



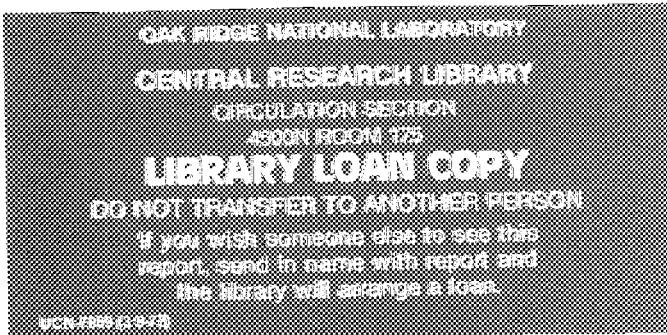
3 4456 0366149 1

ORNL/TM-12096

**ornl****OAK RIDGE  
NATIONAL  
LABORATORY****MARTIN MARIETTA**

A COMPUTER PROGRAM  
INTEGRATING A MULTICHANNEL ANALYZER  
WITH GAMMA ANALYSIS FOR THE  
ESTIMATION OF  $^{226}\text{Ra}$  CONCENTRATION  
IN SOIL SAMPLES

by John E. Wilson



MANAGED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

NTIS price codes—Printed Copy: A03 Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

A COMPUTER PROGRAM  
INTEGRATING A MULTICHANNEL ANALYZER  
WITH GAMMA ANALYSIS FOR THE  
ESTIMATION OF  $^{226}\text{Ra}$  CONCENTRATION  
IN SOIL SAMPLES

John E. Wilson

Date Published—August 1992

Prepared by the  
**OAK RIDGE NATIONAL LABORATORY**  
Bethel Valley Road  
Oak Ridge, Tennessee 37831-6285  
**MARTIN MARIETTA ENERGY SYSTEMS, INC.**  
P.O. Box 2003  
Oak Ridge, Tennessee 37831-7606  
for the  
U.S. Department of Energy  
under contract DE-AC05-84OR21400



3 4456 0366149 1



## CONTENTS

CONTENTS .....	iii
ACKNOWLEDGEMENTS .....	vi
FIGURES .....	viii
EXECUTIVE SUMMARY .....	x
1. INTRODUCTION .....	1
2. SYSTEM DESCRIPTION .....	2
2.1 HARDWARE .....	2
2.1.1 Gamma Detectors and Amplifiers .....	2
2.1.2 The DMR-II™ .....	3
2.1.3 Computer and Associated Hardware .....	3
2.2 SOFTWARE .....	4
2.2.1 Gamma Analysis Program .....	4
2.2.2 Custom Multichannel Analyzer .....	11
2.2.3 Support Features .....	17
3. INSTALLATION PROCEDURES .....	21
3.1 HARDWARE .....	21
3.1.1 Digital Multiplexer Router .....	22
3.1.2 Personal Computer Analyzer .....	22
3.1.3 Digital Multiplexer Timer Card .....	22
3.1.4 Hardware Adjustment .....	23
3.2 SOFTWARE .....	24
3.2.1 Directories .....	24
3.2.2 Program and Files .....	24
REFERENCES .....	27
ACRONYMS AND INITIALISMS .....	28

APPENDIX A	PROGRAM SOURCE CODE
APPENDIX B	DERIVATION OF THE THREE-ROI METHOD FOR ESTIMATING CONCENTRATION OF $^{226}\text{Ra}$
APPENDIX C	ALGORITHMIC DEFINITIONS OF MATRIX OPERATIONS
APPENDIX D	CONTENT DESCRIPTIONS OF FILES USED
APPENDIX E	FILE-NAMING CONVENTIONS
APPENDIX F	PRINTED SPECTRA
APPENDIX G	SAMPLE ANALYSIS, CALIBRATION, AND QUALITY CONTROL PROCEDURES

## **ACKNOWLEDGEMENTS**

The author would like to recognize three individuals of ORNL-PAG for their significant contributions to the development of this program and its subsequent documentation in this memorandum:

Shawn J. Wallace for his careful and patient explanation of the concepts behind gamma spectroscopy and the three-ROI method of sample analysis.

James R. Davidson for his valuable information on gamma radiation detection instrumentation, for his GetEStr function which was incorporated into the program, and for reviewing and testing the program code and its related documentation.

Gretchen A. Pierce for giving thorough guidance to me during development of this program to ensure that it would perform all the necessary tasks as easily and efficiently as possible for her and the other technicians who operate the system.



## **FIGURES**

1.	Interface of hardware components .....	5
2.	Hierarchy of major functions .....	7
3.	MCA screen .....	13
F.1.	Spectrum printed on Hewlett - Packard® LaserJet® - II printer .....	F-1
F.2.	Spectrum printed on Texas Instruments™ Model 855 printer .....	F-2



## **EXECUTIVE SUMMARY**

A new hardware/software system has been implemented using the existing three-regions-of-interest method for determining the concentration of  $^{226}\text{Ra}$  in soil samples for the Pollutant Assessment Group of the Oak Ridge National Laboratory. Consisting of a personal computer containing a multichannel analyzer, the system utilizes a new program combining the multichannel analyzer with a program analyzing gamma-radiation spectra for  $^{226}\text{Ra}$  concentrations. This program uses a menu interface to minimize and simplify the tasks of system operation.



## 1. INTRODUCTION

The Pollutant Assessments Group of Oak Ridge National Laboratory in Grand Junction, Colorado, has been involved for several years in the assessment of the contamination of properties from uranium mill tailings as part of the Uranium Mill Tailings Remedial Action program. At the beginning of this project, a system for rapidly estimating the concentration of  $^{226}\text{Ra}$  in soil samples was needed. To accomplish this, gamma radiation detectors and a stand-alone multichannel analyzer (MCA) were interfaced to a personal computer. The computer received gamma radiation energy spectra from the MCA, calculated the estimated concentrations of the  $^{226}\text{Ra}$ , stored the energy spectra data onto a floppy disk, and transferred the results to a data base.

A new soil analysis program has been written that removes the need for separate MCA and computer hardware and simplifies operator interface with the analysis process. The integrated MCA/gamma analysis program combines an MCA with a gamma analysis program into a single computer program. The program is written specifically to take a gamma energy spectrum from the MCA and, by comparing the sample spectrum to spectra of reference standard materials, determine the concentration of  $^{226}\text{Ra}$  based on the weight of the sample. A system of menus guides the operator through the steps necessary to calibrate and maintain the system, as well as to analyze the actual samples.

The MCA hardware, a plug-in computer board, is part of the personal computer on which the program runs. This configuration is an improvement in terms of simplicity and reliability over systems that connect a separate MCA to the computer with communication lines. Furthermore, the program itself is an improvement over other systems that have one program to control the MCA and another to analyze the spectra. With the MCA/gamma analysis integrated program, these programs are combined for ease and efficiency of operation.

---

\*Research sponsored by U.S. Department of Energy, under contract DE-AC05-84OR-21400 with Martin Marietta Energy Systems, Inc.

The program was developed for two major reasons:

1. To make use of newer, more reliable MCA hardware available as computer plug-in boards. The gamma analysis system being used by ORNL-PAG had become unreliable due to equipment failure.
2. To streamline and simplify the manual processes required for collecting and analyzing gamma spectra. The old system required the operator to control an MCA and a computer for analyzing and saving the spectra on computer disks and to coordinate data communication between the two.

## 2. SYSTEM DESCRIPTION

### 2.1 HARDWARE

The soil analysis system consists of three major hardware subsystems: 1) the gamma radiation detectors and signal amplifiers, 2) the Digital Multiplexer Router (DMR-II™), and 3) the computer and associated hardware.

#### 2.1.1 Gamma Detectors and Amplifiers

The system uses three 9-inch thallium-doped sodium iodide [NaI(Tl)] crystal detectors housed in lead-shielded "pig" type containers. Each of these detectors is attached to an EG&G ORTEC® model 113 preamplifier. The output from each of these preamplifiers is attached via shielded cable to the input of a Tennelec model TC240 shaping amplifier.

### 2.1.2 The DMR-II™

The DMR-II™ is a Digital Multiplexer Router Acquisition System made up of the following components:

#### DMR-II™ Unit:

The DMR-II™ Unit (model DMR™-108) accepts input from up to 8 shaping amplifiers and multiplexes them to a single Personal Computer Analyzer (PCA-II™) card. The PCA-II™ card provides storage of the count data in 8 memory input groups (the number of memory groups on the PCA-II™ board is programmable).

#### PCA-II™ MCA Card:

The PCA-II™, an MCA card designed for installation in IBM®-compatible personal computers, contains a 100 MHz Wilkinson analog to digital Converter (ADC) and 8192 channels of count memory (each channel can count up to 16,777,215). The channel conversion gain can be set from 256 to 8192 channels in binary increments. The memory counters can be programmed to count up (add mode) or count down (subtract mode). Two modes are available: pulse height analysis (PHA) or multichannel scaling (MCS).

#### DMR-II™ Timer Card:

The DMR-II™ Timer Control Board connects to the DMR-II™ unit and is installed next to the PCA-II™ card in the computer. This board allows each input to be programmed with an independent preset collection time.

### 2.1.3 Computer and Associated Hardware

An IBM® Personal Computer AT (or compatible) must be used with this system. The DMR™ hardware and the analysis program were designed to operate on this type of computer. Computer requirements are: a floppy disk drive

installed as device A:, a fixed hard disk drive, and access to a printer. Our system is connected directly to a Texas Instruments™ model 855 printer and has access to a Hewlett-Packard® LaserJet®-II printer on a local area network.

The ORTEC® preamplifiers are mounted on the pigs themselves. The TC240 amplifiers and the DMR™ 108 multiplexer units are all housed in a standard NIM bin. The PCA-II™ and DMR-II™ timer cards are installed in the expansion slots of the computer. Figure 1 diagrams the interface of the hardware components.

## 2.2 SOFTWARE

A stand-alone program that runs on the DOS operating systems, the program was developed using the Borland® C++ development system. Source code consists of 17 files: five C header files, eleven C source-code files, and one assembly language source-code file. See Appendix A for complete listings of these files.

### 2.2.1 Gamma Analysis Program

The major function of the program is to analyze a gamma radiation spectrum collected from a soil sample and estimate the concentration of  $^{226}\text{Ra}$  in the sample. The program also estimates the concentrations of two other elements used to calibrate the system:  $^{40}\text{K}$  and  $^{232}\text{Th}$ . The program specifications for the gamma analysis can be found in the C source-code file named ANALYZE.C.

In the program, a gamma spectrum is stored as an array of 512 numbers (numbered 0 to 511), each number representing one channel of the gamma spectrum. A channel refers to the count of gamma rays detected by the NaI(Tl) crystal at a particular energy level. Each channel represents 6 KeV in the gamma energy spectrum. The KRT method of analysis (which stands for potassium, radium, thorium) uses three regions of interest (ROIs) from the gamma spectrum, one region characteristic of each of the three reference elements. An ROI is the

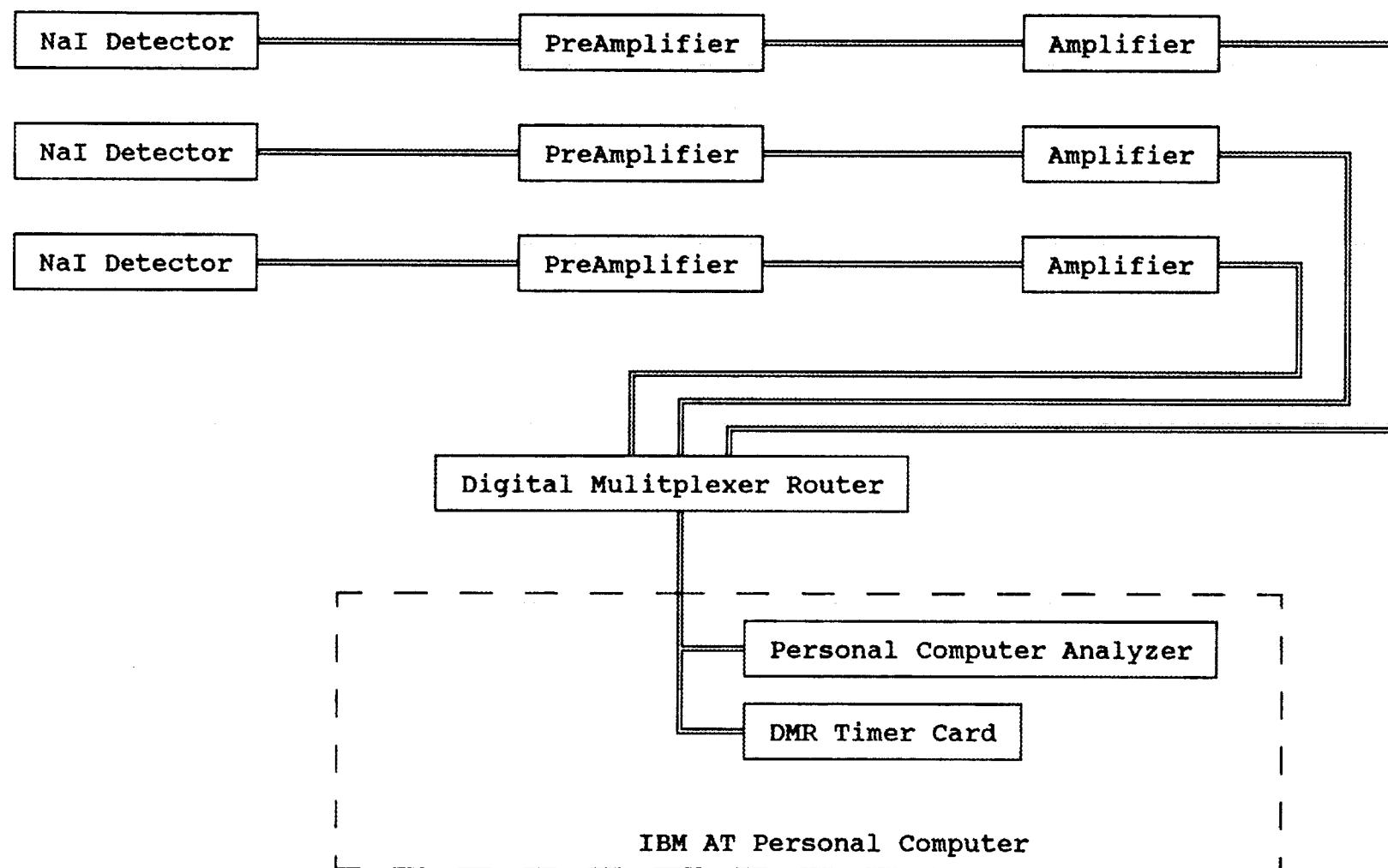


Fig. 1. Interface of hardware components.

sum of the counts from a series of channels. Beginning and ending channel numbers for each of the three ROIs are defined in the file SOIL.H.

The program function named "Sample" controls the collection and analysis of gamma spectra (see Fig. 2). Before the sample menu can be reached, a reference file must be loaded (described below). The sample analysis process proceeds as follows:

1. The operator chooses the "Start" option from the sample menu. The program accepts from the operator, via the keyboard, the sample weight and number of days the sample has been in the container. The program then resets all channels to zero, triggers the PCA™ hardware to collect counts, and starts a timer.
2. While the hardware is acquiring counts, the operator may choose the "View" option from the sample menu to view the spectrum as it is collected on the custom MCA screen. If the operator alters the spectrum in any way during its collection, the program notes that the spectrum is invalid and will not perform any analysis of it.
3. When the timer reaches 300 "live" seconds, counting by the PCA™ hardware is halted. If the operator has not modified the spectrum in any way, the program executes the function "Update\_Samp", which sums the counts from the three ROIs, subtracts the reference background counts, and puts the results into the matrix SAMP. The program notes that the spectrum may now be analyzed.
4. The operator may now choose the "Analyze" option from the sample menu at which time the program executes the function "Calc\_Conc" to analyze the sample. The following sequence of steps outlines the Calc\_Conc function:

## HIERARCHY OF MAJOR FUNCTIONS

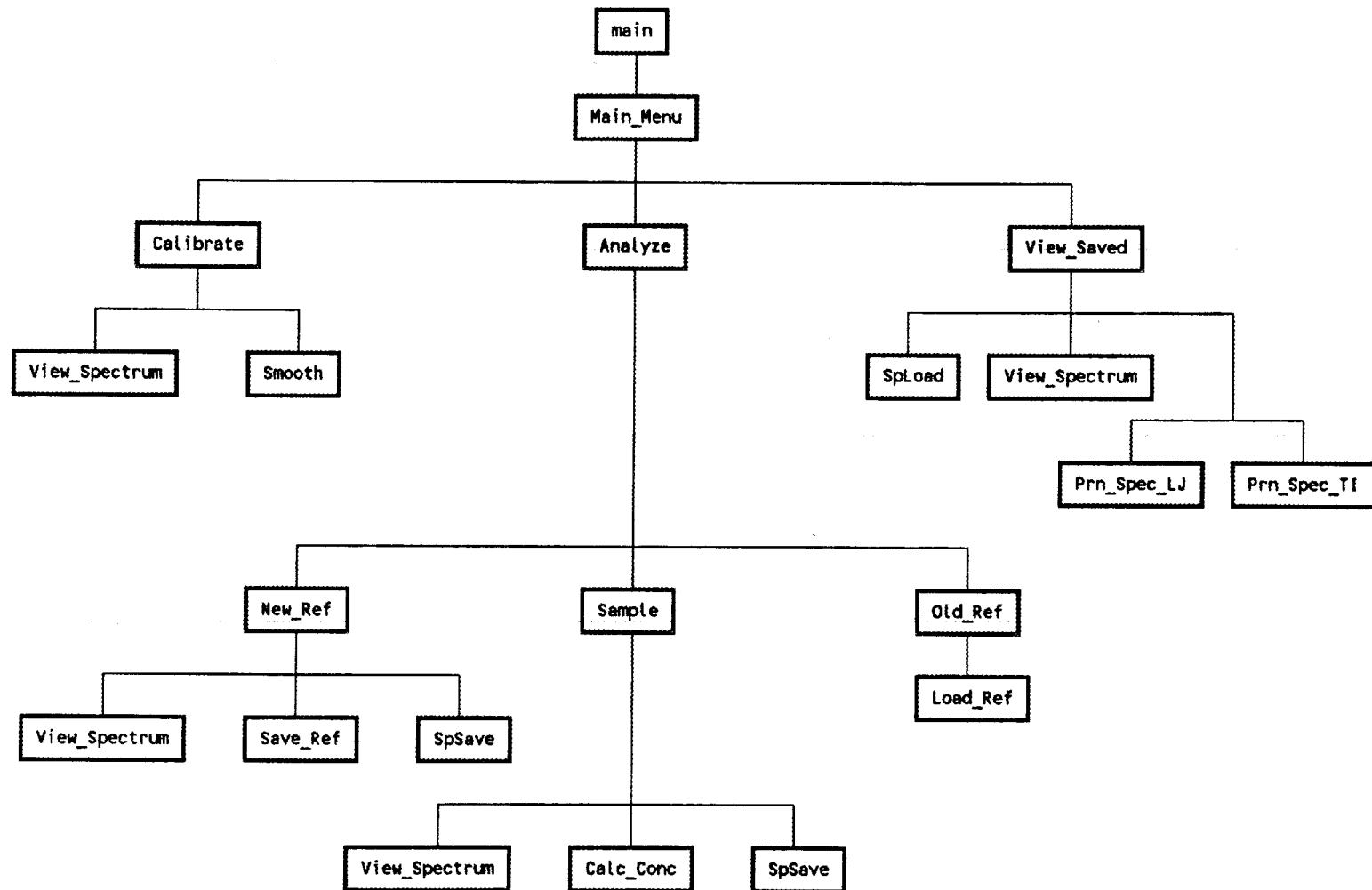


Fig. 2. Hierarchy of major functions.

#### A. Calculation of the concentration estimation

The concentration is estimated using the KRT three ROI method, fully described in Appendix B. The many matrix operations used in this function are documented in Appendix C, and the code specifications of these operations are contained in the file MATRIX.C.

