey str.

```

        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     CL,BL           ;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 Menu 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

```

```

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 position
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_WINDOW        ;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

```

```

PUSH    DX
MOV     AX,1500h                ;row 21,column 0
CALL    MENU_BOX                ;draw menu box
MOV     AL,[Menu]
MOV     [Color],AL
MOV     AX,160Ch                ;row 22,column 12
CALL    GOTOYX
CALL    CSTR_OUT
db      'Use the ',17,205,' 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

```

CALL	CSTR_OUT	
db	'Number of ID's = ',0	
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	;send string to screen
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	;send string to screen
db	' is Ready. ',0	
FLE9:	MOV [Color],CL	;restore color value
POP	DX	
POP	CX	
POP	BX	
POP	AX	
RET		
ENDP	DISPLAY_STATUS	
;		

```

.DATA
TKey db 'FREEBANK'

Shade db 'FreeBank'

```

```

.CODE
;-----Display the programs title on the screen
Input = None
[Color] = current attribute from data section
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,[Warning]
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,0A04h
    MOV    AX,Offset Tkey
;line counter
;row/col
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
;

```

-----Send an ASCIIZ 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 register 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
    POP     CX                 ;restore registers.
    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

```

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                 ;jmp 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
    MOV     AH,0Fh
    INT     10h
    CMP     AL,7
    JNZ     VID3
    CALL    MONO_VIDEO
    JMP     SHORT VID5
VID3: CMP     AL,0Fh
    JNZ     VID4
    CALL    MONO_VIDEO
    JMP     SHORT VID5
VID4: CALL    COLOR_VIDEO
VID5: RET

;
PROC COLOR_VIDEO
    MOV     BX,Offset Menu
    MOV     AL,71h                ;Menu=Blue/Lt White
    MOV     [BX],AL
    INC     BX
    MOV     AL,1Fh                ;Normal=White/Blue
    MOV     [BX],AL
    INC     BX
    MOV     AL,1Eh                ;HiLite=Yellow/Blue
    MOV     [BX],AL
    INC     BX
    MOV     AL,7Fh                ;MenuMes=White/Lt White
    MOV     [BX],AL
    INC     BX
    MOV     AL,4Fh                ;Warning=White/Red
    MOV     [BX],AL
    INC     BX
    MOV     AL,070h               ;Border=Black/White
    MOV     [BX],AL
    RET
ENDP COLOR_VIDEO
PROC MONO_VIDEO
    MOV     BX,Offset Menu

```

```

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 users 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                ;if yes save font
MOV     AX,1114h
CMP     CX,16              ;is it 8x16 font ?
JZ     SV1                ;if yes save font
MOV     AX,1111h          ;assume 8x14 font
SV1:    MOV     [vidfont],AX ;save current font
;-----get current color attributes
MOV     AH,08h            ;read char and attr
MOV     BH,[vidpage]      ;get current video page
INT     10h
MOV     [vidattr],AH      ;save color attribute
MOV     CL,4              ;counter
SHR     AH,CL             ;high nibble to low
MOV     [vidbord],AH      ;save background color
;-----is this a VGA system ?
MOV     AX,1A00h          ;read video config.
INT     10h
CMP     AL,1Ah            ;is it an VGA ?
JNZ     SV2               ;if no then exit
MOV     AX,1008h          ;if yes get border color
INT     10h
MOV     [vidbord],BH      ;save border color
;-----set font type
MOV     AX,1114h          ;8x16 character set
XOR     BL,BL
INT     10h
;-----set text video mode
SV2:    MOV     AX,0003     ;default video text mode
MOV     BL,[vidmode]      ;get current mode
CMP     BL,7              ;is it mono text ?
JZ     SV5                ;if yes no change
INT     10h               ;set color text mode
SV5:    MOV     AL,[Normal]
MOV     [Color],AL
CALL    CLEAR_SCREEN
CLC                                ;clear carry flag
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],[vidcurs],[vidfont]
                     [vidattr] and [vidbord]

```

```

PROC    RESTORE_VIDEO
PUSH    ES
;-----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 ?
    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,DX                ;ptr to data area
    MOV     DX,[ES:4Ah]          ;get number of columns
    DEC     DX                  ;convert to 0 start
    MOV     DH,[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 ASCIIZ 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	;share/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	
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
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

```

CALL    DSTR_OUT                ;display type of error
MOV     [Color],CL              ;restore original color
CALL    HIDE_CUR
CALL    ERR_SOUND
CALL    GET_CHAR                ;to stop program
STC                                ;set carry flag
OP4:    RET
ENDP    OPEN
;
.DATA
OpenErr dw    Offset OpE1,Offset OpE2,Offset OpE3
        dw    Offset OpE4,Offset OpE5,Offset OpE6,Offset OpE7
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     DI,CL                 ;DI = 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 DI register
    INC     SI                   ;point to next char
    DEC     DI                   ;character counter
    JNZ     SHORT DSTR3          ;get next character.
DSTR4:    POP     DX
    POP     CX                   ;restore registers.

```

```

    POP    BX
    POP    AX
    RET
ENDP  SUB_DSTR_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!!!

```

```

PROC GOTOYX
    PUSH    AX                ;save registers
    PUSH    BX
    PUSH    CX
    PUSH    DX
    MOV     DX,AX
    CMP     DH,26             ;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
    CMP     DL,79
    JZ      INC1
    INC     DL
    JMP     SHORT INC2
INC1:  INC     DH
INC2:  MOV     AX,0200h
    INT     10h

```

```

    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                   ;page 0 assumed
    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,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

```

```

        INC     SI                      ;point to next char
        JMP     SHORT DSTR1             ;get next character.
DSTR2: POP     DX
        POP     CX                      ;restore registers.
        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

```

PRCC 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)