- 1) The inverse of the reference matrix (REF) is put into matrix INV.
- 2) The proportionality constant matrix (CON) is multiplied by INV, and the result is put into matrix PROINV.
- 3) PROINV is multiplied by matrix SAMP, and the result is put into matrix RESULT.
- 4) RESULT is scaled by the reciprocal of the sample weight and the acquisition time. Any element of RESULT that is negative is clipped at zero.
- 5) To adjust for radon ingrowth, RESULT is scaled by a factor from a table based on the number of days the sample has been in the container.
- 6) Decision limits are calculated and placed into matrix LC.
- 7) Detection limits are calculated and placed into matrix LD.
- 8) PROINV is multiplied by LD, and the result is placed in the minimal detectable activity matrix MDA.
- 9) MDA is scaled by the reciprocal of the sample weight and the acquisition time.
- 10) Each element in the RESULT matrix is compared to the corresponding element in the MDA matrix. If any element of RESULT is less than the element in MDA, the concentration estimate corresponding to that element of the array is flagged as being below minimal detectable activity levels.

**B. Calculation of the error estimation**

The error file (ERFILE), the contents of which are outlined in Appendix D, is opened for reading. For each of the three analytes, the following steps are performed:

- 1) The first line of the error file is positioned to be read.
- 2) A line is read from the error file. This line contains the pig number, the analyte number, the maximum and minimum concentrations, the error type, and the error amount.
- 3) If the pig number matches the input that this sample was collected from, continue; otherwise, skip to step 8.
- 4) If the analyte number matches the current analyte whose error estimate is being calculated, continue; otherwise, skip to step 8.
- 5) If the concentration estimation falls within the range of minimum and maximum concentrations continue; otherwise, skip to step 8.
- 6) If the error type is "C", then the error is a constant, and the error amount is the error estimation for this analyte. If the error type is "P", then the error amount is a proportion, and the concentration estimate is multiplied by the error amount to determine the error estimate.
- 7) Repeat steps 1 through 6 for the next analyte. The process is complete when each of the three analytes has an error estimate.
- 8) Position error file to read the next line and go back to step 2 until the end of the file is reached. If the end of the file is reached before a match is made, an error message is displayed, and the Calc\_Conc function is aborted.

### C. Display and confirmation of results

- 1) All concentration and error estimates are rounded off to three significant digits.
- 2) The concentration estimate, minimum detectable activity flag, and error estimate are printed to the screen for each of the three analytes.
- 3) The operator then chooses whether or not to save the data. If the operator chooses not to save the data, the "Analyze" routine is complete.
- 4) The operator chooses which of two files (TEMPDB.DAT or TEMPDB.BAD, described below) will receive the results of the analysis.

### D. Saving the gamma spectrum and the analysis results

- 1) This program gives each gamma spectrum saved to disk a unique name. The naming convention is described in detail in Appendix E. Each time a new spectrum is named, the last spectrum file extension (consisting of three letters) is retrieved from the file named LATEST.
- 2) Once the gamma spectrum name is determined, the extension name is incremented and written back to the LATEST file.
- 3) The analysis results are printed on a single line at the attached printer.
- 4) The gamma spectrum is written to floppy disk (drive A:) using the file name created above. This gamma spectrum file conforms to the standards used by Nucleus, Inc., the supplier of the PCA™ and DMR™ hardware. The C language format of these files is defined in the header file DMR.H. The format of these files is further described in

## Appendix D.

- 5) The analysis results are written to the file TEMPDB.DAT or TEMPDB.BAD, whichever is chosen by the operator. See Appendix D for the contents of these files. Used to store analysis results until they can be transferred to a data base for permanent storage, these files are BASIC-compatible ASCII text files that can be retrieved by a number of software products. For our purposes the results are imported from these files into a KnowledgeMan® data base.
- 6) The name of the spectrum is saved in computer memory to be displayed on the custom MCA screen. The spectrum name is also displayed as the last step of the Calc\_Conc function.

A procedure for sample analysis is outlined in Appendix G.

### **2.2.2 Custom Multichannel Analyzer**

The second portion of the program is the custom MCA. The program specifications that comprise the MCA are found in the C source-code file VIEWSPEC.C. The controlling function "View\_Spec", oversees setting up the graphics mode, drawing the screen, displaying the spectrum in real time, and interpreting user commands for manipulating the screen.

View\_Spec uses the  $640 \times 350$  pixel screen mode of enhanced graphics adapter (EGA) or virtual graphics array (VGA) hardware. The routines for controlling and writing data to the screen are provided in a run-time library called EGAVGA.BGI from Borland International, Inc.

The program stores three separate gamma energy spectra, one for each input pig. The PCA™ and DMR™ hardware allow up to eight independent inputs to be used. The custom MCA allows the user to view and manipulate any one of the three gamma energy spectra. Since the settings for each spectrum are stored

separately from the other two, each individual spectrum may be manipulated independently.

The MCA screen displays many important attributes of the gamma spectrum, as well as prompting for commands (see Fig. 3). The important areas of the screen are:

1. INPUT #: displays the number of the spectrum being viewed.
2. Spectrum box: displays the gamma energy spectrum. Each pixel left to right indicates a separate channel. Pixels up and down indicate how many counts have been collected in each channel. Also displayed is the cursor, a vertical line marking an individual channel. Channels in a region of interest are displayed in a color different from the rest of the channels.
3. Function key prompts:
  - a. F1: START/STOP to display whether hardware is acquiring count data or not, and prompt to show which will occur if the F1 key is pressed.
  - b. F2: CLEAR prompt to zero the spectrum.
  - c. F3: ROI BEGIN prompt to set the first channel of a region of interest.
  - d. F4: ROI END prompt to set the last channel of a region of interest.
  - e. F5: PREV ROI prompt to make the previous ROI the current ROI.
  - f. F6: NEXT ROI prompt to make the next ROI the current ROI.
  - g. F7: CLR ROIs prompt to remove all regions of interest.
  - h. F8: DOT/FILL prompt to show whether spectrum box is in DOT or FILL mode and prompt to show which mode will start if the F8 key is pressed.
  - i. ESC: EXIT prompt to exit the MCA screen, back to calling menu.

INPUT #1

F1: START

F2: CLEAR

F3: ROI BEGIN

F4: ROI END

F5: PREV ROI

F6: NEXT ROI

F7: CLR ROIs

F8: DOT

ESC: EXIT

CHANNEL: 241 VERTICAL SCALE: LOG HORIZONTAL SCALE: 1X

COUNTS: 700

LIVE TIME: 0

FILE NAME: A:911122A1.BTN

	BEG	END	SUM
> ROI1:	273	412	1754
ROI2:	226	259	13555
ROI3:	426	511	804

Fig. 3. MCA screen.

Note: pressing the F1 or F2 keys invalidates the spectrum for analysis purposes.

4. CHANNEL: displays which channel (0-511) the cursor is on.
5. COUNTS: displays how many counts are in the channel under the cursor.
6. LIVE TIME: displays the total live time (in seconds) that the current spectrum has collected count data.

Note: Live time refers to the elapsed time during which the spectrum is collected minus the time during which the PCA hardware was analyzing pulses and was, thus, not available to accept another pulse.

7. FILE NAME: displays the disk file name (if any) of the spectrum being displayed.
8. VERTICAL SCALE: displays the vertical scaling factor of the spectrum box. Vertical scaling refers to how many counts can be represented in the vertical space of the spectrum box. A count of zero is always represented as a dot at the bottom of the spectrum box. If the vertical scale is set to 256, then a dot at the top of the spectrum box represents a count of 255. Counts that exceed this value wrap around to the bottom of the screen. The vertical scaling values are powers of 2: 256, 512, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 512K, 1M, 2M, 4M, 8M, and 16M, where K represents a value of 1024, and M represents a value of 1048576. Also available is a log scale that allows all count values to be represented in the box, with greater detail at lower counts. In this mode, the spectrum box is composed of 6 zones from top to bottom

(42 vertical pixels per zone), with each zone representing 16 times as many counts as the zone below it. In other words, the bottom zone on the screen represents counts from 0 to 15, the next zone up represents counts from 16 to 255, the next zone up represents counts from 256 to 4095, etc.

9. HORIZONTAL SCALE: displays 1, 2, 4, or 8, indicating how much horizontal stretching of the spectrum is being shown in the spectrum box. A horizontal scaling factor of 1 means one channel is displayed for each horizontal pixel in the spectrum box, a factor of 2 means one channel is displayed for every two horizontal pixels in the spectrum box, etc.
10. Three columns for each regions of interest:
  - a. BEG displays the first channel number in the region of interest.
  - b. END displays the last channel number in the region of interest.
  - c. SUM displays the sum of all counts in the region of interest.
11. Current region of interest indicator (>): indicates which region of interest is currently under the control of the F3 and F4 keys.

The operator interacts with the MCA entirely through the keyboard. The keys used are the arrow keys, the page up and page down keys, the home and end keys, the plus and minus keys, and the function keys. The details of the functions are:

1. F1: manually starts and stops count collection in the energy spectrum by taking the DMR™/PCA™ hardware in or out of acquire mode. The prompt displays what action will occur if the F1 key is pressed; for example, if the prompt is STOP, the spectrum is currently collecting

count data, and pressing F1 will stop the collection. Performing this function invalidates the spectrum for analysis purposes.

2. F2: zeros all channels in the spectrum, erasing the spectrum. This function does not stop the DMR™/PCA™ hardware from continuing to acquire count data. Performing this function invalidates the spectrum for analysis purposes.
3. F3: sets the cursor channel to be the first channel of the current ROI (indicated by >).
4. F4: sets the cursor channel to be the last channel of the current ROI (indicated by >).
5. F5: changes the current ROI from 1 to 2, from 2 to 3, from 3 to 1. The current ROI is indicated by the > next to the ROI display.
6. F6: changes the current ROI from 3 to 2, from 2 to 1, from 1 to 3.
7. F7: removes all ROIs.
8. F8: changes spectrum box from DOT to FILL mode. In DOT mode, the number of counts in a channel is indicated by a dot placed vertically on the screen. In FILL mode, the number of counts in a channel is indicated by a solid line of color extending from the bottom up to where the dot would be placed. The prompt displays which mode will start if the F8 key is pressed; for example, if the prompt is FILL, the spectrum is currently being displayed in DOT mode, and pressing F8 will switch to FILL mode.
9. Esc: exits the MCA and returns to the point in the program where the MCA was accessed.
10. Left and Right Arrow Keys: move the cursor left (down) and right (up) one channel in the spectrum box.
11. PgDn and PgUp: move the cursor left (down) and right (up) sixteen channels in the spectrum box.
12. Home: moves the cursor to the left side of the spectrum, the first channel in the spectrum (channel 0).
13. End: moves the cursor to the right side of the spectrum, the last channel in the spectrum (channel 511).

14. Up and Down Arrow Keys: decrease or increase the vertical scaling factor of the spectrum box.
15. Plus and Minus Keys: increase or decrease the horizontal scaling factor of the spectrum box.

### **2.2.3 Support Features**

The program has several other functions which support the two main functions of collecting and analyzing gamma spectra.

#### **2.2.3.1 Load/View/Print Gamma Spectra**

By choosing "View Saved Spectra" from the main menu, the operator can retrieve any Nucleus-compatible spectrum from disk. The operator chooses the appropriate "Load" option from the menu to load a spectrum from disk to replace any one of the three spectra stored in memory. The operator may then use the MCA screen to view the spectrum. (Any spectrum that is already in memory may also be viewed). Analysis cannot be performed on spectra that are retrieved from disk. The only spectra that can be analyzed are those that are collected while in the "Analyze" option of the program.

Additionally, a spectrum that is in memory can be printed to paper using either a Hewlett-Packard® LaserJet® II or III or a Texas Instrument™ model 855. The code that performs the printing is contained in the file VIEWSPEC.C, along with the MCA code. Examples of printed spectra are provided in Appendix F.

#### **2.2.3.2 Energy Calibration**

In order to ensure that the PCA™ hardware is storing gamma energy counts in the proper channels, the operator must perform an energy calibration. The procedure for this calibration is found in Appendix G. The Calibration Menu function aids in the energy calibration process by allowing the operator to start collecting a spectrum and set the MCA cursor and ROI for a particular

compound's energy spectrum with only one selection from the menu. The two compounds supported by this menu,  $^{137}\text{Cs}$  and  $^{40}\text{K}$ , are used for energy calibration because of the large characteristic peaks in their gamma spectra. On the other hand, a generic start option starts collecting a spectrum without setting the cursor or highlighting any region of interest.

From the Calibration menu, the operator can choose the "Smooth" option. The Smooth option averages counts over adjacent channels, thus eliminating the roughness caused by the scattering effect of the NaI(Tl) detectors. A smoothed gamma energy spectrum shows its peaks more distinctly, allowing the operator to more easily align those peaks to the proper channel.

#### 2.2.3.3 Reference Calibration

At the heart of the three-ROI method of estimating the concentration of radionuclides is the comparison of two ratios: the ratio of counts (measured nuclear decay events) in the sample to the amount of sample material and the ratio of counts to a known amount of the radionuclide. The three-ROI method uses three different radionuclides in known amounts (referred to as reference materials) and the counts from the same three ROIs in each of them to establish these count-to-amount ratios.

The reference calibration function is the process of collecting gamma spectra from each of the three reference materials, summing the counts from the ROIs, and storing these counts, along with the associated amounts of the radionuclides, in a file called the reference file. The reference file is an ASCII text file whose format is specified in Appendix D. Calibration is normally performed once per quarter unless changes in equipment occur. A complete description of the procedure may be found in Appendix G. A reference file must be loaded into the computer's memory (either by building a new reference file or by loading an old reference file from disk) before analysis of a sample can be performed.

To load an old reference file, the operator may choose to load the most recent reference file or to manually type in the name of a desired reference file. Loading a reference file into memory consists of opening the ASCII text file and

reading the numbers it contains into three matrices (represented in the program by three variable arrays): REF, CON, and BKG. Each of these matrices is actually an array of three matrices, one for each of the three inputs. REF is a  $3 \times 3$  matrix that contains the counts in each of the three regions of interest for each of the three reference materials. CON is a  $3 \times 3$  matrix that contains the amount (concentration of radionuclide  $\times$  weight of reference  $\times$  time) of each of the three radionuclides in each of the three reference standards. BKG is a  $3 \times 1$  matrix of background counts in each of the three ROIs. The program function that reads the numbers from the reference file into these arrays, "Load\_Ref", is located in the file ANALYZE.C.

If the operator chooses to create a new reference file, the program function "New\_Ref" is executed. New\_Ref controls the collecting and saving of the reference file. This function and its supporting functions are contained in the file NEWREF.C. The execution of this program function proceeds as follows:

1. Gamma spectra must be collected from each of the three inputs for background. These spectra are collected for a period ten times longer than the sample spectra to minimize the error inherent in counting random events such as radioactive decay. The following steps are executed for each background spectrum collected:
  - a) When the operator chooses to begin collecting a background spectrum, program function "Setup\_Ref" is executed. This function sets flags indicating that the background spectrum is being collected and that the three spectra for the reference materials (radium, potassium, thorium) for this input have not been collected.
  - b) When the spectrum has been collected (50 minutes "live" time), program function "Update\_Data" is executed. This function sums the counts in the three ROIs and stores the sums in the BKG matrix for that input. (These sums are divided by ten to match the sample collection counts.) Update\_Data also saves the background

spectrum to the A: drive in an appropriately named file. Spectrum file naming conventions are discussed in Appendix E.

2. After the background spectra for the inputs are collected, the spectra for the reference standards (radium, potassium, thorium) may be collected in any order. These spectra are also collected for a period ten times longer than sample spectra. The following steps are executed for each spectrum:
  - a) When the operator chooses to begin collecting a reference spectrum, program function "Setup\_Ref" is executed. This function sets a flag indicating that the particular reference spectrum is being collected and accepts the weight of the reference material from the user via the keyboard. Next the user enters the known concentration of each of the three radionuclides in the reference standards. (The concentrations in our reference standards come up as defaults in this function, so the operator can just press the Enter key.) The concentration  $\times$  reference material weight  $\times$  count time (sample time, not reference time) is entered into the appropriate row of the CON matrix for that input.
  - b) The spectrum is collected for 50 minutes "live" time, after which program function "Update\_Data" is executed. This function sums the counts in the three ROIs, divides the sums by ten to match the sample collection counts, subtracts the background counts for the three ROIs, and stores the sums in the appropriate row of the REF matrix for that input. Update\_Data then saves the reference spectrum to the A: drive in an appropriately named file.
3. At any time the operator may choose the "View" option from the new reference menu to bring up the MCA screen and view the collection of

a spectrum. The operator must take care not to alter the spectrum from the MCA screen since this will invalidate the spectrum, requiring it to be re-collected (and these are very long collection times).

4. After all background and reference spectra have been collected, thus loading the BKG, REF, and CON matrices with the proper values, the program is ready to save the reference file. When the operator chooses the "Save" option from the new reference menu, program function "Save\_Ref" is executed. Save\_Ref first checks to make sure all required spectra have been collected. It next executes program function "Get\_RfName", creating a unique name for the reference file (see Appendix E for file-naming conventions). This ASCII text file is opened in the reference directory, and the values in the REF, CON, and BKG matrices are written to it. See Appendix D for the exact format of this file. Finally, Save\_Ref saves the name of this reference file in the ASCII text file called LATEST in the reference subdirectory (see the section below called "Directories" for a description of the reference subdirectory). The reference file is now in memory, and sample analysis may be performed.

### **3. INSTALLATION PROCEDURES**

#### **3.1 HARDWARE**

Installation of the NaI(Tl) detectors, preamplifiers, and amplifiers is outside the scope of this memorandum. Installation of the unique parts of our system (namely the DMR™ system) is discussed.

### **3.1.1 Digital Multiplexer Router**

The DMR-II™ model 108 must be mounted in a NIM bin. The DMR™ has internal switches, set at the factory to the most common configuration, that determine the number of channels allocated to each of the inputs. Our system uses the default settings, but, if necessary, the left side panel of the DMR™ may be removed and the switches changed.

### **3.1.2 Personal Computer Analyzer**

The PCA-II™ card can be installed in any available expansion slot inside the computer. (See the computer manual for instructions on installing a computer card). If available, the left slot should be used for physical convenience.

The PCA-II™ card has a set of 10 switches, with switch selections printed on the PCA™ card for reference. The first three switches select the number of PCA-II™ cards installed in the computer. The last seven switches select the port address where the computer will access the PCA-II™ card. All switches are set at the factory to the most commonly used configuration. The computer port address used by the program is 1E0 hex, which is also the default setting.

There are several jumper wires on the PCA-II™ card, one of which selects the computer hardware interrupt lines to be used for timing purposes. Normally the IRQ3 line is used by the PCA-II™ card, which corresponds to the COM2 device. If a mouse device is installed on the computer using the COM2 device, a conflict occurs. In this case, the mouse should be installed on a different device; otherwise, the interrupt jumper on the PCA-II™ Card and the program's interrupt routines will need to be modified.

### **3.1.3 Digital Multiplexer Timer Card**

The DMR-II™ timer card needs to be installed in a computer expansion slot next to the PCA-II™ Card. The settings of the switches on the timer card must match exactly the address settings of the PCA-II™ switches. It should be noted

that the program does not take advantage of the timing capabilities of the DMR-II™ timer card but uses instead the timing interrupts from the PCA-II™ card to determine when to turn off the DMR™ inputs.

### 3.1.4 Hardware Adjustment

Once the DMR™ has been installed in the NIM bin, the cards installed in the computer, and the computer case closed, the system components may be connected. The outputs from the TC240 amplifiers are connected via BNC cables to the first three inputs on the front of the DMR™. The ribbon cable from the rear of the DMR™ is connected to the rear panel of the DMR™ timer control board in the computer. The signal output on the back of the DMR™ is connected via BNC cable to the signal input on the back of the PCA™ card.

Of the three trim-pots on the rear of the PCA™ card, two must be disabled when using the PCA™ card in conjunction with the DMR™ unit: the top one, called the upper level discriminator (ULD); and the middle one, called the lower level discriminator (LLD). The ULD trim-pot should be adjusted 20 turns clockwise, and the LLD should be adjusted 20 turns counterclockwise.

The third trim-pot on the rear of the PCA™ card is called the zero intercept (or zero crossover) and must be adjusted. In an MCA system, a linear relationship must exist between pulse height (in volts) and the channel to which the pulse is stored. For example, if a pulse of 8 volts is stored in channel 500 and a pulse of 4 volts is stored in channel 250, a pulse of 0 volts should be stored at channel 0. In reality, a pulse of zero volts is not a pulse at all and, therefore, cannot be detected. Even though this is the case, it is important that the linear relationship between pulse height and channel number match the zeroth channel to zero volts.