```

: Input = none
: Output = binary ASCII keyboard code in AL
: Carry flag = extended code.
: extended codes are converted to control keys by EXT_CHAR
PROC GET_TEXT
GET1: MOV AX,0700h ;input function number
      INT 21h ;wait for character
      CLC ;clear carry flag
      CMP AL,0 ;is the Char extended?
      JNZ @@TEXT ;if not extended return
      MOV AH,07h ;unfilter char input
      INT 21h ;to get extended char.
      CALL EXT_CHAR ;convert extended codes
      JC GET1 ;cf = not on the list
      STC ;set carry flag
@@TEXT: MOV AH,0 ;zero AH register.
      RET ;end of subroutine
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 ;clear keyboard buffer
      INT 21h ;and wait for char.
      CLC ;clear carry flag
      CMP AL,0 ;is the Char extended?
      JNZ @@CHAR ;if not extended return
      MOV AH,07h ;unfilter char input
      INT 21h ;to get extended char.
      CALL EXT_CHAR ;convert extended codes
      JC GET0 ;cf = not on the list
      STC ;set carry flag
@@CHAR: MOV AH,0 ;zero AH register.
      RET ;end of subroutine

```

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:

Key	Extended Code	Converted to:	Ctrl-Char	Ctrl-Value
Home	47h	^A	1h	
UpArr	48h	^E	5h	
PgUp	49h	^R	12h	
LiArr	4Bh	^S	13h	
RtArr	4Dh	^D	4h	
End	4Fh	^F	6h	
DnArr	50h	^X	18h	
PgDn	51h	^C	3h	
Ins	52h	^V	16h	

```

; Del 53h ^G 7h
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,4Bh
    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 ;set carry flag
    JMP SHORT EXT11
EXT10: CLC ;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?
    JC      WIN1               ;exit if yes.
    CMP     BL,AL              ;is starting col>ending?
    JC      WIN1               ;exit if yes.
    CMP     BH,25              ;is row out of bounds?
    JNC     WIN1               ;exit if yes.
```

```

    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      200, 78 DUP (205),186,0
    MOV     AL,[Normal]
    MOV     [Color],AL
    POP     DX
    POP     CX
    POP     BX
    POP     AX

```

```

    RET
ENDP  DISPLAY_BOX
:
:  Display an error message on the screen in row 5 column 3, Normal colors
:  Input  = AX pointer to ASCII 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,0403h            ;point row 4, col 3
    CALL    GOTOYX               ;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
                                ;timer channel 2.
    OUT     61h,AL              ;initialize channel 2
    MOV     AL,0B6h             ;for mode 3
    OUT     43h,AL
SOU1: MOV   DX,[SI]             ;load frequency
    CMP     DX,0                ;is it the end of str?
    JZ      SOU3                ;if yes exit else

```

```

INC     SI                ;advance ptr to
INC     SI                ;point to the duration
MOV     AL,DL             ;low lsb 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 lword to BX
ADD     BX,[SI]           ;add duration to BX
INC     SI                ;advance ptr to
INC     SI                ;point to next frequency.
SOU12:  INT     1Ah       ;get BIOS timer count
        CMP     DX,BX     ;is count > duration?
        JC      SOU2      ;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 lsb 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
        POP     DX         ;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

```

```

PUSH  AX                ;save registers
PUSH  BX
PUSH  CX
PUSH  DX
MOV    AL,[Menu]        ;Menu color
MOV    [Color],AL       ;set color
MOV    AX,0206h         ;row 3; col 8
CALL   GOTOYX           ;position cursor
CALL   CSTR_OUT
db     73 DUP(20h),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 an 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                     ;clear carry flag
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,6             ;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

```

```

MOV     CL,[Color]           ;save color attribute
MOV     AL,[Warning]         ;get new attribute
MOV     [Color],AL           ;assign attri to color
CALL    CSTR_OUT              ;output this line to
db      'Error! creating ',0  ;the screen.
MOV     AX,DX                 ;File name pointer.
CALL    DSTR_OUT              ;output file name.
MOV     AX,[BX]               ;ptr msg string to AX
CALL    DSTR_OUT              ;display type of error
MOV     [Color],CL           ;restore original attri
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        ;contine 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
ENDP BIN_DIG_OUT              ;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

```

PROC EXIT_YN
    PUSH AX                ;save registers
    PUSH BX
    PUSH CX
    PUSH DX
    CALL CLEAR_MESSAGE
    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 'Exit to DOS ? [Y]/N ',0
    MOV AL,[Normal]        ;normal color
    MOV [Color],AL         ;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

```

```

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 comor.
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 procdeures
PROC MENU_BOX
PUSH    AX
PUSH    BX
PUSH    CX
PUSH    DX
XOR     AL,AL                ;set column to 0
CALL    GOTOYX               ;set cursor position
MOV     DL,[Color]           ;save original Color
MOV     AL,[Menu]            ;clange color attribute
MOV     [Color],AL           ;for scren 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 funct 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.

```

```

:      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 then 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      ERRO1               ;if NO display message

```

```

    CMP     AL,37                ;is code < 37 ?
    JC      ERRO1                ;# YES display mess
    CALL    CSTR_OUT
    db      ' DOS Error Number: ',0
    CALL    BIN_OUT
    MOV     AX,37                ;37 = unknown DOS error
ERRO1: SHL     AX,1              ;error number x 2
    MOV     BX,Offset ErrStr     ;pointer to table base
    ADD     BX,AX               ;BX ptr to error ptr.
    CALL    CSTR_OUT
    db      ' DOS Error: ',0
    MOV     AX,[BX]             ;load ptr to error str.
    CALL    DSTR_OUT            ;send string to screen
    CALL    CSTR_OUT
    db      ' Press Any Key. ',0
    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      'unkown unk',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      'unkown unk',0

```

```

Err21 db 'drive not ready',0
Err22 db 'unkown command',0
Err23 db 'data error (crc)',0
Err24 db 'bad request structure length',0
Err25 db 'seek error',0
Err26 db 'unkown media type',0
Err27 db 'sector not found',0
Err28 db 'printer out of paper',0
Err29 db 'write fault',0
Err30 db 'read fault',0
Err31 db 'general failure',0
Err32 db 'sharing violation',0
Err33 db 'lock violation',0
Err34 db 'disk change invalid',0
Err35 db 'FCB unavailable',0
Err36 db 'sharing buffer exceeded',0
Err37 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

```

```

PUSH    ~
PUSH    ~
PUSH    SI
PUSH    DI
PUSH    BP
PUSH    DS
PUSH    ES
MOV     DX,AX                ;save in DX
MOV     AX,@DATA             ;get data segment
MOV     DS,AX                ;assign data segment
MOV     CL,[Color]           ;get current attribute
MOV     AL,[Warning]         ;get warning color
MOV     [Color],AL           ;assign to color
MOV     AX,0220h             ;row 4 / col 0
CALL    GOTOYX               ;position cursor
CALL    CSTR_OUT             ;send string to screen
db      7,7,' Error: Press R to Retry or A to Abort. ',0
CALL    HIDE_CUR

CRT1:   MOV     AH,6           ;Dos function number
MOV     DL,0FFh              ;get char input from
INT     21h                  ;the keyboard.
JZ      CRT1                 ;if not char try again
AND     AL,5Fh               ;make capital letter
MOV     AH,AL                ;input to AH
MOV     AL,3                 ;3 = fail the DOS call
CMP     AH,'A'               ;is it an Abort ?
JZ      CRT2                 ;exit if an Abort
MOV     AL,1                 ;1 = retry the DOS call
CMP     AH,'R'               ;is it a Retry ?
JNZ     CRT1                 ;if not get new char

CRT2:   MOV     DL,AL         ;save return code
MOV     AL,[Menu]           ;get menu color
MOV     [Color],AL         ;assign to color
MOV     AX,0220h           ;row 4 / col 0
CALL    GOTOYX             ;position cursor
CALL    CSTR_OUT           ;send string to screen
db      ,0
MOV     [Color],CL         ;assign original attri
MOV     AX,DX              ;restore DOS return code
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 = hilite bar position # 1 to 14

BL = starting directory number # 1 to MaxFile

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 SEL5

;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 GET_CHAR

;get keyboard input

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    FIND_ZERO                 ;locate 000 in data file
        JC      SEL5
SEL4:  CALL    RELEASE_MEM_DIR             ;release mem block
        CLC                                ;clear carry flag
        JMP     SHORT SEL6
SEL5:  CALL    RELEASE_MEM_DIR             ;release mem block
        STC                                ;error set carry flag
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     AH,AL                      ;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 begining of file ?
        STC                                ;set carry for ret
        JZ      SHORT KYS10               ;if yes Exit no changes

```

```

        CMP     BH,1
        JZ      KYS3
        DEC     BH
        JMP     SHORT KYS9
KYS3:   DEC     BL
        JMP     SHORT KYS9
;-----Is it a Home Key ?
KYS4:   CMP     AL,1
        JNZ     KYS5
        MOV     BX,0101h
        JMP     SHORT KYS9
;-----Is it a End Key ?
KYS5:   CMP     AL,6
        JNZ     KYS7
        MOV     AH,[MaxFile]
        CMP     AH,15
        JNC     KYS6
        MOV     BL,1
        MOV     BH,AH
        JMP     SHORT KYS9
KYS6:   SUB     AH,13
        MOV     BL,AH
        MOV     BH,14
        JMP     SHORT KYS9
;-----Is it the PageUp Key ?
KYS7:   CMP     AL,18
        JNZ     KYS8
        MOV     AH,BL
        SUB     AH,14
        MOV     AL,1
        JLE     KYS4
        MOV     BL,AH
        JMP     SHORT KYS9
;-----Is it the PageDn Key ?
KYS8:   CMP     AL,3
        STC
        JNZ     KYS10
        MOV     AH,BL
        MOV     AL,[MaxFile]
        SUB     AL,14
        ADD     AH,14
        CMP     AL,AH
        MOV     AL,6
        JLE     KYS5
        MOV     BL,AH
KYS9:   CLC
KYS10:  RET
ENDP    CONTROL_KEYS
;
;-----Display Files in Memory Directory
;      Input  AL = starting directory number (1 to MaxFile)
;            AH = hilite bar number (1 to 14)

```

```

;is top of window ?
;if yes Dec starting
;else dec bar number
;exit
;dec starting number
;display new directory
;go to Top of Directroy
;is it Home key?
;if not goto next test
;set top of file
;display new directory
;go Bottom of Directory
;is it End key ?
;if not goto next test
;get number of files
;are # of files > 14 ?
;if yes jump to lastpage
;starting position
;hilite last entry.

;maxfiles-13 = starting
;store in BL
;hilite bottom of window
;display directory

;is it the pageup key ?
;if goto next test
;starting # to AL
;subtract 14 = pageup
;char for homekey
;no goto homekey
;change page
;display new page

;is it the pagedn key ?
;set carry flag for ret
;if goto next test
;starting # to AL
;number of dir files
;ptr to top of lastpage
;page = page + 1
;is this the lastpage?
;char for End key
;if yes goto End
;else go Page + 1

```

Output a 14 line of file names to the screen.

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

Note: the hllite 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 hllite.