Using a precision pulse generator (pulser) connected to an input on the front of the DMR™ unit, the zero intercept can be determined and the trim-pot adjusted accordingly. If 8 volt pulses are being stored in channel 500 and 4 volt pulses are being stored in channel 200, it follows that a pulse of zero volts would be stored in channel -100 (the zero intercept). In this case, the zero intercept should be raised by turning the trim-pot clockwise. Conversely, if 8 volt pulses are being stored in

channel 500 and 4 volt pulses are being stored in channel 300, it follows that zero volt pulses would be stored in channel 100 (the zero intercept). In this case, the zero intercept should be lowered by turning the trim-pot counterclockwise.

### **3.2 SOFTWARE**

The following paragraphs describe the procedures necessary to install the program and supporting files on the fixed hard disk drive of the personal computer.

#### **3.2.1 Directories**

To keep them together for easier access and maintenance, the program and supporting files should be placed in a file subdirectory. Although the name of the directory is not important to the operation of the program, a suggested name is SOIL. It would appear as C:\SOIL (where C: is the hard disk drive name). This subdirectory is called the program directory henceforth.

Another subdirectory must be placed within the program subdirectory to contain the reference files. The name of this subdirectory is important. It must be named REFER. It appears as C:\SOIL\REFER (where C:\SOIL is the program subdirectory already established). This subdirectory is called the reference directory.

#### **3.2.2 Program and Files**

The program (named SOIL.EXE) must be copied into the program directory. A supporting file, EGAVGA.BGI, must also be copied into the program directory. This contains a run-time library from Borland International, Inc., distributor of the C language compiler used to develop the SOIL.EXE program. The file extension BGI stands for Borland Graphics Interface; the library contains graphic routines needed to display the MCA screen.

The two data files, TEMPDB.DAT and TEMPDB.BAD, that contain the analysis results are placed into the program directory by the SOIL.EXE program when needed. The operator does not need to create them.

The next file that needs to be placed in the program directory is LATEST (no file extension). It is an ASCII text file that contains three letters used in naming the spectra files. Once this file is created, the SOIL.EXE program will maintain it. It may be created with a text editor or a word processor that can save files as ASCII text. To do so, create a file that contains AAA, with no leading, trailing, or intervening spaces. To create the file without the use of a text editor or word processor, type the following at the DOS prompt:

```
COPY CON LATEST <Return>
AAA <Ctrl-Z> <Return>
```

where <Ctrl-Z> is accomplished by holding down the control key and pressing Z. The following message should appear:

1 File(s) copied

To verify that the LATEST file has been properly created, type the following command at the DOS prompt:

```
TYPE LATEST<Enter>
```

The following should appear:

AAA

If not, start the process again from COPY CON LATEST.

The last file that needs to be put into the program directory is ERFILE (no file extension). Also an ASCII text file, it can be created by the same methods used to create LATEST. This file is used to calculate the error estimation for the

analysis results. The format of this file is detailed in Appendix D.

None of the files in the reference directory needs to be created by the operator. The SOIL.EXE program maintains the LATEST file and creates the reference files found there.

**REFERENCES**

- Boas, Mary L. 1983. *Mathematical Methods in the Physical Sciences*. 2nd edition, John Wiley & Sons, New York.
- Doane, R. W., B. A. Berven, and M. S. Blair. 1984. A Computer-Controlled System for Rapid Soil Analysis of  $^{226}\text{Ra}$ . In *Proceedings of the Health Physics Society: Computer Applications in Health Physics*, Pasco, Washington, Feb. 5-9, 1984, 173-178.
- Rood, Arthur S. 1986. *Operation Procedures for the KRTCOM (Potassium, Radium, and Thorium) Gamma Analysis Program*. Oak Ridge National Laboratory, Oak Ridge, Tenn. Typescript.
- Shields, Paul C. 1980. *Elementary Linear Algebra*. 3rd edition, Worth Publishers, New York.

**ACRONYMS AND INITIALISMS**

<b>ADC</b>	Analog to digital converter
<b>DMR™</b>	Digital Multiplexer Router
<b>EGA</b>	enhanced graphics adapter
<b>LLD</b>	lower level discriminator
<b>MCA</b>	multichannel analyzer
<b>MCS</b>	multichannel scaling
<b>NaI(Tl)</b>	thallium doped sodium iodide
<b>ORNL-PAG</b>	Oak Ridge National Laboratory Pollutant Assessments Group
<b>PCA™</b>	Personal Computer Analyzer
<b>PHA</b>	pulse height analysis
<b>ROI</b>	region of interest
<b>ULD</b>	upper level discriminator
<b>VGA</b>	virtual graphics array

**APPENDIX A**  
**PROGRAM SOURCE CODE**



**PROGRAM SOURCE CODE**

**Instructions for using this appendix**

The first page is an index of all the user defined functions in the program. The source code file name is listed for each function.

Next are listed all the C language header files. These have a .H extension and are listed in alphabetic order.

Next are listed all the C language and assembly language source code files. These have .C and .ASM extensions and are listed in alphabetic order.

All C language programs begin by executing the function called "main". Program execution can be traced from function to function from that point.

## Index of all user-defined functions in soil analysis program.

Function Name	File Name	Function Name	File Name
Acquire_Off.....	PCA .C	Matrix_Adj.....	MATRIX .C
Acquire_On.....	PCA .C	Matrix_Cof.....	MATRIX .C
Analyze.....	ANALYZE .C	Matrix_Det.....	MATRIX .C
Blank.....	VIEWSPEC.C	Matrix_Inv.....	MATRIX .C
Calc_Conc.....	ANALYZE .C	Matrix_Mul.....	MATRIX .C
Calc_DMR.....	PCA .C	Matrix_Scal.....	MATRIX .C
Calibrate.....	CALIB .C	Matrix_Trans.....	MATRIX .C
ChArrDiff.....	PCA .C	Menu.....	MENU .C
Clear_Spec_Box.....	VIEWSPEC.C	main.....	MAIN .C
Clr_Input.....	SOILASM .ASM	New_Ref.....	NEWREF .C
Clr_ROI.....	VIEWSPEC.C	Old_Ref.....	ANALYZE .C
Cursor.....	GETSTR .C	PCA_Interrupt_Off...PCA	.C
Draw_Acquire.....	VIEWSPEC.C	PCA_Interrupt_On...PCA	.C
Draw_Channel.....	VIEWSPEC.C	PCAOut.....	PCA .C
Draw_Counts.....	VIEWSPEC.C	Prn_Spec_LJ.....	VIEWSPEC.C
Draw_Cur_ROI.....	VIEWSPEC.C	Prn_Spec_TI.....	VIEWSPEC.C
Draw_Cursor.....	VIEWSPEC.C	Read_Input.....	SOILASM .ASM
Draw_FName.....	VIEWSPEC.C	Ref_ROIs.....	NEWREF .C
Draw_HScale.....	VIEWSPEC.C	Sample.....	ANALYZE .C
Draw_MCA_Screen.....	VIEWSPEC.C	Save_Ref.....	NEWREF .C
Draw_Menu_Box.....	MENU .C	SetCursor.....	GETSTR .C
Draw_Plot_Mode.....	VIEWSPEC.C	Setup_Ref.....	NEWREF .C
Draw_ROI_Beg.....	VIEWSPEC.C	Setup_Samp.....	ANALYZE .C
Draw_ROI_End.....	VIEWSPEC.C	Show_DskErr.....	NEWREF .C
Draw_ROI_Sum.....	VIEWSPEC.C	Show_Err.....	ANALYZE .C
Draw_Spec.....	VIEWSPEC.C	Signif.....	ANALYZE .C
Draw_Time.....	VIEWSPEC.C	Smooth.....	CALIB .C
Draw_VScale.....	VIEWSPEC.C	SpLoad.....	PCA .C
Fill_LogTable.....	VIEWSPEC.C	SpSave.....	PCA .C
Get_LJ_Plot.....	VIEWSPEC.C	Start_PCA.....	PCA .C
Get_Plot_Y.....	SOILASM .ASM	Stop_PCA.....	PCA .C
Get_RfName.....	NEWREF .C	SubMatrix.....	MATRIX .C
GetESTr.....	GETSTR .C	Toggle_PCA_3.....	PCA .C
GetKey_If_Ready....	MENU .C	UnLight.....	MENU .C
GetVideoType.....	GETSTR .C	Update_Data.....	NEWREF .C
Handler.....	PCA .C	Update_Samp.....	ANALYZE .C
HighLight.....	MENU .C	Valid_Clr.....	PCA .C
Init_Plot.....	VIEWSPEC.C	Valid_Read.....	PCA .C
Init_ROI.....	VIEWSPEC.C	Valid_Write.....	PCA .C
LJ_Line.....	VIEWSPEC.C	View_Saved.....	VIEWSAVD.C
Load_Ref.....	ANALYZE .C	View_Spectrum.....	VIEWSPEC.C
Main_Menu.....	MAINMENU.C	Write_Input.....	SOILASM .ASM
Make_Prn_Num.....	VIEWSPEC.C		

```

/* DMR.H    Structure definitions provided by Nucleus Inc. for making */
/* compatible spectrum files.                                         */
/* Last Modified: 08/01/91                                         */

***** CALIB STRUCTURE *****

typedef struct {

    char  units[4];          /* Units Strings (ie: Kev) - ASCIIZ String      */
    double calx0;            /* Coefficient for 0th Order Term                */
    double calx1;            /* Coefficient for 1st Order Term                */
    double calx2;            /* Coefficient for 2nd Order Term                */
    int   numpoints;          /* Number of calibration points                */
    double calctrd[5];        /* Centroids for Up to 5 Points                 */
    double calvalue[5];        /* Values for Up to 5 Calibration Points       */

} CALIB;

***** ACQUIRE STRUCTURE *****

typedef struct {

    long  preset;             /* ( 0- 3) Preset Time in Seconds           */
    long  elapsed;            /* ( 4- 7) Elapsed Time in Seconds          */
    long  remain;             /* ( 8-11) Remaining Time in Seconds         */
    char  inacquire;          /* (12) 0H if in acquire else 00h           */
    char  armed;              /* (13) 0H if armed else 00h                */
    char  starttime[13];       /* (14-26) Acquire Start Time - hh:mm:ss PM */
    char  startdate[13];       /* (27-39) Acquire Start Date - mmm dd yyyy */
    char  stoptime[13];        /* (40-52) Acquire Start Time - hh:mm:ss PM */
    char  stopdate[13];        /* (53-65) Acquire Start Date - mmm dd yyyy */

} ACQUIRE;

***** DMRHEADER STRUCTURE *****

typedef struct {
    /* Byte Offset          Description      */
    /*-----*/
    char  etime[3];           /* ( 0- 2) Elapsed Time - 3 Byte BCD number */
    char  ltimeflag;          /* ( 3) Live Time Flag                      */
    char  rtimeflag;          /* ( 4) Real Time Flag                      */
    char  convergain;         /* ( 5) Conversion Gain Number               */
    char  digoffset;          /* ( 6) Digital Offset Number                */
    char  idcodestr[15];      /* ( 7- 21) Filename - 15 Byte ASCIIZ       */
    char  pca_date[12];        /* (22- 33) Date of File Save - mmm dd yyyy */
    char  pca_time[12];        /* (34- 45) Time of File Save - hh:mm:ss PM */
    char  group;               /* ( 46) Group Number                       */
    char  units;               /* ( 47) System Calibration Flag            */
    CALIB  calib;              /* (48-157) Energy Calibration Structure   */
    char  fill[99];             /* (158-256) For Compatibility With PCA-I  */
    char  phemode;              /* (257) PHA Mode Flag                      */
    char  mcs;                  /* (258) MCS Mode Flag                      */
    char  mcstimelab;          /* (259) MCS Time Label Flag                */
    char  mcsdwellnumb;         /* (260) MCS Dwell Number                   */
    char  mcspasct[3];          /* (261-263) MCS Pass Count - 3 Byte BCD Number */
    char  centupflag;           /* (264) Centroid Update Flag                */
}

```

```
char fwhmupflag; /* (265 ) FWHM Update Flag */
int numchanspm; /* (266-267) Number of Channels in Spectrum */
char asday; /* (268 ) Acquisition Start Day */
char asmonth; /* (269 ) Acquisition Start Month */
int asyear; /* (270-271) Acquisition Start Year */
char ashund; /* (272 ) Acquisition Start Hundredth */
char assec; /* (273 ) Acquisition Start Second */
char asmin; /* (274 ) Acquisition Start Minute */
char ashour; /* (275 ) Acquisition Start Hour */
char astpday; /* (276 ) Acquisition Stop Day */
char astpmouth; /* (277 ) Acquisition Stop Month */
int astpyear; /* (278-279) Acquisition Stop Year */
char astphund; /* (280 ) Acquisition Stop Hundredth */
char astpsec; /* (281 ) Acquisition Stop Second */
char astpmin; /* (282 ) Acquisition Stop Minute */
char astphour; /* (283 ) Acquisition Stop Hour */
char id[72]; /* (284-355) Identification - ASCIIZ String */
char majvers; /* (356 ) Program Major Version Number */
char minvers; /* (357 ) Program Minor Version Number */
long real_elap_time; /* (358-361) Real Elapsed Time in Seconds */
char acqstop_time[13]; /* (362-374) Acquire Stop Time - hh:mm:ss PM */
char acqstop_date[13]; /* (375-387) Acquire Stop Date - mmm dd yyyy */
char acqstart_time[13]; /* (388-399) Acquire Start Time - hh:mm:ss PM */
char acqstart_date[13]; /* (400-412) Acquire Start Date - mmm dd yyyy */
char future[96]; /* (413-509) For Future Expansion */
int endheader; /* (510-511) Version Number * 100 */
```

```
) DMRHEADER;
```

```
/* PCA.H  Definitions needed for interface with PCA Card */
/*          and prototypes of PCA interface functions.      */
/*          Last Modified: 08/01/91                         */

#define PCA_INT          0x0b
#define ICC_CONTROL      0x20
#define ICC_MASK          0x21
#define ICC_ACKNOWLEDGE 0x20

#define PCA_INT_ON        0xf7
#define PCA_INT_OFF       0x08

#define PCAPORT          0x1e0
#define NEWPORT           0x1ff

void interrupt Handler(void);
void Acquire_Off(int);
void Acquire_On(int);
unsigned char Calc_DMR(unsigned char);
void Start_PCA();
void Stop_PCA();
void Toggle_PCA_3(char);
void PCAOut(int,char);
void Valid_Read(int, {long *});
void Valid_Write(int, {long *});
void Valid_Clr(int);
```

```
/* SCANCODE.H      Scancodes of keys returned by BIOS Keyboard Interrupt */
/*          and prototype of function that reads the keyboard.      */
/*          Last Modified: 08/01/91                                     */

char GetKey_If_Ready();

#define ESC      1
#define ENTER    28
#define KEY_UP   72
#define KEY_LEFT 75
#define KEY_RIGHT 77
#define KEY_DOWN 80
#define F1       59
#define F2       60
#define F3       61
#define F4       62
#define F5       63
#define F6       64
#define F7       65
#define F8       66
#define F9       67
#define F10      68
#define PGUP     73
#define PGDN     81
#define HOME     71
#define END      79
#define PLUS     13
#define MINUS    12
#define PLUS_KP  78
#define MINUS_KP 74
```

```
/* SOIL.H General definitions for soil analysis program. */
/* Last Modified: 08/01/91 */

#define NUM_INPUTS      3
#define CHANS_PER_INPUT 512
#define BYTES_PER_INPUT 512 * 4

#define ACQ_TIME        5.0
#define ACQ_TICKS       5L * 60L * 100L

#define MAX_ROI         3

#define NOT_COLLECTED  0
#define COLLECTING     1
#define COLLECTED      2

#define ROI_BEG_1       273
#define ROI_END_1       412
#define ROI_BEG_2       226
#define ROI_END_2       259
#define ROI_BEG_3       426
#define ROI_END_3       511
```

---

```
/* SOILASM.H      Prototypes of assembly language functions. */
/*           Last Modified: 08/01/91 */
```

```
int Get_Plot_Y(int, int);
void Read_Input(int, long far *);
void Write_Input(int, long far *);
void Clr_Input(int);
```

```
*****
*          ANALYZE.C
*
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains Analyze() which is the main menu for analysis. From
* this menu Old_Ref(), New_Ref(), and Sample() menus are called. This module
* contains Old_Ref() which allows an existing reference file to be loaded,
* and Sample() which allows for sample analysis. Several other functions
* are included here which are used by Old_Ref() and Sample().
*****
```

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <time.h>
#include <string.h>

#include <soil.h>
#include <pca.h>

*****
*          GLOBAL DATA REFERENCES
*
*****
```

```
extern long *counts;           /* Buffer for Count Data */
extern long timer[NUM_INPUTS]; /* Live Timers */
extern char acquire[NUM_INPUTS]; /* Acquire Status */
extern char invalid[NUM_INPUTS]; /* Data Altered Status */
extern double bkg[NUM_INPUTS][MAX_ROI]; /* Background Matrices */
extern double con[NUM_INPUTS][MAX_ROI][MAX_ROI]; /* Concentration Matrices */
extern double ref[NUM_INPUTS][MAX_ROI][MAX_ROI]; /* Reference Count Mats */
extern char *spect_fname[NUM_INPUTS]; /* Spectra Names */
extern int insert;             /* Insert/OverType Mode for GetEstr */
extern char ref_name[14];      /* Current Reference File Name */

*****
*          LOCAL DEFINITIONS
*
*****
```

```
#define ANMENU 4
#define ANBOX_W 25
#define ANBOX_H 3
#define ANCOL_COORD 29
#define LRMENU 3
```

```
#define LRBOX_W 20
#define LRBOX_H 3

#define LRCOL_COORD 31

#define SMENU_COL 3
#define SMENU_ROW 3

#define SBOX_W 15
#define SBOX_H 3

#define K 2.576

/*
 *          LOCAL DATA DEFINITIONS
 *
 *****/
int samp_status[NUM_INPUTS];      /* Sample Data Status for each input */
double samp[NUM_INPUTS][MAX_ROI]; /* Sample Count Matrices */

double samp_wt[NUM_INPUTS];        /* Sample Weights */
int samp_dy[NUM_INPUTS];          /* Sample Days in Can */
int samp_br[NUM_INPUTS];          /* Sample Barrel Number */
char *samp_no[NUM_INPUTS] =       /* Sample Numbers */
{
    "          ",
    "          ",
    "          ";
};

char *weight_str = "    ";
char *day_str = "    ";
char *barrel_str = "    ";

double ingrowth[19] = {1.00, 0.79, 0.83, 0.89,
                      0.89, 0.90, 0.92, 0.93,
                      0.94, 0.95, 0.96, 0.97,
                      0.98, 0.98, 0.98, 0.98,
                      0.98, 0.98, 1.00};

/*
 *          Analyze()
 *
 * Main menu for analysis. Calls Old_Ref(), New_Ref(), and Sample().
 *
 *****/
Analyze()
{
    int choice = 0;
```

```
int row_crd[ANMENU] = {6, 9, 12, 15};
char *anmenu_text[ANMENU] = {"Load Reference File",
                            "Make New Reference",
                            "Analyze Sample",
                            "Quit"};
int i;

while (choice != 3 && choice != -1) {
    clrscr();

    gotoxy(29,2);
    puts("A N A L Y S I S   M E N U");

    gotoxy(40,24);
    printf("Current reference file: %s",ref_name);

    for (i=0; i<ANMENU; i++)
        Draw_Menu_Box(ANCOL_COORD, row_crd[i], ANBOX_W, ANBOX_H, anmenu_text[i]);

    choice = Menu(choice, ANMENU, 1, ANCOL_COORD, row_crd[0], ANBOX_W, ANBOX_H, 0, 0);

    switch (choice) {
        case 0:
            Old_Ref();
            break;

        case 1:
            New_Ref();
            break;

        case 2:
            Sample();
            break;
    }
}

return 0;
}

*****
*          Load_Ref()          *
* Physically loads a reference file into matrices: REF, COW, & BKG.      *
*****
```

Load\_Ref(fname)