PROC DISPLAY_FILES

```
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV DL,15 ;convert 1 to 14 into
SUB DL,AH ;14 to 1
MOV AH,[Menu] ;normal attribute
MOV [Color],AH ;set default color
MOV DH,[Warning] ;hllite bar attribute
MOV BX,0520h ;row 6/ col 32
MOV CX,14 ;number of rows
DIS0: CMP DL,CL ;is this the hllite bar
JNZ DIS1 ;< > 0 = no color change
MOV [Color],DH ;if yes color = warning
DIS1: CALL DIR_STR ;display one file name
CMP DL,CL ;is the the hllite bar
JNZ DIS2 ;< > 0 = no color change
MOV [Color],AH ;if yes color = menu
DIS2: INC BH ;ptr to next row
INC AL ;ptr to next dir entry
LOOP DIS0 ;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

—— Move Selected file Name from memory block to FileNa in data segment.

Input BL = starting directory number (1 to MaxFile)

BH = hllite bar number (1 to 14)

Output an ASCIIZ file name string in FileNa in the data section.

PROC MOVE_NAME

```
PUSH AX ;save registers
PUSH BX
PUSH CX
PUSH DX
MOV AX,BX ;membk ptr to AX
ADD AL,AH ;start dir no + hllite
DEC AL ;minus 1 = membk recno
```

```

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                ;membik offset to SI
MOV     DI,Offset FileNa     ;destination offset
MOV     AX,DS                ;assign ES to the
MOV     ES,AX                ;data section
MOV     AX,[DirSeg]          ;membik base ptr
MOV     DS,AX                ;assign membik to DS
CLD                           ;auto inc SI & DI
MOV0:   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 beg 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 FilTyp     ;point to 5 byte string
REP     MOVSB                ;move type to FileNa
CLC                           ;clear carry flag
POP     DX                   ;restore registers
POP     CX
POP     BX
POP     AX
RET
ENDP   MOVE_NAME

```

```

:-----Display the Select a file screen.
:   Input = None
:   Output = Carry Flag if no FIL files in current directory
:

```

```

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

```

```

db      'Press the <Enter> key to select the highlighted file.',0
MOV     AL,[Normal]                ;set Color
MOV     [Color],AL
CALL    CLEAR_TITLE
MOV     AX,0405h
CALL    GOTOYX
CALL    CSTR_OUT
db      'Directory Path : ',0
MOV     AX,Offset Path
CALL    DSTR_OUT
MOV     AX,1500h                   ;row 21,column 0
CALL    MENU_BOX                   ;draw menu box
MOV     AL,[Menu]                  ;set color = menu
MOV     [Color],AL
MOV     AX,1806h                   ;row 23,column 5
CALL    GOTOYX
CALL    CSTR_OUT
db      'Press the <Esc> key to return to the menu without '
db      '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.
db      " files with a type of ",0
JMP     CEE4
CEE1:   CALL    CSTR_OUT            ;singular text mess.
db      "1 file with a type of ",0
JMP     CEE4
CEE2:   CALL    CSTR_OUT            ;zero files statement
db      "no files with a type of ",0
MOV     AX,Offset FilTyp
CALL    DSTR_OUT
CALL    CSTR_OUT
db      " ",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                   ;end 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    "<PageDn> = 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

```

```

RET
ENDP SELECT_WINDOW
ENDP SELECT_SCREEN
ENDP SELECT_FILE

```

```

: 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
    JNC     CRE0                ;continue if no error
    JMP     CRE9                ;exit on DOS error
CRE0: CALL    COUNT_FILES        ;how many SLDI files?
    CMP     AX,251              ;is files found < 251 ?
    JNC     CRE1                ;if no display err mess
    MOV     [MaxFile],AL        ;save number of files
    JMP     CRE2                ;if yes OK! continue
CRE1: MOV     AX,030Bh           ;row 3 / col 1
    CALL    GOTOYX              ;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    GOTOYX              ;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    GOTOYX              ;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     CRE8                ;exit to many files
CRE2: CMP     AL,0              ;were any files found ?
    JZ      CRE8                ;if no files Exit
CRE3: CALL    GET_DIR_BLK        ;allocate memory blk
    JC      CRE9
    CALL    MAKE_DIR
    JC      CRE9
    CALL    SHELL_SORT
CRE8: CLC                      ;clear carry flag
CRE9: POP     ES

```

```

    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                      ;is memblk allocated ?
    JZ      COP4             ;set carry if error
                                ;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 ASCIIZ
    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               ;distination 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

```

```

;-----move one file name
MOV SI,9Eh
;max length of Name
MOV CX,12
COP1: MOV AL,[SI]
CMP AL,0
JZ COP2
CLD
MOVSB
LOOP COP1
;-----pad end of file name with spaces.
COP2: ADD CX,2
MOV AL,20h
COP3: MOV [ES:DI],AL
INC DI
LOOP COP3
;-----find next match
MOV AX,4F00h
INT 21h
JNC COP0
CLC
COP4: POP ES
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
PUSH BX
PUSH CX
PUSH DX
MEM1: MOV BL,[MaxFile]
XOR BH,BH
INC BX
MOV AH,48h
INT 21h
JC MEM2
MOV [DirSeg],AX
JMP SHORT MEM3
MEM2: MOV CL,[Color]

```

;80h = DTA; 1Eh = offset
 ;DTA + offset = 9Eh
 MOV CX,12
 ;max file name length
 ;load byte to be moved
 ;is it end of string ?
 ;if end exit loop
 ;auto inc SI & DI
 ;copy file name
 ;number of bytes
 ;space to AL
 ;place space in dir
 ;ptr to next byte
 ;loop until CX = 0
 ;fine next file function
 ;do next search
 ;loop until all found
 ;clear carry flag
 ;restore registers
 ;get number of files
 ;zero high byte
 ;get an extra paragraph
 ;allocate men function
 ;request memory block
 ;jump if memory error.
 ;base address of seg
 ;normal exit of proc.
 ;save original color

```

MOV     AL,[Warning]           ;warning color
MOV     [Color],AL             ;set color
MOV     AX,0101h               ;row 1/Col 1
CALL    GOTOYX                 ;position cursor
CALL    CSTR_OUT               ;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
        POP     CX
        POP     BX
        POP     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      REL0            ;exit 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.
REL0:   POP     ES              ;restore registers
        POP     DX
        POP     CX
        POP     BX
        POP     AX

```

```

RET
ENDP  RELEASE_MEM

```

```

; 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           ;source offset
    MOV     DI,Offset Search         ;destination offset
    CLD                               ;auto inc DI and SI
COU1:  MOVSB                          ;copy one byte
    CMP     BYTE PTR [SI],0          ;is next char = 0
    JNZ     COU1                     ;copy bytes
;-----place "\" after path name
    MOV     AL,'\'
    MOV     [DI],AL
    INC     DI
;-----copy search name to end of path
    MOV     SI,Offset SearNa        ;source ptr
    MOV     CX,13                   ;number of bytes
    CLD                               ;auto inc SI & DI
    REP     MOVSB                     ;copy all 13 bytes
;-----search for key files
    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 ASCIIZ
    INT     21h                       ;do the first search
    JC      COU6                     ;if no match exit
COU5:  INC     BX                     ;found:increase counter
    MOV     AX,4F00h                 ;fine next file function
    INT     21h                       ;do next search
    JNC     COU5                     ;loop until not found
COU6:  MOV     AX,BX                 ;file count to AX
    CLC                               ;clear carry flag
    POP     DX                       ;restore registers
    POP     CX
    POP     BX
    RET

```

```

ENDP  COUNT_FILES

```

```

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

```

```

:      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+Gap] then
:           Begin
:               Temp := A[J];
:               A[J] := A[J+Gap];
:               A[J+Gap] := Temp;
:               J := J-Gap;
:           End
:           Else J := 0;
:       End;
:       End;
:       Gap := Gap DIV 2;
:       End;
:       End;
:       The follow registers hold the above variables:
:       AX = Gap; BX = J; CX = I; DX = MaxRec; and BP = temp storage

```

```

PROC  SHELL_SORT
    PUSH  AX                      ;save registers
    PUSH  BX
    PUSH  CX
    PUSH  DX
    PUSH  DS
    PUSH  ES
    PUSH  BP
    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                 ;Gap = MaxRec
    SHR    AX,1                  ;Gap = Gap Div by 2
SHEL1: CMP    AX,0               ;when Gap = 0 exit.
    JLE    SHEL4                 ;exit if <= 0
    MOV    CX,AX                 ;I is stored in CX
    INC    CX                    ;I = Gap + 1
SHEL2: MOV    BX,CX              ;J in BX
    SUB    BX,AX                 ;J = I - Gap
    JZ     SHEL3                 ;skip if J = 0
    JC     SHEL3                 ;skip if J is < 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                  ;Gap = Gap Div by 2
    JMP    SHORT SHEL1
SHEL4: POP    BP                 ;restore registers
    POP    ES
    POP    DS
    POP    DX
    POP    CX

```

```

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 6 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

```

ENDP SHELL_SORT

```

:-----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 = ASCIIZ 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     DIR0                   ;if in bounds jump
    MOV     AL,AH                  ;else make blank file
    DIR0:   MOV     CL,4            ;shift 4 = times 16
    SHL     AX,CL                  ;multi by 16
    MOV     SI,AX                  ;ASCIIZ message ptr SI
    MOV     AX,BX                  ;row/col to AX
    CALL    GOTOYX                 ;position cursor
    MOV     DI,Offset Input        ;ptr to input string
    MOV     AX,DS                  ;place data seg
    MOV     ES,AX                  ;in the ES register.
    MOV     AX,[DirSeg]            ;place the memory blk
    MOV     DS,AX                  ;seg in DS.
    MOV     CX,8                   ;8 words = 16 bytes
    CLD
    REP     MOVSW
    MOV     AX,ES                  ;restore reg DS to
    MOV     DS,AX                  ;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    DSTR_OUT               ;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

```

;---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,020Bh            ;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 hlight = 1

XOR AX,AX ;zero to register

MOV [DirSeg],AX ;set memory bock to 0

MOV [MaxFile],AL ;set maxfiles to 0

CLC ;clear carry flag

REL2: POP ES

```

        POP    DX                      ;restore registers
        POP    CX
        POP    BX
        RET
ENDP    RELEASE_MEM_DIR

:-----Read the files DOS date to the [Date] string
        Input = None
        Output = files 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 ext
        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
        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 in [Date]

```

```

:      Output = two byte number into [Date] string
:
PROC   CONVERT_ASCII
    PUSH    AX
    PUSH    BX
    PUSH    CX
    PUSH    DX
    CMP     AX,100                ;is it a 2 digit number?
    JC      COV1                  ;if yes continue else
    XOR     AX,AX                 ;set number to 00
    COV1:   MOV    CL,10          ;divisor
    DIV     CL                   ;AX/10
    OR      AX,3030h             ;convert to ASCII
    COV2:   MOV     [BX],AX       ;place in Date string
    CLC                                ;clear carry flag
    POP     DX
    POP     CX
    POP     BX
    POP     AX
    RET
ENDP   CONVERT_ASCII
ENDP   READ_DATE
:

```

```

:-----Set search file name variables for file type .NEG or .POS
:      Input = AX = none
:      Output = Adjust the following strings [FITyp], [FileNa] and [SearNa]
:

```

```

PROC   SET_TYPE
    PUSH    AX
    PUSH    BX
    PUSH    CX
    PUSH    DX
    MOV     AX,DS                ;set ES = DS
    MOV     ES,AX
    MOV     BX,Offset Postyp     ;ptr to Positive string
    CMP     BYTE PTR [Report],0 ;is it a POS report?
    JZ      STT1                 ;if YES goto STT1
    MOV     BX,Offset Negtyp     ;if NO ptr to NEG string
STT1:   MOV     DX,2             ;save byte counter
    CLD                          ;auto inc SI & DI
    MOV     DI,Offset FITyp      ;destination pointer
    MOV     SI,BX                ;source pointer
    MOV     CX,DX                ;loop counter = 2
    REP     MOVSW                ;move two Words
;clear carry flag
    MOV     DI,Offset FileNa + 8 ;destination pointer
    MOV     SI,BX                ;source pointer
    MOV     CX,DX                ;loop counter = 2
    REP     MOVSW                ;move two Words
;clear carry flag
    MOV     DI,Offset SearNa + 8 ;destination pointer

```