```
char *fname; /* Name of Reference File */

{
    FILE *fh;
```

```
int i,j,k;
int errchk;

fh = fopen(fname,"rt");
if (fh == NULL) {
    Show_DskErr(fname); /* Couldn't open file */
    return 1;
} else {
    for (i=0; i < NUM_INPUTS; i++) /* Load REF matrices */
        for (j=0; j < MAX_ROI; j++)
            for (k=0; k < MAX_ROI; k++) {
                errchk = fscanf(fh,"%lf\n",&ref[i][j][k]);
                if (errchk != 1) {
                    fclose(fh);
                    Show_DskErr(fname);
                    ref_name[0] = 0;
                    return 1;
                }
            }
    for (i=0; i < NUM_INPUTS; i++) /* Load CON matrices */
        for (j=0; j < MAX_ROI; j++)
            for (k=0; k < MAX_ROI; k++) {
                errchk = fscanf(fh,"%lf",&con[i][j][k]);
                if (errchk != 1) {
                    fclose(fh);
                    Show_DskErr(fname);
                    ref_name[0] = 0;
                    return 1;
                }
            }
    for (i=0; i < NUM_INPUTS; i++) /* Load BKG matrices */
        for (j=0; j < MAX_ROI; j++) {
            errchk = fscanf(fh,"%lf",&bkg[i][j]);
            if (errchk != 1) {
                fclose(fh);
                Show_DskErr(fname);
                ref_name[0] = 0;
                return 1;
            }
        }
    fclose(fh);
    strcpy(ref_name, &fname[6]); /* Set new ref_name */
    return 0;
}
```

```
*****
*          Old_Ref()
*
* Menu for loading in a reference file that was previously saved.
*
*****
```

---

```
Old_Ref()
{
    FILE *fh;
    struct date d;
    char fname[80];
    int choice = 0;
    int errchk;
    int row_crd[LRMENU] = {6, 9, 12};
    char *lrmenu_text[LRMENU] = {"Use Most Recent",
                                 "Type In Ref Name",
                                 "Quit"};
    int i;

    while (choice != 2 && choice != -1) {
        clrscr();
        gotoxy(23,2);
        puts("L O A D   R E F E R E N C E   M E N U");
        gotoxy(40,24);
        printf("Current reference file: %s", ref_name);

        for (i=0; i<LRMENU; i++)
            Draw_Menu_Box(LRCOL_COORD, row_crd[i], LRBOX_W, LRBOX_H, lrmenu_text[i]);
        choice = Menu(choice, LRMENU, 1, LRCOL_COORD, row_crd[0], LRBOX_W, LRBOX_H, 0, 0);
        switch (choice) {
            case 0:           /* Load most current reference file */
                fh = fopen("REFER\\LATEST","rt");
                if (fh == NULL) {
                    Show_DskErr("REFER\\LATEST");
                } else {
                    errchk = fscanf(fh,"%18s",fname); /* Get Latest Name */
                    fclose(fh);
                }
            }
    }
}
```

```
if (errchk == EOF) {
    Show_DskErr("REFER\\LATEST");
} else {
    Load_Ref(fname);           /* Load It */
}
break;

case 1:          /* Manually enter file name */
strcpy(fname, "REFER\\");
strcpy(&fname[6], ref_name); /* Add current reference name */
for (i=6; i < 17; i++) /* Pad with Spaces */
    if (fname[i] == '\0')
        for (; i < 17; i++)
            fname[i] = ' ';
    fname[17] = '\0';        /* End of String Marker */

gotoxy(8,18);
puts("(The file must be in subdirectory REFER)");
gotoxy(8,16);
printf("Input reference file name: ");
GetEStr(&fname[6], wherey(), wherex(), &insert, 1, 1);

Load_Ref(fname);
break;
}
return 0;
}

*****
*          *
*      Setup_Samp()          *
*          *
* Get a sample ready to collect. Also check for conflicts.          *
*          *
*****
```

Setup\_Samp(input)

```
int input; /* DMR Input: 0,1,2 */

{
    int i;
    int yn;

    ***** See if this Input is Currently Collecting *****
    i = input+1;      /* Input # for printint */
```

```

if (samp_status[input] == COLLECTING) {
    gotoxy(8, 21);
    printf("Input %d is currently collecting. Re-Start anyway? No Yes", i);
    yn = Menu(0, 1, 2, 61, 21, 3, 1, 3, 0);
    if (yn == 1) {
        samp_status[input] = NOT_COLLECTED;
    } else {
        return 1; /* Abort */
    }
}

***** See if Sample has already been collected for this Input *****/
if (samp_status[input] == COLLECTED) {
    gotoxy(8, 21);
    printf("Overwrite the data in Input %d? No Yes", i);
    yn = Menu(0, 1, 2, 41, 21, 3, 1, 3, 0);
    if (yn != 1)
        return 1; /* Abort */
}

***** Input the Sample Weight, Days in can, and ID # *****/
yn=0;
while (yn == 0) {
    clrscr(); /* Get User Inputs */

    gotoxy(1, 8);
    printf("Input %d sample weight: ", i);
    GetEStr(weight_str, wherey(), wherex(), &insert, 0, 0);
    samp_wt[input] = atof(weight_str);

    gotoxy(1, 10);
    printf("Input %d days in can: ", i);
    GetEStr(day_str, wherey(), wherex(), &insert, 0, 0);
    samp_dy[input] = atoi(day_str);

    gotoxy(1, 12);
    printf("Input %d sample number: ", i);
    GetEStr(samp_no[input], wherey(), wherex(), &insert, 1, 1);

    gotoxy(1, 14);
    printf("Input %d barrel number: ", i);
    GetEStr(barrel_str, wherey(), wherex(), &insert, 0, 0);
    samp_br[input] = atoi(barrel_str);

    clrscr(); /* Re-Print */

    gotoxy(1, 8);
    printf("Input %d sample weight: %lg", i, samp_wt[input]);
}

```

```

gotoxy(1, 10);
printf("Input %d days in can: %d", i, samp_dy[input]);
gotoxy(1, 12);
printf("Input %d sample number: %s", i, samp_no[input]);
gotoxy(1, 14);
printf("Input %d barrel number: %d", i, samp_br[input]);

gotoxy(25, 21); /* Confirm */
puts("Change Okay Cancel");
yn = Menu(1, 1, 3, 25, 21, 6, 1, 3, 0);
}

if (yn == 2 || yn == -1)
    return 1; /* Abort */

return 0; /* Okay */
}

*****
*          Update_Samp()
*
* If any inputs are finished collecting then get data and update the flags.
*****
Update_Samp()
{
    int i,j;

    for (i=0; i < NUM_INPUTS; i++) {
        if (acquire[i] == 0 && samp_status[i] == COLLECTING) {
            samp_status[i] = COLLECTED;           /* Update Data Status */
            Valid_Read(i, counts);             /* Get Counts from PCA Card */

            samp[i][0] = 0.0;                  /* Sum ROI 1 */
            for (j=ROI_BEG_1; j <= ROI_END_1; j++)
                samp[i][0] += counts[j];

            samp[i][1] = 0.0;                  /* Sum ROI 2 */
            for (j=ROI_BEG_2; j <= ROI_END_2; j++)
                samp[i][1] += counts[j];

            samp[i][2] = 0.0;                  /* Sum ROI 3 */
            for (j=ROI_BEG_3; j <= ROI_END_3; j++)
                samp[i][2] += counts[j];

            for (j=0; j < MAX_ROI; j++)      /* Minus Background */
                samp[i][j] -= bkg[i][j];
        }
    }
}

```

```

        }
    }

    return 0;
}

/*
*          Signif()
*
* Rounds off a number to the nearest significant digit.
*
***** */

double Signif(n, sd)
double n; /* Number */
int   sd; /* Significant Digits */

{
    double dv,ans,lg;
    long int rnd;

    /* Check for Errors */
    if (sd < 1 || sd > 9 || n <= 0.0L)
        return 0.0;

    /* Save of copy of log base 10 of number */
    lg = log10(n);

    /* Find divisor that will put significant digits just left of decimal point */
    if (lg < 0.0L) {
        dv = pow10((int)lg - sd);
    } else {
        dv = pow10((int)lg + 1 - sd);
    }

    /* Round off to nearest significant digit */
    rnd = (long int)(n / dv + 0.5000000001L);
    /* Shift back to proper decimal alignment */
    ans = (double)rnd * dv + 0.0000000001L;
    return ans;
}

```

```
*****
*          Show_Err()          *
* Displays a message string (usually an error) and waits for user response. *
*****
Show_Err(str)
char *str; /* Message string */

{
    int x;

    x = 40 - (strlen(str) / 2);
    gotoxy(x, 21);

    puts(str);
    gotoxy(39, 22);
    puts("Okay");

    Menu(0, 1, 1, 39, 22, 4, 1, 0, 0);

    return 0;
}

*****
*          Calc_Conc()          *
* Calculate the sample concentrations and save the results.                 *
*****
Calc_Conc(input)

int input; /* DMR Input: 0,1,2 */

{
    FILE *prn;           /* Printer Stream */
    FILE *dat;           /* Data File Stream */
    FILE *ext;           /* Latest Spectrum Extension */
    FILE *erfile;        /* File Containing Error Estimates */

    char spect_ext[4];   /* Spectrum File Extension */
    char spect_next[4];  /* New Spectrum File Extension */
    char spect_name[16]; /* Spectrum File Name */

    char *drive = "A:";  /* Drive for Spectrum File */

    double inv[MAX_ROI][MAX_ROI]; /* Inverse of matrix REF */
    double proinv[MAX_ROI][MAX_ROI]; /* Inverse of proportionality constant */
}
```

```
double result[MAX_ROI];           /* Sample concentration */
double errest[MAX_ROI];          /* Estimate of Error in Concentration */

double lc[MAX_ROI];              /* Decision Limit */
double ld[MAX_ROI];              /* Detection Limit */
double mda[MAX_ROI];             /* Minimal Detectable Activity */

char flag[MAX_ROI];              /* MDA Flags (for printing) */

int days;                        /* Temp days in can */

int i;                            /* Temp Integer */
int errchk;                      /* Error Checking Temp */
int dfile;                        /* Save Data to Which Data File */

int erpig,ercomp,found;           /* Temp Variables for */
double erlo,erhi,erpct;          /* Finding Error Estimates */

char ertype;                     /* Error Type */

int sigdig;                      /* Significant Digits for Rounding */

struct date d;                  /* Date Structure */
time_t t;                         /* Time Structure */
char *timeptr;                   /* Full Time String from ctime() */
char timestr[6];                 /* Time String dd:dd:dd */

char *label[3] = {"Radium",
                  "Potassium",
                  "Thorium"};

char message[40];                /* Spectrum Name Message */

***** Get the inverse of matrix REF *****
errchk = Matrix_Inv(&ref[input], inv, MAX_ROI);
if (errchk) {
    Show_Err("MEMORY ERROR");
    return 1;
}

***** Get the inverse of proportionality constant matrix (PROINV) *****
errchk = Matrix_Mul(&con[input], inv, proinv, MAX_ROI, MAX_ROI, MAX_ROI);
if (errchk) {
    Show_Err("MEMORY ERROR");
    return 1;
}

***** Get the sample concentrations *****
errchk = Matrix_Mul(proinv, &samp[input], result, MAX_ROI, 1, MAX_ROI);
if (errchk) {
    Show_Err("MEMORY ERROR");
    return 1;
}
```

```

>

***** Multiply by the sample weight and the sample time *****/
Matrix_Scal(result, result, MAX_ROI, 1, (1.0 / (samp_wt[input] * ACQ_TIME)));

***** Zero out negative concentrations *****/
for (i=0; i < MAX_ROI; i++)
    if (result[i] < 0.0)
        result[i] = 0.0;

***** Adjust for Radon ingrowth *****/
days = samp_dy[input];
if (days < 0)                      /* Clip the Days */
    days = 0;
if (days > 18)
    days = 18;
Matrix_Scal(result, result, MAX_ROI, 1, (1.0 / ingrowth[days]));

***** Calculate the Decision Limit (lc) *****/
for (i=0; i < MAX_ROI; i++)
    lc[i] = K * (sqrt(bkg[input][i] + sqrt(bkg[input][i])));

***** Calculate the Detection Limit (ld) *****/
for (i=0; i < MAX_ROI; i++)
    ld[i] = (K * K) + (lc[i] * 2.0);

***** Get the Minimal Detectable Activity *****/
errchk = Matrix_Mul(proinv, ld, mda, MAX_ROI, 1, MAX_ROI);
if (errchk) {
    Show_Err("MEMORY ERROR");
    return 1;
}

Matrix_Scal(mda, mda, MAX_ROI, 1, (1.0 / (samp_wt[input] * ACQ_TIME)));
for (i=0; i < MAX_ROI; i++)
    if (result[i] < mda[i])
        flag[i]='*';
    else
        flag[i]=' ';

```

```

***** Calculate the Error Estimate *****/
erfile = fopen("ERFILE","rt");
if (erfile == NULL) {
    Show_Err("CANNOT OPEN ERFILE");
    return 1;
} else {
    for (i=0; i < MAX_ROI; i++) {
        errchk = fseek(erfile, 0L, SEEK_SET); /* Beginning of File */
        if (errchk) {
            Show_Err("SEEK ERROR ON ERFILE");
            fclose(erfile);
            return 1;
        }
        found = 0;
        while (found == 0) {
            errchk = fscanf(erfile, "%c %d %d %lf %lf %lf",
                            &ertype, &erpig, &ercomp, &erlo, &erhi, &erpct);
            if (errchk == EOF) {
                Show_Err("ENTRY NOT FOUND IN ERFILE");
                fclose(erfile);
                return 1;
            }
            if (input == erpig && i == ercomp && result[i] >= erlo && result[i] <= erhi) {
                if (ertype == 'P') {
                    errest[i] = result[i] * erpct; /* Percentage */
                } else {
                    errest[i] = erpct;           /* Constant */
                }
                found++;
            }
        }
    }
    fclose(erfile);
}

***** Round Off to 3 (or fewer) significant digits *****/
for (i=0; i < MAX_ROI; i++) {
    sigdig = 3;
    if (result[i] < 0.1)
        sigdig = 2;
}

```

```
if (result[i] < 0.01)
    sigdig = 1;
if (result[i] < 0.001)
    sigdig = 0;
result[i] = Signif(result[i], sigdig);

sigdig = 3;
if (errest[i] < 0.1)
    sigdig = 2;
if (errest[i] < 0.01)
    sigdig = 1;
if (errest[i] < 0.001)
    sigdig = 0;
errest[i] = Signif(errest[i], sigdig);
}

***** Print Calculated Concentrations and MDA Flags to Screen *****/
clrscr();
for (i=0; i < MAX_ROI; i++)
    printf("%-20s: %5g%+/- %5g\n", label[i], result[i], flag[i], errest[i]);
printf("\n* Minimum Detectable Activity\n");

***** Confirm the Saving of the Data *****/
gotoxy(27,16);
puts("Save Data? Yes No");
i = Menu(0, 1, 2, 40, 16, 3, 1, 3, 0);
if (i)          /* Exit */
    return 1;

***** Choose Which Data File *****/
gotoxy(27,18);
puts("Which Database? Good Junk");
dfile = Menu(0, 1, 2, 45, 18, 4, 1, 3, 0);

***** Determine Name of Spectrum File *****/
ext = fopen("LATEST","rt+");
if (ext == NULL) {
```

```
Show_Err("CANNOT OPEN FILE: LATEST");
return 1;

} else {
    errchk = fscanf(ext, "%3s", &spect_ext);      /* Get Latest Extension */
    if (errchk == EOF) {
        fclose(ext);
        Show_Err("READ ERROR ON FILE: LATEST");
        return 1;
    }

    errchk = fseek(ext, 0L, SEEK_SET);           /* Move to Start of File */
    if (errchk) {
        fclose(ext);
        Show_Err("SEEK ERROR ON FILE: LATEST");
        return 1;
    }

    ***** Increment Extension for Next Time *****/
    strcpy(spect_next, spect_ext);

    spect_next[2]++;
    if (spect_next[2] == 'Z'+1) {
        spect_next[2] = 'A';
        spect_next[1]++;
        if (spect_next[1] == 'Z'+1) {
            spect_next[1] = 'A';
            spect_next[0]++;
            if (spect_next[0] == 'Z'+1) {
                spect_next[0] = 'A';
            }
        }
    }
}

errchk = fprintf(ext,"%3s",spect_next);   /* Write out New Extension */
fclose(ext);

if (errchk == EOF) {
    Show_Err("WRITE ERROR ON FILE: LATEST");
    return 1;
}

***** Build the Spectrum File Name *****/
strcpy(spect_name, drive);                  /* A: */
strcat(spect_name, ref_name);              /* Reference Name */
spect_name[9] = (char)(input + '1');        /* Input Number */
spect_name[10] = '.';                      /* . (for extension)
```

```

spect_name[11] = 0;
strcat(spect_name,spect_ext);           /* Extension */

}

***** Print Calculated Concentrations and MDA Flags to Printer *****
prn = fopen("PRN","wt");
if (prn == NULL) {
    Show_Err("PRINTER ERROR");
    return 1;
} else {
    fprintf(prn,"%-11s %-12s %5g %2d %5g%c +/- %5g %5g%c +/- %5g %5g%c +/- %5g\n",
            samp_no[input], &spect_name[2], samp_wt[input], samp_dy[input],
            result[0], flag[0], errest[0],
            result[1], flag[1], errest[1],
            result[2], flag[2], errest[2]);
    fclose(prn);
}

***** Save the Spectrum File *****
errchk = SpSave(spect_name, input, counts);
if (errchk) {
    Show_Err("CANNOT WRITE SPECTRUM FILE");
    return 1;
}

***** Print Calculated Concentrations and MDA Flags to Data File *****
getdate(&d);                         /* Current Date */
t = time(NULL);                      /* Get Current Time */
timeptr = ctime(&t);                /* Convert to String */
for (i=0; i < 8; i++)
    timestr[i] = timeptr[i+1];        /* Strip out the Time Part */
timestr[8] = 0;

if (dfile) {
    dat = fopen("TEMPDB.BAD","rt+"); /* Open File if it Exists */
} else {
    dat = fopen("TEMPDB.DAT","rt+");
}

if (dat == NULL) {
    if (dfile) {
        dat = fopen("TEMPDB.BAD","wt"); /* Open File if it Doesn't Exist */
    } else {

```

```
        dat = fopen("TEMPDB.DAT","wt");
    }

    if (dat == NULL) {
        if (dfile) {
            Show_Err("CANNOT OPEN TEMPDB.BAD");
        } else {
            Show_Err("CANNOT OPEN TEMPDB.DAT");
        }
        return 1;
    } else {
        errchk = fseek(dat, 0L, SEEK_END);
        if (errchk) {
            fclose(dat);
            Show_Err("SEEK ERROR ON DATA FILE");
            return 1;
        }

        errchk = fprintf(prn,
"\'%s\',\'%s\',\'%s\',\'%g\',\'%d\',\'%g%c\',\'%g\',\'%g%c\',\'%g\',\'%g%c\',\'%g\',\'%s\',\'%02d-%02d-%02d\',\'%d\'\n",
ref_name,
samp_no[input],
&spec_name[2],
samp_wf[input],
samp_dy[input],
result[0], flag[0], errest[0],
result[1], flag[1], errest[1],
result[2], flag[2], errest[2],
timestr,
d.de_mon, d.da_day, (d.da_year % 100),
samp_br[input]);

        fclose(dat);
        if (errchk == EOF) {
            Show_Err("CANNOT WRITE TO DATA FILE");
            return 1;
        }
    }

    strcpy(spect_fname[input], spect_name); /* Save Spectrum File Name */
    sprintf(message,"SPECTRUM NAME IS: %s",spect_name); /* Print it out */
    Show_Err(message);

    return 0;
}

*****  
*          *  
*          Sample()  
*          *
```

```

/*
* Main menu for collecting sample spectra and analyzing concentrations. *
*****/
```

---

```

Sample()
{
    int choice = 0;

    int col_crd[SMENU_COL] = { 10, 33, 56};
    int row_crd[SMENU_ROW] = { 5, 8, 11};

    char *smenu_text[SMENU_COL][SMENU_ROW] = {"View 1",
                                                "Start 1",
                                                "Analyze 1",
                                                "View 2",
                                                "Start 2",
                                                "Analyze 2",
                                                "View 3",
                                                "Start 3",
                                                "Analyze 3");

    int i,j,k;
    int yn;
    int quit=0;

    FILE *prn;
    struct date d;

    if (ref_name[0] == 0) { /* Make sure a reference file is in memory */
        gotoxy(13,21);
        puts("A reference file must be loaded before analyzing. Okay");
        yn = Menu(0, 1, 1, 65, 21, 4, 1, 0, 0);
        return 0;
    }

    prn = fopen("PRN","Wt");
    if (prn != NULL) {
        getdate(&d);
        fprintf(prn,"\\nSOIL GAMMA ANALYSIS %02d/%02d/%02d REFERENCE FILE: %12s\\n\\n",
                d.da_mon, d.da_day, (d.da_year % 100), ref_name);

        fprintf(prn," SAMPLE ID      SPECTRUM      SAMPLE      pCi/g      pCi/g
                fprintf(prn," NAME          WEIGHT      DY      Ra-226      K-40      Th-232\\n");
        fprintf(prn,"-----  -----  -----  -----  -----  -----  -----\\n");

        fclose(prn);
    }

    for (i=0; i < NUM_INPUTS; i++)
        samp_status[i] = NOT_COLLECTED; /* Default Data Status */
}
```

```

while (quit == 0) { /* Loop until Exit is Confirmed */
    while (choice != -1) {
        //***** Check to make sure the collecting data is not altered *****/
        for (i=0; i < NUM_INPUTS; i++)
            if (invalid[i] && samp_status[i] == COLLECTING)
                samp_status[i] = NOT_COLLECTED;

        //***** Update the Data if any Inputs are Finished Collecting *****/
        Update_Samp();

        //***** Draw the Menu Screen *****/
        clrscr();
        gotoxy(23,2);
        puts("A N A L Y Z E   S A M P L E   M E N U");

        for (i=0; i < SMENU_ROW; i++)
            for (j=0; j < SMENU_COL; j++)
                Draw_Menu_Box(col_crd[j], row_crd[i], SBOX_W, SBOX_H, smenu_text[j][i]);

        gotoxy(40,24);
        printf("Current reference file: %s", ref_name);

        gotoxy(69,16);
        puts("Esc to quit");

        //***** Draw the Data Status Flags on the Menu Boxes *****/
        for (i=0; i < NUM_INPUTS; i++) {
            if (samp_status[i] == COLLECTING) {
                gotoxy(col_crd[i]+1, row_crd[1]+1);
                puts("++");
                gotoxy(col_crd[i]+SBOX_W-2, row_crd[1]+1);
                puts("++");
            }
            if (samp_status[i] == COLLECTED) {
                gotoxy(col_crd[i]+1, row_crd[1]+1);
                puts("**");
                gotoxy(col_crd[i]+SBOX_W-2, row_crd[1]+1);
                puts("**");
            }
        }

        //***** Get the Menu Choice *****/
        choice = Menu(choice, SMENU_ROW, SMENU_COL, col_crd[0], row_crd[0], SBOX_W, SBOX_H, 8, 0);
        //***** Update the Data if any Inputs are Finished Collecting *****/
        Update_Samp();

        //***** Carry out the Choice *****/
    }
}

```

```

switch (choice) {
    case 0:
    case 1:
    case 2:
        View_Spectrum(choice); /* Go Watch the Data Collecting */
        break;

    case 3:
    case 4:
    case 5:
        i = choice - 3;
        if (Setup_Samp(i) == 0) { /* Confirm, Weight, Days */
            samp_status[i] = COLLECTING; /* Flag for Collecting */
            Acquire_Off(i); /* Turn off the acquire */
            Valid_CTr(i); /* Clear the count data */
            timer[i] = 0L; /* Reset the acq timer */
            Ref_ROIs(i); /* Set all the ROIs */
            Acquire_On(i); /* Start acquire */
            strset(spect_fname[i], ' ');
            invalid[i] = 0; /* Data not altered */
        }
        break;

    case 6:
    case 7:
    case 8:
        i = choice - 6;
        if (samp_status[i] == COLLECTED) {
            if (Calc_Conc(i) == 0) /* Calc concentration */
                samp_status[i] = NOT_COLLECTED; /* Reset if Success */
        } else {
            gotoxy(13,21);
            puts("Data must be collected before analyzing. Okay");
            yn = Menu(0, 1, 1, 56, 21, 4, 1, 0, 0);
        }
        break;
    }

    //***** Confirm the Exit *****/
    gotoxy(8,21);
    puts("You will lose collected data if you have not saved it.");
    gotoxy(44,22);
    puts("Quit anyway? No Yes");
    yn = Menu(0, 1, 2, 59, 22, 3, 1, 3, 0);
    if (yn == 1) {
        quit = 1;
    } else {
        choice = 0;
    }
}

```

```
} return 0;
```

```
*****
*                               CALIB.C
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains Smooth() which averages the data in the count buffer
* over 5 channels, and Calibrate() which is the menu used to aid in
* calibrating the energy peaks to the proper channel.
*****
#include <stdio.h>
#include <soil.h>
#include <pca.h>

*****
*                               GLOBAL DATA REFERENCES
*
*****
extern long far *counts;           /* Buffer for Count Data */
extern long timer[NUM_INPUTS];    /* Live Timers */
extern int cursor[NUM_INPUTS];     /* Cursor Positions */
extern int roi_beg[NUM_INPUTS][MAX_ROI]; /* ROI Start Channels */
extern int roi_end[NUM_INPUTS][MAX_ROI]; /* ROI End Channels */
extern char invalid[NUM_INPUTS];   /* Data Altered Status */
extern char *spect_fname[NUM_INPUTS]; /* Spectra Names */

*****
*                               LOCAL DEFINITIONS
*
*****
#define CMENU_COL 3
#define CMENU_ROW 5
#define CBOX_W 13
#define CBOX_H 3
#define K_PEAK      242
#define CS_PEAK     109

*****
*                               Smooth()
*
* This module smooths out the data in the count buffer over 5 channels.
* Each channel is assigned the average of that channel and the 2 channels
*
```

```

* above and the 2 channels below. The first two and last two channels      *
* aren't assigned new values.                                              *
***** */
void Smooth()
{
    int          i,j;
    unsigned long sum;
    unsigned long temp[5];

    temp[1] = counts[0];           /* Initialize Temp Array */
    temp[2] = counts[1];

    for (i=2; i < CHANS_PER_INPUT-2; i++) {
        temp[0] = temp[1];         /* Hold counts in temp variables */
        temp[1] = temp[2];         /* to keep them from being */
        temp[2] = counts[i+0];     /* written over */
        temp[3] = counts[i+1];     /* Shift in new data, shift out */
        temp[4] = counts[i+2];     /* Old Data */

        sum = 0L;
        for (j=0; j<5; j++)       /* Sum the five values */
            sum += temp[j];
        counts[i] = sum / 5L;       /* Put the average back in buffer */
    }

    ****
    *
    *          Calibrate()          *
    *
    * This module draws the menu screens for the calibration, accepts user's   *
    * choice, and calls the proper functions. The purpose of the calibration   *
    * is to make energy peaks appear on the proper channels.                   *
    ****
Calibrate()
{
    int choice = 0;
    int input;

    int col_crd[CMENU_COL] = {11, 34, 57};
    int row_crd[CMENU_ROW] = {5, 8, 11, 14, 17};

    char *cmenu_text[CMENU_COL][CMENU_ROW] = {"View 1",
                                                "Potassium",
                                                "Cesium",
                                                "Generic",
                                                "Smooth 1",
                                                "View 2",
                                                "Potassium",
                                                "Cesium",
                                                "Generic",
                                                "Smooth 2",
                                                };
}

```

```
"View 3",
"Potassium",
"Cesium",
"Generic",
"Smooth 3");

int i,j;

while (choice != -1) {
    clrscr();

    gotoxy(25,2);
    puts("C A L I B R A T I O N      M E N U");
    gotoxy(69,25);
    puts("Esc to Quit");

    for (i=0; i<CMENU_ROW; i++)
        for (j=0; j<CMENU_COL; j++)
            Draw_Menu_Box(col_crd[j], row_crd[i], CBOX_W, CBOX_H, cmenu_text[j][i]);

    choice = Menu(choice, CMENU_ROW, CMENU_COL, col_crd[0], row_crd[0], CBOX_W, CBOX_H, 10, 0);

    switch (choice) {
        case 0:
        case 1:
        case 2:
            View_Spectrum(choice);
            break;

        case 3:
        case 4:
        case 5:
            input = choice - 3;

            Acquire_Off(input);           /* Turn off the acquire */
            Valid_CTr(input);           /* Clear the input */
            timerT[input] = 0L;          /* Reset the acquire timer */
            cursor[input] = K_PEAK;      /* Put Cursor on Potassium Peak */
            Clr_ROI(input);             /* Reset all the ROIs */
            roi_beg[input][0] = K_PEAK - 24; /* Set an ROI around the Peak */
            roi_end[input][0] = K_PEAK + 24;
            strset(spect_fname[input], ' ');
            Acquire_On(input);          /* Start acquire */
            break;

        case 6:
        case 7:
        case 8:
            input = choice - 6;

            Acquire_Off(input);           /* Turn off the acquire */
            Valid_CTr(input);           /* Clear the input */
            timerT[input] = 0L;          /* Reset the acquire timer */
            cursor[input] = CS_PEAK;     /* Put Cursor on Cesium Peak */
```

```
    Clr_ROI(input);          /* Reset all the ROIs */
    roi_beg[input][0] = CS_PEAK - 24; /* Set an ROI around the Peak */
    roi_end[input][0] = CS_PEAK + 24;
    strset(spect_fname[input], ' ');
    Acquire_On(input);      /* Start acquire */
    break;

case 9:
case 10:
case 11:
    input = choice - 9;
    Acquire_Off(input);     /* Turn off the acquire */
    Valid_CTr(input);      /* Clear the input */
    timerI[input] = 0L;     /* Reset the acquire timer */
    cursor[input] = 0;      /* Put Cursor on Channel 0 */
    Clr_ROI(input);        /* Reset all the ROIs */
    strset(spect_fname[input], ' ');
    Acquire_On(input);      /* Start acquire */
    break;

case 12:
case 13:
case 14:
    input = choice - 12;
    Acquire_Off(input);     /* Turn off the acquire */
    Valid_Read(input, counts); /* Read Count Data From PCA Card */
    Smooth();               /* Smooth Out the Data in Buffer */
    Valid_Write(input, counts); /* Write Count Data to PCA Card */
    invalid[input] = 1;      /* Data No Longer Valid */
    strset(spect_fname[input], ' ');
    break;
}
}
return 0;
}
```

```
*****
*          GETSTR.C
* Last Modified: 07/31/91
* Written By: Jim Davidson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains GetEStr() which is a modified version of C Gazette,
* Spring 1990, p.43 example by Andrew Binstock. Code was re-formatted by
* J.E.Wilson. Some modifications were made by Wilson.
*****
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

*****
*          LOCAL DEFINITIONS
*****
/* Key code definitions below as:
   0x47 00
   Scan code ----- ASCII code

Key names from CLIPPER 5.0 INKEY.CH file */

#define K_HOME      0x4700 /* Home */
#define K_CTRL_HOME 0x4737 /* Ctrl-Home */
#define K_SHFT_HOME 0x7700 /* Shft-Home */
#define K_END        0x4F00 /* End */
#define K_CTRL_END   0x4F31 /* Ctrl-End */
#define K_SHFT_END   0x7500 /* Shft-End */
#define K_ESC        0x011B /* Esc */
#define K_ENTER      0x1COD /* Enter */
#define K_LEFT       0x4800 /* Leftarrow */
#define K_RIGHT      0x4D00 /* Rightarrow */

#define K_BS         0x0E08 /* Backspace */
#define K_INS        0x5200 /* Ins */
#define K_DEL        0x5300 /* Del */
#define K_TAB         0x0F09 /* Tab */
#define K_SH_TAB     0x0F00 /* Shift-Tab */

#define K_UP         0x4800 /* Uparrow */
#define K_DOWN       0x5000 /* Dnarrow */
#define K_PGDN       0x5100 /* PgDn */
#define K_CTRL_W     0x1117 /* Ctrl-W */

#define INSERT      1 /* Insert mode flag */

```

```
#define OVERWRITE 0
#define ON      1 /* Cursor on/off */
#define OFF     0

#define LINE    2 /* Cursor Ins shape */
#define BLOCK   3 /* Cursor Overwrite shape */
#define HBLOCKTOP 4 /* added 7/9/91 */
                  /* Half block top cursor scan line */
#define TRUE    1
#define FALSE   0

#define UpdateCursor() SetCursor(Pos, SRow, SCol)
#define Updateline() SetCursor(Pos, SRow, SCol); cputs((char*)(Buff+Pos))
#define GotoRC(Row, Col) gotoxy(Col, Row)
```

```
*****
*                               LOCAL DATA DEFINITIONS
******
*****
```

```
static int Shape;
static void SetCursor(int, int, int);
static void Cursor(int);
int GetVideoType(void);
```

```
*****
*                               GetEStr()
* This function gets an editable string. The current length of the default
* string determines the field width.
* GetEStr() returns:
* K_ESC on Esc exit      (Current Str not changed)
* K_ENTER on Enter Key exit (Current Str changed)
* K_PGDN on PgDn Key exit (Current Str changed)
* K_CTRLW on Ctrl_W exit (Current Str changed)
* ****
unsigned int GetEStr(Curr, SRow, SCol, InsertMode, Caps, AlphaNum)
```

```
char *Curr;      /* Default String */
int SRow;        /* Starting Row */
int SCol;        /* Starting Column */
int *InsertMode; /* Pointer to Insert/Overtype Flag */
int Caps;        /* Force Uppercase Flag */
int AlphaNum;    /* AlphaNumeric | Digits only Flag */

{
  union REGS InRegs, OutRegs;
  int i, j, c, Done, Pos, SLen;
  char *Buff;
  int blank_flag = 0;
```

```
SLen = strlen(Curr);
Buff = (char *) calloc(SLen+1, 1); /* Allocate temp buffer */

textcolor(BLACK);
textbackground(LIGHTGRAY);
SetCursor(0, --SRow, --SCol);           /* ASM Row,Cols are 0,0 based, but */
for (i=1; i<=SLen; i++)             /* TC is 1,1 based, so decrement */
    putch(' ');
/* Position cursor */
if (*InsertMode == OVERWRITE)
    Shape = BLOCK;
else
    Shape = LINE;

Cursor(ON);
SetCursor(0, SRow, SCol);

/* Copy chars to screen */
i = j = 0;
if (*Curr != NULL) {
    for (; (j < SLen) && (Curr[i]); i++, j++) { /* i/j = char in/out */
        if (Caps)
            Curr[i] = toupper(Curr[i]);
        putch(Curr[i]);
        Buff[j] = Curr[i];                      /* Initialize buffer */
    }
    while (j < SLen)                         /* Pad buffer with spaces */
        Buff[j++] = ' ';
    Buff[SLen] = '\0';
}
Pos = 0;                                     /* Move cursor to start of str */
UpdateCursor();

Done = 0;
while (!Done) {                                /* Process keystrokes until GoodBye */
    InRegs.h.ah = 0x00;
    int86(0x16, &InRegs, &OutRegs);
    if (blank_flag)                          /* 1st time through: set to 1 */
        blank_flag = 2;                     /* After that: set to 2 */
    else
        blank_flag = 1;
    switch(OutRegs.x.ax) {                  /* Contains Scan + ASCII codes */
        case K_HOME:
            Pos = 0;
```

```
UpdateCursor();
break;

case K_END:
    for (Pos = SLen-1; Pos >= -1; Pos--)
        if (isgraph(Buff[Pos]))
            break;
    if (Pos != SLen-1)
        Pos += 1;
UpdateCursor();
break;

case K_LEFT:
    if (Pos > 0) {
        Pos--;
        UpdateCursor();
    }
break;

case K_RIGHT:
    if (Pos < SLen -1)
        Pos++;
    UpdateCursor();
break;

case K_BS:
    if (Pos == 0)
        break;
    if (*InsertMode == OVERWRITE) {
        Buff[--Pos] = ' ';
        Cursor(OFF);
        UpdateLine();
        UpdateCursor();
        Cursor(ON);
    } else {
        Cursor(OFF);
        for (i= SLen -1; i >= 0; i--) /* Find last non-space */
            if (i== 0 || isgraph(Buff[i]))
                break;
        Pos--;
        strcpy((void *) (Buff + Pos), (void *) (Buff + Pos + 1));
        Buff[SLen - 1] = ' ';
        UpdateLine();
        UpdateCursor();
        Cursor(ON);
    }
break;

case K_DEL:
    Cursor(OFF);
    for (i= SLen -1; i >= 0; i--) /* Find last non-space */
```

```
if (i==0 || isgraph(Buff[i]))
    break;
strcpy((void *) (Buff + Pos), (void *) (Buff + Pos + 1));
Buff[SLen - 1] = ' ';
UpdateLine();
UpdateCursor();
Cursor(ON);
break;

case K_INS:
    if (*InsertMode == OVERWRITE) {
        *InsertMode = INSERT;
        Shape = LINE;
    } else {
        *InsertMode = OVERWRITE;
        Shape = BLOCK;
    }
    Cursor(ON);
    break;

case K_ESC:
    Done = 1;
    break;

case K_UP:
case K_DOWN:
case K_PGDN:
case K_CTRL_W:
case K_ENTER:
    Done = 1;
    strcpy(Curr, Buff);
    break;

default:                                     /* Display char if OK */
    if (isprint(c=OutRegs.h.al)) {
        if (blank_flag == 1)          /* Blank string if not */
            strset(Buff, ' ');
            /* starting w/ cursor move */

        if (Caps)
            c = toupper(c);

        if (!AlphaNum) {
            if (!isdigit(c) && c != '.') {
                putchar('\a');
                break;
            }
        }
        if (*InsertMode == OVERWRITE) /* Mode is OverWrite */
            Buff[Pos] = (char) c;
        Cursor( OFF );
        UpdateLine();
    }
}
```

```
if (Pos < SLen-1)
    Pos++;
UpdateCursor();
Cursor( ON );

} else {                                /* Mode is Insert */

    memmove((void *) (Buff + Pos + 1),
            (void *) (Buff + Pos),
            SLen - Pos - 1);

    Buff[SLen] = '\0';
    Buff[Pos] = (char) c;
    Cursor(OFF);
    UpdateLine();
    if (Pos < SLen-1)
        Pos++;
    UpdateCursor();
    Cursor(ON);
}
}
break;
}

textcolor(LIGHTGRAY);
textbackground(BLACK);

Shape = LINE;
Cursor(ON);                                /* Just to be sure */

return( OutRegs.x.ax );                      /* Last scancode */
}

/*********************************************
*                                         *
*             SetCursor()                 *
*                                         *
* This function sets the cursor position by way of INT 10H, Function 2 *
*                                         *
*****************************************/
static void SetCursor(Col, TheRow, OffsetCol)

int Col;
int TheRow;
int OffsetCol;

{
    union REGS InRegs, OutRegs;
    static int Page = -1;

    if (Page == -1) { /* Since Page can not be -1, done only once */
        InRegs.h.ah = 0xF;
        int86(0x10, &InRegs, &OutRegs);
    }
}
```

```
    Page = OutRegs.h.bh;
}

InRegs.h.ah = 0x02;
InRegs.h.dh = (char) TheRow;
InRegs.h.dl = (char) Col + OffsetCol;
InRegs.h.bh = (char) Page;
int86(0x10, &InRegs, &OutRegs);
}

//*********************************************************************
*          Cursor()
*          *
* This function turns the cursor on or off by way of INT 10H, Function 1
*          *
//********************************************************************/
static void Cursor(Status)

int Status; /* ON | OFF */

{
    union REGS InRegs, OutRegs;
    static unsigned int Top, Bottom; /* Cursor scan lines */

    if ( (Top | Bottom) == 0) { /* Only done 1st time */
        switch( GetVideoType()) {
            case 1: /* MDA */
                default:
                    Top = 12;
                    Bottom = 13;
                    break;

            case 2: /* CGA */
            case 3: /* EGA */
                Top = 6;
                Bottom = 7;
                break;

            case 4: /* VGA */
                Top = 13;
                Bottom = 14;
                break;
        }
    }

    InRegs.h.ah = 0x1; /* Set cursor size */
    InRegs.h.cl = (char) Bottom;
    InRegs.h.ch = (char) (Shape == BLOCK ? HBLOCKTOP : Top);
    if (Status == OFF)
        InRegs.h.ch |= 0x10; /* Turn bit 4 on to hide cursor */
    int86(0x10, &InRegs, &OutRegs);
}
```

```
*****
*          GetVideoType()
*
* This function finds out the type of video hardware.
* Returns: 1 = MDA
*          2 = CGA
*          3 = EGA
*          4 = VGA/MCGA
*****
GetVideoType(void)
{
    union REGS InRegs, OutRegs;

    InRegs.h.ah = 0x1A;
    InRegs.h.cl = 0x0;
    int86(0x10, &InRegs, &OutRegs);
    if (OutRegs.h.al == 0x1A)           /* VGA/MCGA */
        return 4;