```

MOV     SI,BX                ;source pointer
MOV     CX,DX                ;loop counter = 2
REP     MOVSW                ;move two Words
;clear carry flag
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                ;assign ES = 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 BegOfFile
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,0FFh              ;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 FilBuf     ;ptr to data file line
CLD                          ;auto inc DI and SI
REPZ    CMPSB                ;are the bytes = ?
JNZ     FZR3                 ;if NO goto next test

```

CALL COPY PERCNT	;read in ranking var
JMP SHORT FZR5	;do not count '000'
;-----is this an ID data line	
FZR3: MOV SI,Offset FilBuf	;ptr to data file line
MOV CX,3	;loop counter
FZR4: MOV AL,[SI]	;get first byte
CMP AL,'0'	;is it < ASCII 0
JC FZR5	;if Yes read next line
CMP AL,':'	;is it a digit?
JNC FZR5	;if No read next line
INC SI	;point to next type
LOOP FZR4	;check next byte
INC BX	;YES it is an ID number
;-----is this the last line ?	
FZR5: XOR AL,AL	;zero AX register
CMP AL,[EOF]	;is EndOfFile TRUE?
JZ FZR1	;False = get next line
MOV [MaxNo],BX	;save number of ID's
CLC	;clear cf = found
FZR8: POP DX	;restore registers
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	;true marker
MOV [Ranked],AL	;mark [ranked] TRUE
MOV CX,192	;max no of variables
MOV BX,Offset FilBuf + 3	;beyond 000
MOV SI,Offset PerCnt + 5	;ptr to PerCnt variables
GES1: MOV AL,[BX]	;read word
CMP AL,0	;is this EndOfString?
JZ GES5	;if YES then stop loop
CMP AL,':'	;is it a space ?
JZ GES2	;if NO check range else
;-----is value an ASCII digit ? 0 - 9	
CMP AL,'0'	;is value < ASCII 0
JC GES3	;if Yes then error
CMP AL,':'	;is value a digit ?
JNC GES3	;if NO then error
;-----save digit in PerCnt variable	

```

    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
;-----were 80 two byte ASCII variables found?
; 20 var per dim and 4 dim = 80 two digit variables or 160 bytes
GES5: MOV     AX,SI              ;get var pointer
    MOV     BX,Offset PerCnt + 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,0207h             ;row 3/Col 8
    CALL    GOTOYX               ;set cursor
    CALL    CSTR_OUT             ;display warning
    db      " Line '000' is incorrect length for 4 "
    db      'diminsions. 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
;

```

.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_SLDI
    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                     ;continuel I found it.
    LOOP    KE1                     ;if NO 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.fil handle
;-----set disk drive of open file
    XOR     AX,AX                    ;zero to [DiskDr] =
    MOV     [FileDr],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

```

```

SUB     AL,64                      ;convert to hex value
JC      KE8                        ;if error continue
MOV     [FileDr],AL                ;save drive of file
KE8:    CLC                        ;clear carry flag
KE9:    POP     ES
        POP     DX
        POP     CX
        POP     BX
        RET
ENDP    OPEN_SLDI

:
:
:----- Close the data files used by the Trial program
: Input = None
: Output = None (message displayed and carry flag set on error)
:         File handle stored in [FileHd]
: Note:   Major registers saved.
:
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    GOTOYX                ;set cursor
        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
        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,0209h          ;row 3/Col 12
    CALL    GOTOYX            ;set cursor
    CALL    CSTR_OUT          ;display warning
    db      ' Reading File ',0
    MOV     [Color],CL        ;restore orig. color att
    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,020Fh            ;row 2, column 7
    CALL    GOTOYX
    CALL    CSTR_OUT
    db      'Press the <Enter> key to open the highlighted file.',0
    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 FilBuf    ;mark position in buffer

```

```

MOV BP,Offset FilBuf + 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 endoffile
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 endofline ?
JZ REE4                             ;if YES exit
CMP AL,128                          ;is 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 BP,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 endofline
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

MOV BX,[FileHd]

CMP AX,BX

;zero register

;is a file open ?

;if not then exit

```

    JNZ TOP1                ;if open goto next test
    STC                    ;else set error flag
    JMP SHORT TOP2          ;exit on error
;---place file point to the beginning of the file
TOP1: MOV CX,AX             ;set offset = 0
    MOV DX,AX              ;set offset = 0
    MOV AX,4200h           ;set file pointer no.
    INT 21h               ;set to beg. of file
    JC TOP2               ;exit if error.
    XOR AL,AL             ;zerp to register
    MOV [EOF],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 to 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,3600h           ;dis space function
    INT 21h               ;get disk space
    CMP AX,0FFFFh         ;is drive valid?
    JZ ISF3               ;if NO exit error
    CMP BX,BP              ;avail cluster > bytes
    JNC ISF4               ;yes OK! lots of room
    MUL BX                 ;get available sectors
    CMP DX,BP              ;is avail sectors/65000
    JNC ISF4               ;greater then bytes ?
    CMP AX,BP              ;is avail sectors > bytes
    JNC ISF4               ;if Yes lots of room
    MUL CX                 ;get available bytes
    CMP DX,BP              ;is avail bytes/65000
    JNC ISF4               ;greater then bytes ?
    CMP AX,BP              ;is avail bytes > bytes
    JNC ISF4               ;if YES exit OK!
    CALL FULL_ERR          ;else inform user
ISF3: STC
    JMP SHORT ISF5
ISF4: CLC

```

```

ISFS: POP    BP
      POP    DX
      POP    CX
      POP    BX
      POP    AX
      RET

:
:   Input = none
:   Output = none
PROC  FULL_ERR
      CALL    CLEAR_MESSAGE
      MOV     AL,[Warning]           ;warning color
      MOV     CL,[Color]             ;save original color
      MOV     [Color],AL             ;set color
      MOV     AX,0209h               ;row 3/Col 8
      CALL    GOTOYX                 ;set cursor
      CALL    CSTR_OUT               ;display warning
      db      'Not enough Disk Space to save the rankings.'
      db      'Press Any Key. ',0
      MOV     [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     GEP1                   ;if NO then display Path
:-----get default drive             ;if YES get default path
      MOV     AH,19h                 ;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
      MOV     AH,47h                 ;get current path
      MOV     DL,0                   ;on default drive
      INT     21h                    ;get path
GEP1:  CALL    EDIT_PATH

```

```

    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
    MOV     BX,DS
    MOV     ES,BX
    CALL    PATH_TO_INPUT      ;move Path str to Input
    CALL    PATH_MESS_TOP
    CALL    PATH_MESS_BTM     ;edit message
EDT1:  MOV     AX,0106h
    CALL    PATH_EDITOR       ;edit this field
    JC      EDT2              ;exit on <Esc> key
    CALL    CHECK_PATH        ;if valid save path
    JC      EDT1              ;if Not valid loop
    JMP     SHORT EDT3        ;exit path OK!
EDT2:  CALL    MENU_INSTRU    ;draw bottom box
    STC
EDT3:  POP     DX
    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,0206h
    CALL    GOTOYX
    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 ASCIIZ 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                      ;save original str ptr
    PUSH  CX
    PUSH  DX
    MOV   BX, Offset Input        ;pointer to input str
    MOV   CX, 12                 ;string length - 1
TRI1:  MOV  AL, 'A'               ;is character less than
    CMP   [BX], AL               ;the letter "A" ?
    JNC   TRI3                   ;if yes remove character
TRI2:  CALL DELETE_CHAR          ;shift string left
    DEC   BX                     ;check same byte again
    JMP   SHORT TRI4
TRI3:  MOV  AL, 'Z'               ;is character greater
    CMP   AL, [BX]               ;than letter "Z" ?
    JC    TRI2                   ;if yes remove character
TRI4:  INC  BX                     ;ptr to next byte
    LOOP  TRI1                   ;check next byte
:-----convert trailing spaces to hex 0.
    MOV   CX, 13                 ;loop counter
    MOV   BX, Offset Input + 12 ;ptr to LastByte
    MOV   AX, 20h                ;AH = hex 0 AL = space
TRI5:  CMP  AL, [BX]              ;is char a <space> ?
    JNZ   TRI6                   ;if no exit.
    MOV   [BX], AH               ;mark as end of string
    DEC   BX                     ;ptr to last byte
    LOOP  TRI5                   ;loop until beg of str
TRI6:  POP  DX
    POP   CX
    POP   BX
    POP   AX
    RET
:----- delete a character at the cursor
PROC  DELETE_CHAR
    PUSH  AX
    PUSH  BX                      ;save original str ptr
DEP1:  MOV  AX, [BX]              ;read ptr BX and BX+1
    CMP   AH, 0                  ;is it the end of str?
    JZ    DEP2                   ;if yes then done.
    MOV   [BX], AH               ;place BX+1 in BX
    INC   BX                     ;point to next byte
    JMP   SHORT DEP1             ;loop until end of str.
DEP2:  MOV  AH, ' '              ;place a <space> at

```

```

    MOV    [BX],AH                ;end of the string.
    POP    BX                    ;restore original ptr
    POP    AX
    RET
ENDP    DELETE_CHAR
ENDP    FILTER_FIELD
:
:-----Instructions for entering the path name.
:   Input = None
:   Output = None
:
PROC    PATH_MESS_BTMT
    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,180Ah                ;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 '
    db      'the Menu.',0
    MOV     [Color],CL                ;restore orig. color
    POP     DX                    ;restore registers
    POP     CX
    POP     BX
    POP     AX
    RET
ENDP    PATH_MESS_BTMT
:
:-----Copy path ASCII string in [Path] to [Input].
:   Input = none
:   Output = none
:   AX-DX register saved.
:
PROC    PATH_TO_INPUT
    PUSH    AX
    PUSH    BX                    ;save registers
    PUSH    CX
    PUSH    DX
:-----fill [Input] with 68 spaces
    MOV     AX,DS                ;Make ES = DS

```

```

MOV     ES,AX
MOV     CX,67                                ;max number of chrs
MOV     BX,Offset Input                      ;pointer to ASCIIZ str
MOV     AL,' '                               ;space AL register
MOV     [BX],AL                             ;0 to first byte of str
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 string with 0's
;-----copy current [path] to [Input]
MOV     DI,Offset Input                     ;destination offset
MOV     SI,Offset Path                      ;source offset
CLD                                          ;auto inc DI and SI
EDT0:   MOVSB                               ;copy one byte
CMP     BYTE PTR [SI],0                    ;is next char = 0
JNZ     EDT0
CLC
POP     DX
POP     CX
POP     BX
POP     AX
RET

```

ENDP PATH_TO_INPUT

```

;----- Get an ASCIIZ 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'
; ASCIIZ 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                                ;save registers
PUSH     CX
PUSH     DX
MOV     DX,AX                             ;save row/column in DX
CALL    GOTOYX                             ;set cursor position
MOV     BX,Offset Input                    ;ptr to [Input] str.
MOV     AX,BX                             ;[Input] ptr to AX
CALL    DSTR_OUT                           ;Display blanks
;-----find first space in string
MOV     SI,0                               ;zero to SI
PATA:   INC     SI                          ;ptr to next char
CMP     SI,67                              ;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 GOTOYX	;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	;handel 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	
JC PAT0	;carry flag = not char
;-----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	;clear carry flag
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	

```

PUSH    AX                      ;save registers
PUSH    BX
PUSH    CX
PUSH    DX
MOV      DL,[Color]             ;save current color
MOV      AL,[HiLite]            ;set color for insert
MOV      [Color],AL            ;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
db      ' ',0                  ;the screen.
JMP      SHORT PAH2             ;exit.
PAH1:    CALL     CSTR_OUT       ;send the following
db      '<Insert On>',0         ;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,08h                ;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.