    InRegs.h.ah = 0x12;
    InRegs.h.bl = 0x10;
    int86(0x10, &InRegs, &OutRegs);
    if (OutRegs.h.bl != 0x10)          /* EGA */
        return 3;

    InRegs.h.ah = 0xF;
    int86(0x10, &InRegs, &OutRegs);
    if (OutRegs.h.al == 0x7)           /* Mono */
        return 1;
    else
        return 2;                     /* CGA */
}
```

```
*****
*          MAIN.C
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This Module is the start of the soil program. Main allocates memory,
* initializes interrupts, initializes the PCA and DMR hardware, calls
* Main_Menu, then cleans up before exiting to operating system.
*
*****/
```

```
#include <stdio.h>
#include <alloc.h>
#include <dos.h>

#include <soil.h>
#include <pca.h>

*****
*          GLOBAL DATA DEFINITIONS
*
*****/
```

```
long *counts;           /* Buffer for PCA Channel Counts */
long timer[NUM_INPUTS] = {0L, 0L, 0L};    /* Live Acquire Timers */
char acquire[NUM_INPUTS] = {0, 0, 0};      /* 1=Acquiring, 0=Not */
char invalid[NUM_INPUTS] = {0, 0, 0};      /* 1=Data Altered, 0=Okay */
double bkg[NUM_INPUTS][MAX_ROI];          /* 3 Background Matrices */
double con[NUM_INPUTS][MAX_ROI][MAX_ROI]; /* 3 Concentration Matrices */
double ref[NUM_INPUTS][MAX_ROI][MAX_ROI]; /* 3 Reference Count Matrices */

char ref_name[14];           /* Current Reference File Name */
/* IMPORTANT: ref_name should always reflect the current contents of
   the three matrices: bkg, con, and ref */

int vscale[NUM_INPUTS] = {0, 0, 0};        /* Spectra Vertical Scale */
int hscale[NUM_INPUTS] = {0, 0, 0};        /* Spectra Horizontal Scale */

int roi_beg[NUM_INPUTS][MAX_ROI] = {-1,-1,-1, /* Spectra ROI Beginnings */
-1,-1,-1;
-1,-1,-1};

int roi_end[NUM_INPUTS][MAX_ROI] = {-1,-1,-1, /* Spectra ROI Ends */
-1,-1,-1;
-1,-1,-1};

int insert = 0;                      /* Default to OverStrike for GetEStr */

char *spect_tname[NUM_INPUTS] =       /* Spectrum File Temporary Names */
```

```

        ","
        ","
char *spect_fname[NUM_INPUTS] = /* Spectrum File Names */
        ","
        ","
        ",";
/*
*          LOCAL DATA DEFINITIONS
*
*****
void interrupt (*OldHandler)(void); /* Saved Interrupt Function Pointer */

/*
*          main()
*
*****
main()
{
    int i;

    ref_name[0] = 0;           /* Blank Current Reference Name */

    counts = malloc(sizeof(long)*CHANS_PER_INPUT); /* Allocate Count Buffer */
    if (counts == NULL) {
        printf("Not Enough Free Memory\n");
        return 1;
    }

    for (i=0; i < NUM_INPUTS; i++) /* Turn off the DMR inputs */
        Acquire_Off(i);

    for (i=0; i < NUM_INPUTS; i++) /* Clear the PCA Card's Count Data */
        Valid_Clr(i);

    OldHandler = getvect(PCA_INT); /* Save the Old Interrupt Handler */
    setvect(PCA_INT, Handler); /* Set the New Interrupt Handler */

    Start_PCA();                /* Put PCA Card into Acquire */
    PCA_Interrupt_On();         /* Start the PCA Interrupts */
    Main_Menu();                /* Go Do the Main Menu */
    Stop_PCA();                 /* Take PCA Card out of Acquire */
    PCA_Interrupt_Off();        /* Stop the PCA Interrupts */
    setvect(PCA_INT, OldHandler); /* Reset the Old Interrupt Handler */
}

```

```
    free(counts);
    return 0;
}
```

```
*****  
*  
*          MAINMENU.C  
*  
* Last Modified: 07/31/91  
* Written By:  John E. Wilson  
* Compiler:    Borland C++ V2.0  
* Memory Model: Large  
*  
* Main_Menu() draws the main menu screen, accepts the user's choice, and  
* calls the proper function. Upon choosing Quit or Escape it returns to  
* main().  
*  
*****  
  
#include <stdio.h>  
  
*****  
*  
*          LOCAL DEFINITIONS  
*  
*****  
  
#define MMENU 4  
#define MBOX_W 22  
#define MBOX_H 3  
#define MCOL_COORD 30  
  
*****  
*          Main_Menu()  
*  
*****  
  
Main_Menu()  
{  
    int choice = 0;  
  
    int row_crd[MMENU] = {11, 14, 17, 20};  
  
    char *mmenu_text[MMENU] = {"Calibrate",  
                               "Analyze",  
                               "View Saved Spectra",  
                               "Quit"};  
  
    int i;  
  
    while (choice != 3 && choice != -1) {  
        clrscr();  
  
        gotoxy(27,1);  
        puts("Oak Ridge National Laboratory");  
        gotoxy(31,2);  
        puts("Grand Junction Office");
```

```
gotoxy(31,3);
puts("Soil Analysis Program");
gotoxy(36,5);
puts("Written By");
gotoxy(34,6);
puts("John E. Wilson");
gotoxy(36,7);
puts("June, 1991");
gotoxy(32,9);
puts("M A I N      M E N U");

for (i=0; i<MMENU; i++)
    Draw_Menu_Box(MCOL_COORD, row_crd[i], MBOX_W, MBOX_H, mmenu_text[i]);
choice = Menu(choice, MMENU, 1, MCOL_COORD, row_crd[0], MBOX_W, MBOX_H, 0, 0);

switch (choice) {
    case 0:
        Calibrate();
        break;

    case 1:
        Analyze();
        break;

    case 2:
        View_Saved();
        break;
}
return;
```

```
*****
*          MATRIX.C
* Last Modified: 07/31/91
* Written By:  John E. Wilson
* Compiler:    Borland C++ V2.0
* Memory Model: Large
*
* This module contains several routines for working on matrices. These
* matrices are defined as arrays of type double. Pointers to and size of
* the matrices are passed to the routines.
*****
#include <stdio.h>
#include <alloc.h>

*****
*          SubMatrix()
* Finds a Submatrix of a Matrix missing a specified row and column.
* Returns: 0
*****
SubMatrix(a,b,siz,row,col)

double *a; /* Square Matrix of Type Double */
double *b; /* Square Matrix of Type Double (one size smaller than Matrix A) */
int   siz; /* Size of Matrix A */
int   row; /* The Row to be Deleted from Matrix A */
int   col; /* The Column to be deleted from Matrix A */

{
    int i,j,k,l;

    for (i=k=0; i < siz-1; i++,k++) {
        if (k==row) /* Skip this row */
            k++;
        for (j=l=0; j < siz-1; j++,l++) {
            if (l == col) /* Skip this column */
                l++;
            b[(i*(siz-1))+(j)] = a[(k*siz)+(l)];
        }
    }
    return 0;
}

*****
*          Matrix_Det()
*
```

```
* Recursively finds the determinant of a matrix.  
* Returns: 1 if failed, 0 if Okay  
*  
*****  
Matrix_Det(a,det,siz)  
  
double *a; /* Square Array of Type Double */  
double *det; /* Address of Double that Will Contain the Determinant Upon Exit */  
int siz; /* Size of Array A */  
  
{  
    double dett; /* Temporary Variable */  
    double *s; /* Pointer to Temporary Submatrix */  
    int i;  
  
    if (siz == 1) {  
        *det=a[0]; /* If Size=1 then Determinant is that Single Element */  
        return 0;  
    } else {  
        /* Allocate Memory for a Submatrix */  
        if ((s=(double *)malloc(sizeof(double) * (siz-1) * (siz-1))) == NULL)  
            return 1;  
  
        *det=0.0;  
        for (i=0; i < siz; i++) {  
            /* Get a SubMatrix with the ith row and the 0th Column missing */  
            SubMatrix(a,s,siz,i,0);  
  
            /* Get Determinant of Submatrix */  
            if (Matrix_Det(s,&dett,siz-1) == 1) {  
                free(s);  
                return 1; /* If Failed Return Fail */  
            }  
  
            /* Multiply Element in First Column by the Deter. of submatrix */  
            dett = a[(i*siz)+(0)] * dett;  
  
            /* Accumulate Determinant */  
            *det += ((i%2)?(-dett):(dett));  
        }  
        free(s);  
        return 0;  
    }  
}  
*****
```

```
*
*          Matrix_Trans()          *
* Transposes Matrix A to Matrix B.      *
* Returns: 1 if failed, 0 if Okay       *
*
*****  
Matrix_Trans(a,b,row,col)  
  
double *a; /* Array of Type Double */  
double *b; /* Array of Type Double */  
int   row; /* Number of Rows in Matrix A (number of Columns in Matrix B) */  
int   col; /* Number of Columns in Matrix A (number of Rows in Matrix B) */  
  
{  
    double *t;      /* Pointer to Temporary Matrix */  
    int   i,j;  
  
    /* Allocate Memory for Temporary Matrix */  
    if ((t=(double *)malloc(sizeof(double) * (row) * (col))) == NULL)  
        return 1;  
  
    /* Transpose Matrix A to Temporary Matrix */  
    for (i=0; i < row; i++)  
        for (j=0; j < col; j++)  
            t[(j*row)+(i)] = a[(i*col)+(j)];  
  
    /* Transfer Temporary Matrix to Matrix B */  
    for (i=0; i < row; i++)  
        for (j=0; j < col; j++)  
            b[(i*col)+(j)] = t[(i*col)+(j)];  
  
    free(t);  
    return 0;  
}  
  
*****  
*          Matrix_Scal()          *
* Scales Matrix A to Matrix B.      *
* Returns: 0                      *
*****
Matrix_Scal(a,b,row,col,scal)  
  
double *a; /* Array of Type Double */  
double *b; /* Array of Type Double */  
int   row; /* Number of Rows */  
int   col; /* Number of Columns */  
double scal; /* Scalar of Type Double */  
  
C
```

```
int i,j;

for (i=0; i < row; i++)
    for (j=0; j < col; j++)
        b[(i*col)+(j)] = a[(i*col)+(j)] * scal;
return 0;
}

/********************* Matrix_Cof() ********************
 * Puts the Cofactor of Matrix A into Matrix B.
 * Returns: 1 if failed, 0 if Okay
 */
Matrix_Cof(a,b,siz)

double *a; /* Square Array of Type Double */
double *b; /* Square Array of Type Double */
int siz; /* Size of Arrays */

{
    double *t; /* Temporary Matrix */
    double *s; /* SubMatrix */
    double det; /* Temporary Determinant */
    int i,j;

    /* Allocate Memory for Temporary Matrix */
    if ((t=(double *)malloc(sizeof(double) * (siz) * (siz))) == NULL)
        return 1;

    /* Allocate Memory for SubMatrix */
    if ((s=(double *)malloc(sizeof(double) * (siz-1) * (siz-1))) == NULL) {
        free(t);
        return 1;
    }

    for (i=0; i < siz; i++) {
        for (j=0; j < siz; j++) {

            /* Get the Submatrix of A missing the ith Row and the jth Column */
            SubMatrix(a,s,siz,i,j);

            /* Find the Determinant of the Submatrix */
            if (Matrix_Det(s,&det,siz-1)==1) {
                free(s);
                free(t); /* If Failed Return Fail */
                return 1;
            }
        }
    }
}
```

```
/* Put the Cofactor into Temporary Matrix */
    t[(i*siz)+(j)] = ((i+j)%2)?(-det):(det);
}
}

/* Transfer the Temporary Matrix to Matrix B */
for (i=0; i < siz; i++)
    for (j=0; j < siz; j++)
        b[(i*siz)+(j)] = t[(i*siz)+(j)];

free(s);
free(t);
return 0;
}

*****
*          Matrix_Adj()          *
*          *          *
* Puts the Classic Adjoint for Matrix A into Matrix B.          *
* Returns: 1 if failed, 0 if Okay          *
*          *          *
*****
Matrix_Adj(a,b,siz)

double *a; /* Square Array of Type Double */
double *b; /* Square Array of Type Double */
int   siz; /* Size of Arrays */

{
    double *t; /* Temporary Matrix */

    /* Allocate Memory for Temporary Matrix */
    if ((t=(double *)malloc(sizeof(double) * (siz) * (siz))) == NULL)
        return 1;

    /* Put the Cofactor of Matrix A into Temporary Matrix */
    if (Matrix_Cof(a,t,siz) == 1) {
        free(t);
        return 1; /* If Failed Return Fail */
    }

    /* Transpose Temporary Matrix into Matrix B */
    if (Matrix_Trans(t,b,siz,siz) == 1) {
        free(t);
        return 1; /* If Failed Return Fail */
    }

    free(t);
    return 0;
}
```

```
}

/*********************************************
*          Matrix_Inv()                  *
*                                     *
* Puts the Inverse of Matrix A into Matrix B.   *
* Returns: 1 if failed, 0 if Okay               *
*                                     *
*****************************************/
Matrix_Inv(a,b,siz)

double *a; /* Square Array of Type Double */
double *b; /* Square Array of Type Double */
int    siz; /* Size of Arrays */

{
    double *t; /* Temporary Matrix */
    double det; /* Temporary Determinant */

    /* Allocate Memory for Temporary Matrix */
    if ((t=(double *)malloc(sizeof(double) * (siz) * (siz))) == NULL)
        return 1;

    /* Get the Determinant of Matrix A */
    if (Matrix_Det(a,&det,siz) == 1) {
        free(t);
        return 1;
    }

    /* Get Classic Adjoint of Matrix A into Temporary Matrix */
    if (Matrix_Adj(a,t,siz) == 1) {
        free(t);
        return 1;
    }

    if (det == 0.0) {      /* Watch for divide by zero */
        free(t);
        return 1;
    }

    /* Scale Temporary Matrix by inverse of Determinant into Matrix B */
    Matrix_Scal(t,b,siz,siz,(1.0/det));

    free(t);
    return 0;
}

/*********************************************
*          Matrix_Mul()                  *
*                                     *
```

```
* Puts the Product of Matrix A and Matrix B into Matrix C.          *
* Returns: 1 if failed, 0 if Okay                                     *
*                                                       *  
*****  
Matrix_Mul(a,b,c,rowa,colb,cola)  
  
double *a;    /* Array of Type Double */  
double *b;    /* Array of Type Double */  
double *c;    /* Array of Type Double */  
int   rowa;  /* Rows in Matrix A */  
int   colb;  /* Columns in Matrix B */  
int   cola;  /* Columns in Matrix A (Rows in Matrix B) */  
  
{  
    double *t; /* Temporary Matrix */  
    int   i,j,k;  
  
    /* Allocate Memory for Temporary Matrix */  
    if ((t=(double *)malloc(sizeof(double) * (rowa) * (colb))) == NULL)  
        return 1;  
    for (i=0; i < rowa; i++) {  
        for (j=0; j < colb; j++) {  
            /* Accumulate Sum of RowA * ColB */  
            t[(i*colb)+(j)] = 0.0;  
            for (k=0; k < cola; k++) {  
                t[(i*colb)+(j)] += a[(i*cola)+(k)] * b[(k*colb)+(j)];  
            }  
        }  
    }  
    /* Transfer Temporary Matrix to Matrix C */  
    for (i=0; i < rowa; i++)  
        for (j=0; j < colb; j++)  
            c[(i*colb)+(j)] = t[(i*colb)+(j)];  
    free(t);  
    return 0;  
}
```

```
*****
*          MENU.C
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contain Menu() which allows a user to pick an item from a
* that has been setup. It also contains Draw_Menu_Box() which is use for
* setting up a menu screen. Also here is GetKey_IT_Ready() which reads the
* keyboard directly. This module also contains some small support routines.
*
*****
```

```
#include <stdio.h>
#include <dos.h>

#include <scancode.h>

*****
*          LOCAL DEFINITIONS
*
*****
```

```
#define BOX_TOP_LEFT    201 /* Character Numbers */
#define BOX_TOP_RIGHT   187
#define BOX_BOT_LEFT    200
#define BOX_BOT_RIGHT   188
#define BOX_HORIZONTAL 205
#define BOX_VERTICAL    186
```

```
*****
*          Draw_Menu_Box()
*
* Put a menu box on the screen. Position, Size, and Text are Passed In.
*
*****
```

```
Draw_Menu_Box(x, y, width, height, text)

int x;      /* Column at left edge of box */
int y;      /* Row at top edge of box */
int width;  /* Width of box */
int height; /* Height of box */
char *text; /* Text to put inside box */

{
    int i,j;

    gotoxy(x, y);           /* Draw Top Row of Box */
    putch(BOX_TOP_LEFT);
```

```
for (i = 0; i < (width-2); i++)
    putch(BOX_HORIZONTAL);
putch(BOX_TOP_RIGHT);

for (j = 1; j <= (height-2); j++) { /* Draw Middle Rows of Box */
    gotoxy(x, y+j);
    putch(BOX_VERTICAL);
    for (i = 0; i < (width-2); i++)
        putch(' ');
    putch(BOX_VERTICAL);
}

gotoxy(x, y+height-1); /* Draw Bottom Row of Box */

putch(BOX_BOT_LEFT);
for (i = 0; i < (width-2); i++)
    putch(BOX_HORIZONTAL);
putch(BOX_BOT_RIGHT);

gotoxy(x+(width/2)-(strlen(text)/2), y+(height/2)); /* Draw the Text */
printf("%s",text);

return 0;
}

*****
*
*          Highlight()
*
* Highlights a menu box by drawing > < beside box.
*
*****
void Highlight(x, y, width, height)

int x;           /* Column at left edge of box */
int y;           /* Row at top edge of box */
int width;       /* Width of box */
int height;      /* Height of box */

{
    int i;

    for (i=0; i<height; i++) /* Highlight Left Side of Box */
        gotoxy(x-1,y+i);
        putch('>');
    }

    for (i=0; i<height; i++) /* Highlight Right Side of Box */
        gotoxy(x+width,y+i);
        putch('<');
}
```

```
*****
*          UnLight()
*
* Removes the Highlight from a menu box.
*
*****
```

```
void UnLight(x, y, width, height)

int x;           /* Column at left edge of box */
int y;           /* Row at top edge of box */
int width;       /* Width of box */
int height;      /* Height of box */

{
    int i;

    for (i=0; i<height; i++) { /* Blank Highlight on Left Side of Box */
        gotoxy(x-1,y+i);
        putch(' ');
    }

    for (i=0; i<height; i++) { /* Blank Highlight on Right Side of Box */
        gotoxy(x+width,y+i);
        putch(' ');
    }
}

*****
*          GetKey_If_Ready()
*
* Return the keyboard scan number of any key pressed. Return 0 if no key.
*
*****
```

```
char GetKey_If_Ready()

{
    unsigned int zf = 0;      /* 80x86 zero flag */
    union REGS regs;         /* register union */

    regs.h.ah = 0x01;         /* select function one */
    int86(0x16,&regs,&regs); /* * BIOS keyboard int */

    zf = regs.x.flags;       /* set flags to zf */
    zf = zf & 0x40;          /* get bit six only */

    if (zf != 0) {            /* if no key ready */
        return(0);             /* Return NULL */
    } else {
        regs.h.ah = 0x00;       /* get a keystroke from the BIOS buffer */
        int86(0x16,&regs,&regs); /* do the BIOS keyboard interrupt */
        return(regs.h.ah);      /* return the keyboard scan code */
    }
}
```

```
}

/*****
 *          *
 *      Menu()          *
 *          *
 * Allows Arrow and Enter Keys to Select an Option from the Menu. Position, *
 * Size, Number of Rows, Number of Columns, and Current Selection are Passed *
 * In. The New Selection Number is Returned (-1 if ESC).          *
 */
Menu(current, rows, cols, x, y, width, height, xb, yb)

int current;    /* Current Choice Number */
int rows;       /* Number of Rows in Menu */
int cols;       /* Number of Columns in Menu */
int X;          /* X Coordinate of Upper Left Box */
int y;          /* Y Coordinate of Upper Left Box */
int width;      /* Width of a Menu Choice Box */
int height;     /* Height of a Menu Choice Box */
int xb;         /* Blank Spaces between Boxes Horizontally */
int yb;         /* Blank Spaces between Boxes Vertically */