```

```

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
      PUSH   BX
      MOV    AL,[BX]
      INC    BX
SHI1:  MOV    AH,[BX]
      CMP    AH,0
      JZ     SHI2
      MOV    [BX],AL
      MOV    AL,AH

```

```

;is it the End key?
;if not continue.

```

```

;is it the delete key?
;if not continue.
;delete char at cursor.

```

```

;is it a left arrow key
;if not continue.
;beginnig of the string?
;yes = beg. of line
;so loop will continue.

```

```

;is it Rt Arrow key?
;if not jump.

```

```

;is 'end of string' ?
;if = 0 no right
;advance pointer

```

```

;check for end of str.
;zero = end of string
;ret on end of string
;advance pointer
;advance cursor

```

```

;save new character
;save str pointer
;load char to be moved
;ptr to the next char
;load next char.
;is it the end of str?
;if yes then Exit.
;last char in the str.
;next char to last char

```

```

        INC     BX                      ;ptr for new next 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                      ;save original str ptr
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 ignor 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    GOTOYX                 ;set cursor position
        MOV     AX,BX                  ;str pointer to AX
        CALL    DSTR_OUT                ;display string
BA3:    POP     AX
        RET
ENDP    BACKSPACE
;

```

```

;-----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,0DFh               ;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                 ;else remove space
    MOV     DI,BX                 ;offset to 1st byte
    MOV     SI,BX
    INC     SI                    ;offset to 2nd byte
    CLD                          ;auto inc DI and SI
    REP     MOVSB                 ;shift line left
    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                    ;ptr to last char
    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?

```

```

CHE3: 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
      MOV    [BX],AX
      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

;-----save valid path string 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
      MOV    [DI],AL
      CLC                        ;zero = EndOfString
                                ;clear carry flag
CHE7: POP     DX
      POP     CX
      POP     BX
      POP     AX
      RET

;-----Is this a Valid path?
; Input = ASCIIZ drive/directory string in [Input]
; Output = carry flag in not a valid path
; AX - DX registers saved
PROC  IS_PATH
      PUSH    AX                ;save registers
      PUSH    BX
      PUSH    CX
      PUSH    DX
;-----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,'*\'
      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 ASCIIZ 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,' '                   ;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      ',0'
MOV     [Color],CL               ;restore original Color
POP     DX                      ;restore registers

```

```

        POP    CX
        POP    BX
        POP    AX
        RET
ENDP    PATH_ERROR
ENDP    CHECK_PATH
;

```

.CODE

```

; 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    GOTOYX
        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

```

```

;-----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
;ptr to string to edit
;ptr to string to fill
;ASCII zeros
;place 1st two bytes
;zero = end of string
;advance string ptr
;ASCII 0 and hex 0

```

```

;-----edit ID string

```

```

        MOV     AX,0132h
        CALL    NUMBER_EDITOR
ID8:    POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
;row/col position
;edk this field
;restore registers

```

```

ENDP    GET_ID
;

```

```

;----- Get an ASCIIZ string input from the keyboard.

```

Input = AX = Row/Column position on the screen
 [ID] must contain the string to be edited before
 calling this subroutine.

OutPut = ASCIIZ 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 = Starting 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              ;save 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,D $\bar{X}$             ;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   ;cf = Yes DoNot search
;-----Is it any other control character?
NUE3:  CMP  AL,1Bh        ;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          ;beginnig 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.

```

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
;-----is it the <Esc> key ?	
NUE7: STC	;set carry flag
JZ NUE11	;exit on <Esc> key
;-----filter unwanted characters	
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
;-----place the character in the [ID] string.	
NUE8: MOV [BX],AL	;place char in [ID]
MOV AX,BX	;row/col to AX
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 NUE0	;yes = do not move cursor
INC BX	;advance [ID] ptr.
INC CX	;advance cursor
JMP NUE0	;if not continue input.
NUE10: CLC	;clear carry flag
NUE11: POP ES	
POP DS	
POP DX	
POP CX	
POP BX	
RET	
ENDP NUMBER_EDITOR	
;	
;-----is the ID Number = 000.	
; Input = ASCII ID string in [ID]	
; Output = Carry Flag = Yes	
;	
PROC IS_ZERO	
PUSH AX	;save registers
PUSH BX	
PUSH CX	
PUSH DX	
MOV BX,Offset ID	
MOV CX,3	;loop counter
MOV AL,30h	;ASCII zero value
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

```

CALL    ZERO_MESS                ;NO zero ID numbers
STC                                ;Exit on ID Error
JMP     SHORT ISZ3                ;exit error no search
CALL    ZERO_MESS                ;inform user NO 0 ID's
ISZ2:   CLC                        ;OK! ID number
ISZ3:   POP     DX                ;restore registers
        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    GOTOYX                    ;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  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 ID Number and press the <Enter> key to continue',0
MOV   AX,170Ch             ;row 23,column 13
CALL  GOTOYX
CALL  CSTR_OUT
db    ' or press the <Esc> key to return to the Main Menu.',0
MOV   AX,0207h             ;row 23,column 13
CALL  GOTOYX
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
;
```