{
    int cur_row;
    int cur_col;
    char key;

    cur_row = current / cols;
    cur_col = current % cols;

    HighLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
    key = GetKey_If_Ready();

    while (key != ENTER && key != ESC) {
        switch (key) {
            case KEY_LEFT:
                if (cur_col > 0) {
                    UnLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
                    cur_col--;
                    HighLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
                }
                break;
            case KEY_RIGHT:
                if (cur_col < (cols-1)) {
                    UnLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
                    cur_col++;
                    HighLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
                }
                break;
            case KEY_UP:
                if (cur_row > 0) {
                    UnLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
                    cur_row--;

```

```
    HighLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
}
break;
case KEY_DOWN:
if (cur_row < (rows-1)) {
    UnLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
    cur_row++;
    HighLight(x+cur_col*(width+xb), y+cur_row*(height+yb), width, height);
}
break;
}
key = GetKey_If_Ready();
}

if (key == ESC) {
    return(-1); /* Indicate Abort */
} else {
    return((cur_row * cols) + cur_col); /* Return # of Highlighted Box */
}
```

```
*****
*          NEWREF.C
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains New_Ref() which is the main menu for building a new
* Reference File. It also contains Save_Ref() which physically saves the
* matrices BKG, CON, and REF to a reference file. Also here are several
* small support modules.
*
*****/
```

```
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <math.h>

#include <soil.h>
#include <pca.h>

*****
*          LOCAL DEFINITIONS
* *****/
```

```
#define NRMENU_COL 3
#define NRMENU_ROW 6

#define NRBOX_W 16
#define NRBOX_H 3

#define RFTIMES 10L /* How many times to oversample reference materials */

*****
*          GLOBAL DATA REFERENCES
* *****/
```

```
extern long *counts;           /* Buffer for Count Data */
extern long timer[NUM_INPUTS]; /* Live Timers */
extern int cursor[NUM_INPUTS]; /* Cursor Positions */
extern int roi_beg[NUM_INPUTS][MAX_ROI]; /* ROI Start Channels */
extern int roi_end[NUM_INPUTS][MAX_ROI]; /* ROI End Channels */
extern char acquire[NUM_INPUTS]; /* Acquire Status */
extern char invalid[NUM_INPUTS]; /* Data Altered Status */
extern double bkg[NUM_INPUTS][MAX_ROI]; /* Background Matrices */
extern double con[NUM_INPUTS][MAX_ROI]; /* Concentration Matrices */
extern double ref[NUM_INPUTS][MAX_ROI][MAX_ROI]; /* Reference Count Mats */
extern char ref_name[T4];      /* Reference File Name */
extern int insert;            /* Insert Mode for GetEstr */
```

```

extern char *spect_fname[NUM_INPUTS];           /* Spectra Names */

/*********************************************************************
*
*          LOCAL DATA DEFINITIONS
*
*********************************************************************/

int ref_status[NUM_INPUTS][MAX_ROI+1]; /* Status of Data for Each Input */
char trf_name[20];                      /* Temporary Reference File Name */
int ref_saved;                          /* Reference File Saved Flag */

char *rf_wt_str = "      ";             /* Default Reference Weight */
char *rf_con_str[MAX_ROI][MAX_ROI] = /* Default Reference Concentrations */
{
    {"50.2 ",           /*      */
     "0      ",         /*      */
     "0      ",         /*      */
     "0.617",           /*      */
     "104.6",           /*      */
     "0.368",           /*      */
     "1.24 ",           /*      */
     "1.87 ",           /*      */
     "71.2 "},          /*      */
};

/*********************************************************************
*
*          Setup_Ref()
*
* Get a reference ready to collect. Also check for conflicts.
*
*********************************************************************/
Setup_Ref(input, rff)

int input; /* DMR Input: 0,1,2 */
int rff;   /* Reference: 0=Background, 1=Radium, 2=Potassium, 3=Thorium */

{
    double weight;
    double tcon[MAX_ROI];
    int i;
    int yn;
    char *ref_label[MAX_ROI+1] = {"Background",
                                   "Radium",
                                   "Potassium",
                                   "Thorium"};

    ***** Make Sure Background has been Collected First *****

    if (rff != 0 && ref_status[input][0] != COLLECTED) {
        gotoxy(8, 24);
        puts("Background must be collected first. Okay");
        Menu(0, 1, 1, 46, 24, 4, 1, 0, 0);
    }
}

```

```
    return 1; /* Abort */
}

***** If Re-Collecting Background then Reset other References *****
if (rff == 0 && ref_status[input][0] == COLLECTED) {
    gotoxy(8, 23);
    puts("Re-collecting background will reset all references.");
    gotoxy(44, 24);
    puts("Re-collect anyway? No Yes");
    yn = Menu(0, 1, 2, 65, 24, 3, 1, 3, 0);
    if (yn == 1) {
        for (i=0; i<MAX_ROI+1; i++)
            ref_status[input][i] = NOT_COLLECTED;
        return 0;
    } else {
        return 1; /* Abort */
    }
}

***** See if this Input is Currently Collecting Anything Else *****
for (i=0; i < MAX_ROI+1; i++)
    if (ref_status[input][i] == COLLECTING) {
        gotoxy(8, 24);
        printf("%s is currently collecting.",ref_label[i]);
        gotoxy(46, 24);
        puts("Re-start anyway? No Yes");
        yn = Menu(0, 1, 2, 65, 24, 3, 1, 3, 0);
        if (yn == 1) {
            ref_status[input][i] = NOT_COLLECTED;
        } else {
            return 1; /* Abort */
        }
    }

***** See if Reference has already been collected for this Input *****
if (ref_status[input][rff] == COLLECTED) {
    gotoxy(8, 24);
    printf("%s has already been collected.",ref_label[rff]);
    gotoxy(44, 24);
    puts("Re-collect anyway? No Yes");
    yn = Menu(0, 1, 2, 65, 24, 3, 1, 3, 0);
    if (yn != 1)
        return 1; /* Abort */
}

***** Input the Reference Concentrations (Except for Background) *****
if (rff != 0) {
```

```
yn=0;
while (yn == 0) {

    clrscr();
    gotoxy(1, 8);

    printf("Input %s reference weight: ", ref_label[rff]);
    GetEStr(rf_wt_str, wherey(), wherex(), &Insert, 0, 0);
    weight = atof(rf_wt_str);

    gotoxy(1, 9);

    for (i=0; i < MAX_ROI; i++) {
        printf("\nInput %-9s concentration: ", ref_label[i+1]);
        GetEStr(rf_con_str[rff-1][i], wherey(), wherex(), &Insert, 0, 0);
        tcon[i] = atof(rf_con_str[rff-1][i]);

        con[input][i][rff-1] = weight * tcon[i] * ACQ_TIME;
    }

    /***** RePrint for Verification *****/
    clrscr();
    gotoxy(1, 8);

    printf("Input %s reference weight: %lg\n\n", ref_label[rff], weight);
    for (i=0; i < MAX_ROI; i++)
        printf("Input %-9s concentration: %lg\n", ref_label[i+1], tcon[i]);

    /***** Verify Input *****/
    gotoxy(25, 24);
    puts("Change Okay Cancel");
    yn = Menu(1, 1, 3, 25, 24, 6, 1, 3, 0);
}

if (yn == 2 || yn == -1)
    return 1; /* Abort */

}

ref_saved = 0; /* Reference File Not Saved */
return 0; /* Okay */
}

*****
*           Show_DskErr()
*
* Shows the file name where disk error happened. Allows user confirmation.
*
*****
```

```
Show_DskErr(fname)
```

```
char *fname; /* File Name where disk error occurred */

{
    gotoxy(2, 23);
    printf("DISK ERROR, File: %s", fname);

    gotoxy(2, 24);
    puts("Okay");

    Menu(0, 1, 1, 2, 24, 4, 1, 0, 0);

    return 0;
}

/*********************************************
*          Get_RfName()
*
* Gets the next reference name consisting of the date and a unique letter.
* Returns a pointer to the string (trf_name).
*
********************************************/
char *Get_RfName()

{
    struct date d;
    FILE *fh;

    /* Get system date and use it to make reference file name */
    getdate(&d);
    sprintf(trf_name,"REFER\\%02d%02d%02dA.REF",(d.da_year % 100),d.da_mon,d.da_day);

    /* Create a uniquely named reference file */
    fh = fopen(trf_name, "r");
    while (fh != NULL) {           /* See if file already exists */
        trf_name[12]++;
        /* If so change letter in filename */
        fclose(fh);
        /* Close old file */
        fh = fopen(trf_name, "r"); /* Try new file */
    }

    return trf_name;             /* Return address of file name */
}

/*********************************************
*          Update_Data()
*
* If any inputs are finished collecting then get data and update the flags.
* Also save the spectrum out to disk.
*
********************************************/
void Update_Data()
{
```

```
char *fname;
char spname[16];
int errchk;
int i,j,k;

for (i=0; i < NUM_INPUTS; i++) {
    for (j=0; j < MAX_ROI+1; j++) {
        if (acquire[i] == 0 && ref_status[i][j] == COLLECTING) {

            sprintf(spname, "A:");
                /* Start of Spectrum File Name */
            fname = Get_RfName();
            strcat(spname, &fname[6]);
            spname[9] = (char)(i + '1');
            spname[10] = 0;

            Valid_Read(i, counts);           /* Get Counts from PCA Card */

            if (j == 0) {                   /* BACKGROUND */
                bkg[i][0] = 0.0;           /* Sum ROI 1 */
                for (k=ROI_BEG_1; k <= ROI_END_1; k++)
                    bkg[i][0] += counts[k];

                bkg[i][0] =
                    floor((bkg[i][0] / RFTIMES) + 0.5);      /* OverSample */

                bkg[i][1] = 0.0;           /* Sum ROI 2 */
                for (k=ROI_BEG_2; k <= ROI_END_2; k++)
                    bkg[i][1] += counts[k];

                bkg[i][1] =
                    floor((bkg[i][1] / RFTIMES) + 0.5);

                bkg[i][2] = 0.0;           /* Sum ROI 3 */
                for (k=ROI_BEG_3; k <= ROI_END_3; k++)
                    bkg[i][2] += counts[k];

                bkg[i][2] =
                    floor((bkg[i][2] / RFTIMES) + 0.5);

                strcat(spname, ".BK");     /* Spectrum File Extension */
            } else {                     /* REFERENCES */
                ref[i][0][j-1] = 0.0;       /* Sum ROI 1 */
                for (k=ROI_BEG_1; k <= ROI_END_1; k++)
                    ref[i][0][j-1] += counts[k];

                ref[i][0][j-1] =
                    floor((ref[i][0][j-1] / RFTIMES) + 0.5); /* OverSample */

                ref[i][1][j-1] = 0.0;       /* Sum ROI 2 */
                for (k=ROI_BEG_2; k <= ROI_END_2; k++)
                    ref[i][1][j-1] += counts[k];

                ref[i][1][j-1] =
                    floor((ref[i][1][j-1] / RFTIMES) + 0.5);
            }
        }
    }
}
```

```
ref[i][2][j-1] = 0.0;           /* Sum ROI 3 */
for (k=ROI_BEG_3; k <= ROI_END_3; k++)
    ref[i][2][j-1] += counts[k];

ref[i][2][j-1] =
    floor((ref[i][2][j-1] / RFTIMES) + 0.5);

for (k=0; k < MAX_ROI; k++)      /* Minus Background */
    ref[i][k][j-1] -= bkg[i][k];

strcat(spname, ".RFO");        /* Spectrum File Extension */
spname[13] = (char)(j + '0');
}

errchk = SpSave(spname, i, counts); /* Save the Spectrum File */
if (errchk) {
    Show_DskErr(spname);
    return;
}

strcpy(spect_fname[i], spname); /* Set Spectrum File Name */
ref_status[i][j] = COLLECTED;   /* Update Data Status */
}

}

/*****************************************
*          Ref_ROIs()
* Sets up the ROIs for a specified input.
*****************************************/
Ref_ROIs(input)

int input; /* DMR Input: 0,1,2 */

{
    Clr_ROI(input);           /* Reset the ROIs */
    roi_beg[input][0] = ROI_BEG_1; /* Set Default ROIs */
    roi_end[input][0] = ROI_END_1;

    roi_beg[input][1] = ROI_BEG_2;
    roi_end[input][1] = ROI_END_2;

    roi_beg[input][2] = ROI_BEG_3;
    roi_end[input][2] = ROI_END_3;

    return 0;
}

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

```
*          Save_Ref()
*
* Physically saves the contents of matrices: REF, CON, & BKG to disk file.
* Returns 0 if successful, 1 if failed.
*
***** Save_Ref()
{
    FILE *fh;
    char *fname;           /* Name for File Name */
    int i,j,k;
    int errchk;

    /* Check to make sure that all data has been collected */
    for (i=0; i < NUM_INPUTS; i++)
        for (j=0; j < MAX_ROI+1; j++)
            if (ref_status[i][j] != COLLECTED) {
                gotoxy(8,24);
                puts("All data must be collected before saving. Okay");
                Menu(0, 1, 1, 52, 24, 4, 1, 0, 0);
                return 1;
            }

    fname = Get_RfnName(); /* Get next reference file name */
    fh = fopen(fname,"wt");
    if (fh == NULL) {
        Show_DskErr(fname); /* Cannot open file */
        return 1;
    } else {
        for (i=0; i < NUM_INPUTS; i++) /* Save REF matrices */
            for (j=0; j < MAX_ROI; j++)
                for (k=0; k < MAX_ROI; k++) {
                    errchk = fprintf(fh,"%f\n",ref[i][j][k]);
                    if (errchk == EOF) {
                        Show_DskErr(fname);
                        fclose(fh);
                        return 1;
                    }
                }

        for (i=0; i < NUM_INPUTS; i++) /* Save CON matrices */
            for (j=0; j < MAX_ROI; j++)
                for (k=0; k < MAX_ROI; k++) {
                    errchk = fprintf(fh,"%f\n",con[i][j][k]);
                    if (errchk == EOF) {
                        Show_DskErr(fname);
                        fclose(fh);
                        return 1;
                    }
                }
    }
}
```

```
        }

    for (i=0; i < NUM_INPUTS; i++) /* Save BKG matrices */
        for (j=0; j < MAX_ROI; j++) {
            errchk = fprintf(fh, "%f\n", bkg[i][j]);
            if (errchk == EOF) {
                Show_DskErr(fname);
                fclose(fh);
                return 1;
            }
        }
    fclose(fh);

    strcpy(ref_name, &fname[6]); /* Copy file name to reference name */
    fh = fopen("REFER\\LATEST", "wt"); /* Update Latest Name on Disk */
    if (fh == NULL) {
        Show_DskErr("REFER\\LATEST");
        return 1;
    } else {
        errchk = fprintf(fh, "%s", fname);
        fclose(fh);
        if (errchk == EOF) {
            Show_DskErr("REFER\\LATEST");
            return 1;
        }
        return 0;
    }
}

/*********************************************
*          New_Ref()
* Main menu for building a new reference file.
*****
New_Ref()
{
    int choice = 0;

    int col_crd[NRMENU_COL] = { 9, 33, 57};
    int row_crd[NRMENU_ROW] = { 5, 8, 11, 14, 17, 20};

    char *nrmenu_text[NRMENU_COL][NRMENU_ROW] = {"View 1",
                                                "Background",
                                                "Radium",
                                                "Potassium",
```

```
"Thorium",
"Save",
"View 2",
"Background",
"Radium",
"Potassium",
"Thorium",
"Quit",
"View 3",
"Background",
"Radium",
"Potassium",
"Thorium",
"Reset");

int i,j,k;
int yn;
int input_rff;
int quit=0;

ref_name[0] = 0; /* Reset Reference Name */
ref_saved = 0; /* Reset File Saved Flag */

for (i=0; i < NUM_INPUTS; i++)
    for (j=0; j < MAX_ROI+1; j++)
        ref_status[i][j] = NOT_COLLECTED; /* Default Data Status */

while (quit == 0) { /* Loop until Exit is Confirmed */

    while (choice != 16 && choice != -1) {

        ***** Check to make sure the collecting data is not altered *****/
        for (i=0; i < NUM_INPUTS; i++)
            if (invalid[i]) /* If data altered */
                for (j=0; j < MAX_ROI+1; j++)
                    if (ref_status[i][j] == COLLECTING)
                        ref_status[i][j] = NOT_COLLECTED; /* Then reset status */

        ***** Update the Data if any Inputs are Finished Collecting *****/
        Update_Data();

        ***** Draw the Menu Screen *****/
        clrscr();
        gotoxy(24,2);
        puts("NEW REFERENCE MENU");

        for (i=0; i < NRMENU_ROW; i++)
            for (j=0; j < NRMENU_COL; j++)
                Draw_Menu_Box(col_crd[j], row_crd[i], NRBOX_W, NRBOX_H, nrmenu_text[j][i]);

        ***** Draw the Data Status Flags on the Menu Boxes *****/
    }
}
```

```
for (i=0; i < NUM_INPUTS; i++)  
    for (j=0; j < MAX_ROI+1; j++) {  
        if (ref_status[i][j] == COLLECTING) {  
            gotoxy(col_crd[i]+1, row_crd[j+1]+1);  
            puts("+");  
            gotoxy(col_crd[i]+NRBOX_W-2, row_crd[j+1]+1);  
            puts("+");  
        }  
        if (ref_status[i][j] == COLLECTED) {  
            gotoxy(col_crd[i]+1, row_crd[j+1]+1);  
            puts("*");  
            gotoxy(col_crd[i]+NRBOX_W-2, row_crd[j+1]+1);  
            puts("*");  
        }  
    }  
  
    if (ref_saved) {  
        gotoxy(col_crd[0]+1, row_crd[5]+1);  
        puts("##");  
        gotoxy(col_crd[0]+NRBOX_W-2, row_crd[5]+1);  
        puts("##");  
    }  
  
    /****** Get the Menu Choice *****/  
    choice = Menu(choice, NRMENU_ROW, NRMENU_COL, col_crd[0], row_crd[0], NRBOX_W, NRBOX_H, 8, 0);  
    /****** Update the Data if any Inputs are Finished Collecting *****/  
    Update_Data();  
    /****** Carry out the Choice *****/  
    input = choice % 3;  
    rff = 3;  
  
    switch (choice) {  
        case 0:  
        case 1:  
        case 2:  
            View_Spectrum(input); /* Go Watch the Data Collecting */  
            break;  
  
        case 3:  
        case 4:  
        case 5:  
            rff--;  
        case 6:  
        case 7:  
        case 8:  
            rff--;  
        case 9:  
        case 10:  
        case 11:  
            rff--;  
        case 12:
```

```
case 13:
case 14:
    if (Setup_Ref(input, rff) == 0) { /* Confirm, Weight, Conc */
        ref_status[input][rff] = COLLECTING; /* Flag Collecting */
        Acquire_Off(input); /* Turn off the acquire */
        Valid_Ctr(input); /* Clear the count data */

        timer[input] = /* Let timer count extra */
        -(RFTIMES-1L) * ACQ_TICKS; /* by starting negative */

        Ref_ROIs(input); /* Set all the ROIs */
        strset(spect_fname[input], ' '); /* Clear Spectrum Name */
        Acquire_On(input); /* Start acquire */
        invalid[input] = 0; /* Data not altered */
    }
    break;

case 15: /* Save the reference file */
    if (ref_saved) {
        gotoxy(8,24);
        puts("Reference File Has Already been Saved. Okay");
        Menu(0, 1, 1, 48, 24, 4, 1, 0, 0);
    } else {
        if (Save_Ref() == 0)
            ref_saved = 1; /* Set Flag */
    }
    break;

case 17: /* Reset the collected data */
    gotoxy(8,24);
    puts("You will lose the collected data. Reset anyway? No Yes");

    yn = Menu(0, 1, 2, 59, 24, 3, 1, 3, 0);
    if (yn == 1) {

        for (i=0; i < NUM_INPUTS; i++) {
            Acquire_Off(i); /* Turn off Acquire */
            for (j=0; j < MAX_ROI+1; j++)
                ref_status[i][j] = NOT_COLLECTED; /* Default Status */
        }
    }
    break;
}

***** Confirm the Exit *****

if (ref_saved == 0) {
    gotoxy(8,23);
    puts("You Will lose collected data.");
}
gotoxy(8,24);
puts("Are You Sure You Want To Quit? No Yes");

yn = Menu(ref_saved, 1, 2, 41, 24, 3, 1, 3, 0);
if (yn == 1) {
    quit = 1;
```

```
    } else {
        choice = 0;
    }
    return 0;
}
```

```
*****
*          PCA.C
* Last Modified: 07/31/91
* Written By:    John E. Wilson
* Compiler:      Borland C++ V2.0
* Memory Model: Large
*
* This module contains the routines needed for controlling the PCA and DMR
* hardware. Also here is the handler for the PCA Card's timer interrupt.
* This module contains the Routines which read and write Nucleus-compatible
* spectrum files. Also here is the Valid_Read() routine which reads the
* count data for a PCA input and re-reads it if interference from the
* interrupt acquire stop is suspected.
*
*****
#include <stdio.h>
#include <dos.h>
#include <time.h>

#include <soil.h>
#include <pcat.h>
#include <dmr.h>
#include <soileasm.h>

*****
*          GLOBAL DATA REFERENCES
*
*****
extern long timer[NUM_INPUTS];
extern char acquire[NUM_INPUTS];
extern int roi_beg[NUM_TINPUTS][MAX_ROI];
extern int roi_end[NUM_INPUTS][MAX_ROI];

*****
*          LOCAL DATA DEFINITIONS
*
*****
DMRHEADER head;
int sound_cnt;

*****
*          Handler()
*
* This is the handler for the PCA Card interrupt. The PCA Card triggers
* this interrupt one hundred timers per second (minus the time the card is
* busy when in live time mode).
*****
```

```
*****
void interrupt Handler(void)
{
    disable();          /* Turn Off Interrupts */

    if (acquire[0]) {
        if (++timer[0] == ACQ_TICKS) {
            Acquire_Off(0);
            sound(500);           /* Low Tone */
            sound_cnt=50;         /* 1/2 Second */
        }
    }

    if (acquire[1]) {
        if (++timer[1] == ACQ_TICKS) {
            Acquire_Off(1);
            sound(1000);         /* Medium Tone */
            sound_cnt=50;
        }
    }

    if (acquire[2]) {
        if (++timer[2] == ACQ_TICKS) {
            Acquire_Off(2);
            sound(2000);         /* High Tone */
            sound_cnt=50;
        }
    }

    if (sound_cnt) {
        if ((--sound_cnt) == 0) {
            nosound();           /* Turn Tone Off */
        }
    }

    inportb(NEWPORT);          /* Send Acknowledge to PCA Card */
    outportb(ICC_CONTROL,ICC_ACKNOWLEDGE); /* ICC Acknowledge */
    enable();                  /* Turn On Interrupts */
}

*****
*             PCA_Interrupt_On()
*
* This routine sets up the PCA Card interrupt and starts it running.
*
void PCA_Interrupt_On()
{
    unsigned char mask;

    disable();
    mask = inportb(ICC_MASK);      /* Set Mask to Allow PCA Interr. */
}
```

```
mask = mask & PCA_INT_ON;
outportb(ICC_MASK,mask);

outportb(ICC_CONTROL,ICC_ACKNOWLEDGE);

inportb(NEWPORT);           /* Send Acknowledge to PCA Card */

enable();
}

/*********************************************
*          PCA_Interrupt_Off()
*
* This routine stops the PCA Card interrupt from running.
*
********************************************/
void PCA_Interrupt_Off()
{
    unsigned char mask;

    disable();

    mask = inportb(ICC_MASK);    /* Set Mask to Not Allow IRQ3 */
    mask |= PCA_TNT_OFF;
    outportb(ICC_MASK,mask);

    enable();
}

/*********************************************
*          Acquire_Off()
*
* This routine stops the specified PCA input from collecting any more count
* data.
*
********************************************/
void Acquire_Off(input)

int input; /* DMR Input: 0,1,2 */

{
    unsigned char two = 0x00;    /* flag for input bank two (inputs 5-8) */
    unsigned char bitsout;     /* data byte to send to the DMR card */

    acquire[input] = 0;         /* turn off the acquire flag for input */

    if (input > 4)
        two = 0x20; /* set bank two flag if we are processing inputs 5-8 */

    outportb(PCAPORT+1, 0x00); /* Select the DMR card */
    outportb(PCAPORT+2, two);  /* Select bank */
}
```

```
bitsout = Calc_DMR(two); /* calc the appropriate bit pattern */
outportb(PCAPORT, bitsout ); /* out the bit pattern to the DMR card */
}
```

```
*****
*          Acquire_On()           *
* This routine starts the specified PCA input collecting count data.   *
*****
```

```
void Acquire_On(input)
int input; /* DMR Input: 0,1,2 */

{
    unsigned char two = 0x00; /* flag for input bank two (inputs 5->8) */
    unsigned char bitsout; /* data byte to send to the DMR card */

    acquire[input] = 1; /* turn on the acquire flag for input */

    if (input >= 4)
        two = 0x20; /* set bank two flag if we are processing inputs 5 -> 8 */

    outportb(PCAPORT+1, 0xC0 ); /* Select the DMR card */
    outportb(PCAPORT+2, two ); /* Select bank */
    bitsout = Calc_DMR(two); /* calc the appropriate bit pattern */
    outportb(PCAPORT, bitsout ); /* out the bit pattern to the DMR card */
}
```

```
*****
*          Calc_DMR()           *
* This routine calculates the bit pattern needed to turn on the inputs of *
* the DMR which should be collecting data according to the acquire[] array. *
*****
```

```
unsigned char Calc_DMR(banktwo)
unsigned char banktwo; /* 0x00 for DMR Inputs 0-3, 0x20 for Inputs 4-7 */

{
    unsigned char low_dmr_bits[4] = { 0xef, 0xbf, 0xfe, 0xfb };
    unsigned char high_dmr_bits[4] = { 0x7f, 0xbf, 0xdf, 0xef };

    unsigned char bits = 0xff; /* working bit pattern */
    register int i;

    if (banktwo) {
        for (i = 0; i < 4; i++)
            if (acquire[i + 4]) /* if the input is in acquire */
                bits &= high_dmr_bits[i]; /* copy data from the bit table */
    }
}
```

```
if (banktwo) {
    for (i = 0; i < 4; i++)
        if (acquire[i + 4])
            bits &= high_dmr_bits[i];
}
```

```
    } else {          /* inputs 1,2,3 or 4 */
        for (i = 0; i < 4; i++)      /* loop through the inputs */
            if (acquire[i])        /* if the input is in acquire */
                bits &= low_dmr_bits[i]; /* copy data from the bit table */

    }
    return(bits);           /* return the bit pattern */
}

/*********************************************************************
*
*          Start_PCA()
*
* This routine puts the PCA Card into acquire mode.
*
********************************************************************/
void Start_PCA()
{
    char start_data[] = {0x29, 0x06, 0x7f, 0xe8, 0x00, 0xc0};
    register int i;

    Toggle_PCA_3(start_data[3]);
    for (i=0; i<6; i++)
        PCAOut(0x8000+i, start_data[i]);
}

/*********************************************************************
*
*          Stop_PCA()
*
* This routine takes the PCA Card out of acquire mode.
*
********************************************************************/
void Stop_PCA()
{
    char stop_data[] = {0x29, 0x06, 0x7f, 0xea, 0x00, 0x40};
    register int i;

    Toggle_PCA_3(stop_data[3]);
    for (i=0; i<6; i++)
        PCAOut(0x8000+i, stop_data[i]);
}

/*********************************************************************
*
*          Toggle_PCA_3()
*
* This routine turns bit 1 of PCA Control Register 3 on and off.
*
********************************************************************/
```

```
*****
void Toggle_PCA_3(value)
char value; /* Value to put in Control Register 3 */
{
    char flip = value | 0x02;

    value = value & 0xfd;
    PCAOut(0x8003, flip);      /* Digit 1 On */
    PCAOut(0x8003, value);    /* Digit 1 Off */
}

*****
/*
*          PCAOut()
*
* This routine sends a byte to the PCA Card.
*
*****
void PCAOut(address, value)
int address; /* Destination address on the PCA card for write */
char value; /* Value to write to that address */
{
    outportb(PCAPORT+2, (char)(address & 0xff));
    outportb(PCAPORT+1, (char)((address >> 8) & 0xff));
    outportb(PCAPORT+0, value);
}

*****
/*
*          ChArrDiff()
*
* See if the two character arrays are different.
*
* Returns 1 if different, 0 if same.
*
*****
ChArrDiff(arr1, arr2, num)
char *arr1; /* Pointer to character array 1 */
char *arr2; /* Pointer to character array 2 */
int num; /* Number of entries in the arrays */
{
    int i;

    for (i=0; i < num; i++)
        if (arr1[i] != arr2[i])
```

```
    return 1;
}
return 0;
}

//*********************************************************************
/*
*          Valid_Read()
*
* Reads the count data from PCA Card, ReReads if an input turned off.
* (If an input turns off during read, a false count can be expected)
*
*****/
void Valid_Read(input, counts)

int input; /* DMR Input: 0,1,2 */
long *counts; /* Pointer to array of long ints where count data is kept */

{
    int i;
    char acq[NUM_INPUTS];

    for (i=0; i < NUM_INPUTS; i++)
        acq[i] = acquire[i];

    Read_Input(input, counts);

    while (ChArrDiff(acq, acquire, NUM_INPUTS)) { /* Re-Read if any inputs */
        /* Turned off */
        for (i=0; i < NUM_INPUTS; i++)
            acq[i] = acquire[i];

        Read_Input(input, counts);
    }
}

//*********************************************************************
/*
*          Valid_Write()
*
* Turns off all inputs during the writing to the PCA Count data.
* (If an input turns off during write, corrupt data can appear on the Card)
*
*****/
void Valid_Write(input, counts)

int input; /* DMR Input: 0,1,2 */
long *counts; /* Pointer to array of long ints where count data is kept */

{
    int i;
    char acq[NUM_INPUTS];
```

```
for (i=0; i < NUM_INPUTS; i++) {
    acq[i] = acquire[i];           /* Save Acquire Status */
    Acquire_Off(i);              /* Turn off the Acquire */
}

Write_Input(input, counts);

for (i=0; i < NUM_INPUTS; i++)
    if (acq[i])                  /* Turn Acquire back on */
        Acquire_On(i);
}

/*************************************************************************
*                         Valid_Clr()
*
* Turns off all inputs during the clearing of the PCA Count data.
* (If an input turns off during write, corrupt data can appear on the Card)
*
*************************************************************************/
void Valid_Clr(input)

int input; /* DMR Input: 0,1,2 */

{
    int i;
    char acq[NUM_INPUTS];

    for (i=0; i < NUM_INPUTS; i++) {
        acq[i] = acquire[i];           /* Save Acquire Status */
        Acquire_Off(i);              /* Turn off the Acquire */
    }

    Clr_Input(input);

    for (i=0; i < NUM_INPUTS; i++)
        if (acq[i])                  /* Turn Acquire back on */
            Acquire_On(i);
}

/*************************************************************************
*                         SpSave()
*
* Save the spectrum to a Nucleus compatible file.
*
* Returns 1 if error, Returns 0 if Success.
*
*************************************************************************/
SpSave(fname, input, cdata)

char *fname; /* File name of spectrum */
int input;   /* DMR Input: 0,1,2 */
long *cdata; /* Pointer to array of long ints where count data is kept */
```

```
{  
FILE *fh;  
struct dfree free;  
struct date d;  
long avail;  
int i,j;  
char drive;  
int errchk;  
time_t t;  
char *timestr;  
  
char *month[12] = {"Jan",  
"Feb",  
"Mar",  
"Apr",  
"May",  
"Jun",  
"Jul",  
"Aug",  
"Sep",  
"Oct",  
"Nov",  
"Dec");  
  
***** Read the Count Data *****/  
Valid_Read(input, cdata);  
  
***** Put the ROIs into Count Data *****/  
for (i=0; i < MAX_ROI; i++) {  
    if (roi_beg[input][i] >= 0) {  
        for (j = roi_beg[input][i]; j <= roi_end[input][i]; j++) {  
            cdata[j] |= ((long)(i+1)) << 24;  
        }  
    }  
}  
  
***** Enough Disk Space? *****/  
if (fname[1] == ':') /* Get the Target Drive Number */  
    drive = fname[0] - 'A';  
else  
    drive = (char)(getdisk()); /* 0=A:, 1=B:, etc. */  
drive++; /* 1=A:, 2=B:, etc. */  
  
getdfree(drive, &free);  
if (free.df_sclus == 0xffff) /* Error on Get Disk Free */
```

```
    return 1;

avail = (long)free.df_avail * (long)free.df_bsec * (long)free.df_sclus;
if (avail < (sizeof(DMRHEADER) + BYTES_PER_INPUT)) /* Not Enough Space */
    return 1;

***** Initialize the Header Structure *****

head.ltimeflag = -1;                      /* Live Time Flag */
head.convergain = 4;                       /* Conversion Gain Number */
head.group = 8;                            /* Group Number */
head.phemode = -1;                         /* PHA Mode Flag */
head.numchanspm = CHANS_PER_INPUT;         /* Number of Channels */
head.majvers = 2;                          /* DMR-II Major Version Number */
head.minvers = 10;                         /* DMR-II Minor Version Number */
head.endheader = 210;                       /* Version Number * 100 */

if (fname[1] == ':') {                      /* File Name */
    strcpy(&head.idcodestr, &fname[2]);
} else {
    strcpy(&head.idcodestr, fname);
}

getdate(&d);                             /* Date Saved */
sprintf(head.pca_date,"%3s %02d %04d",
month[d.da_mon-1], d.da_day, d.da_year);

t = time(NULL);                          /* Time Saved */
timestr = ctime(&t);

sprintf(head.pca_time, "      am");
for (i=0; i < 8; i++)
    head.pca_time[i] = timestr[i+1];
i = ((timestr[11] - '0') * 10) + (timestr[12] - '0');

if (i == 0) {
    head.pca_time[0] = '1';
    head.pca_time[1] = '2';
}
if (i >= 12) {
    head.pca_time[9] = 'p';
}
if (i > 12) {
    i -= 12;
    head.pca_time[0] = (char)(i / 10 + '0');
    head.pca_time[1] = (char)(i % 10 + '0');
}

***** Open Spectrum File *****

fh = fopen(fname, "wb");
if (fh == NULL)
```

```
    return 1;

***** Write File Header *****

errchk = fwrite(&head, sizeof(DMRHEADER), 1, fh);
if (errchk != 1) {
    fclose(fh);
    return 1;
}

***** Write Count Data *****

errchk = fwrite(cdata, sizeof(long), CHANS_PER_INPUT, fh);
if (errchk != CHANS_PER_INPUT) {
    fclose(fh);
    return 1;
}

fclose(fh);
return 0;
}

*****
*
*          SpLoad()
*
* Load a spectrum from a Nucleus compatible file.
* Returns 1 if error, Returns 0 if Success.
*
*****
SpLoad(fname, input, cdata)

char *fname; /* File name of spectrum */
int input; /* DMR Input: 0,1,2 */
long *cdata; /* Pointer to array of long ints where count data is kept */

{
    FILE *fh;
    int i,j;
    int errchk;

***** Open the File *****

fh = fopen(fname, "rb");
if (fh == NULL)
    return 1;

***** Read the Header Structure *****

errchk = fread(&head, sizeof(DMRHEADER), 1, fh);
if (errchk != 1) {
    fclose(fh);
    return 1;
}
```

```
***** Check for Valid File Type *****/
if (head.numchanspm != CHANS_PER_INPUT) {
    fclose(fh);
    return 1;
}

***** Read Count Data *****/
errchk = fread(cdata, sizeof(long), CHANS_PER_INPUT, fh);
if (errchk != CHANS_PER_INPUT) {
    fclose(fh);
    return 1;
}

fclose(fh);

***** Set the ROIs *****/
Clr_ROI(input);

for (i=0; i < CHANS_PER_INPUT; i++) {
    j = (int)(cdata[i] >> 24);
    if (j > 0 && j <= MAX_ROI) {
        j--;
        if (roi_beg[input][j] == -1 || i < roi_beg[input][j])
            roi_beg[input][j] = i;
        if (roi_end[input][j] == -1 || i > roi_end[input][j])
            roi_end[input][j] = i;
    }
}

***** Write the Count Data to the PCA Card *****/
Valid_Write(input, cdata);
return 0;
}
```

```
*****
*          SOILASM.ASM
* Last Modified: 07/26/91
* Written By: John E. Wilson
* Assembler: Turbo Assembler V2.5
*
* These are routines that need to be executed quickly so they were written
* in assembly language.
*****
ZONE_HI      EQU      42
SPEC_Y       EQU      8
CHANS_PER_INPUT EQU      512
BYTES_PER_INPUT EQU      512 * 4
PCAPORT      EQU      1e0h

.MODEL    large
.DATA
EXTRN    _vscale:word, _counts:dword, _logtable:word
.CODE

*****
*          Get_Plot_Y()
* Return the Y-Coordinate for a Count given the scaling factor.
* int Get_Plot_Y(int input, int channel)
*****
_Get_Plot_Y proc    far
    push    bp           ;Linkage for C++
    mov     bp,sp
    mov     bx,[bp+6]      ;Get the input number from stack
    shl     bx,1
    mov     cx,_vscale[bx] ;Get the Vertical Scale Number
    cmp     cx,0           ;See if Scale is < 0
    jl      short log_scale ;Yes - go do Log Scale

bin_scale:           ;** Binary Scale - Shift Count Right **
                     ;** Number of Times in Scale   **
    mov     ax,[bp+8]
    shl     ax,1
    shl     ax,1           ;Index into Counts Array
    les     bx,_counts      ;Get Pointer to Counts in Es:Bx
    add     bx,ax           ;Add index to Pointer

```

```

        mov    dx,es:[bx+2] ;Get High Word of Count
        mov    ax,es:[bx]   ;Get Low Word of Count

bin_loop: jcxz  bin_done      ;Skip Shift if Scale is Zero
        shr    dx,1
        rcr    ax,1
        loop   bin_loop

bin_done: jmp   get_y_done
log_scale: jmp   get_y_done
        mov    ax,[bp+8]
        shl    ax,1
        shl    ax,1      ;Index into Counts Array

        les   bx,_counts ;Get Pointer to Counts in Es:Bx
        add   bx,ax      ;Add index to Pointer

        mov    dx,es:[bx+2] ;Get High Word of Count
        mov    ax,es:[bx]   ;Get Low Word of Count

        test  jz           ;Zone 5:   1 Meg - 16 Megs
NZone_5:  NZone_5

        mov    bx,dx
        shl    bx,1
        mov    ax,_logtable[bx] ; LogTable[Count >> 16]
        add   ax,5 * ZONE_HI ; + ( 5 * Zone height )

        jmp   short get_y_done

NZone_5:  test  jz           ;Zone 4:   64 K - 1 Meg
NZone_4:  NZone_4

        mov    cl,4+1
        and   dx,0fh
        shl    dx,cl      ;Use Lower 4 Bits of Dx
        and   ax,0f000h
        rol    ax,cl      ;and Upper 4 Bits of Ax
        mov    bx,dx
        or    bx,ax       ;For an Index into LogTable

        mov    ax,_logtable[bx] ; LogTable[Count >> 12]
        add   ax,4 * ZONE_HI ; + ( 4 * Zone height )

        jmp   short get_y_done

NZone_4:  test  jz           ;Zone 3:   4 K - 64 K
NZone_3:  NZone_3

        mov    cl,8-1
        and   ax,0ff00h

```

---

```

        shr    ax,cl      ;Use Upper 8 Bits of Ax
        mov    bx,ax      ;For an Index into LogTable

        mov    ax,_logtable[bx] ; LogTable[Count >> 8]

        add    ax,3 * ZONE_HI ; + ( 3 * Zone height )

NZone_3:   jmp    short get_y_done

        test   ax,0f00h   ;Zone 2: 256 - 4 K
        jz     NZone_2

        mov    cl,4-1
        and    ax,0ff0h
        shr    ax,cl      ;Use Middle 8 Bits of Ax
        mov    bx,ax      ;For an Index into LogTable

        mov    ax,_logtable[bx] ; LogTable[Count >> 4]

        add    ax,2 * ZONE_HI ; + ( 2 * Zone height )

        jmp    short get_y_done

NZone_2:   test   ax,0f0h    ;Zone 1: 16 - 256
        jz     NZone_1

        and    ax,0ffh
        shl    ax,1       ;Use Lower 8 Bits of Ax
        mov    bx,ax      ;For an Index into LogTable

        mov    ax,_logtable[bx] ; LogTable[Count]

        add    ax,1 * ZONE_HI ; + ( 1 * Zone height )

        jmp    short get_y_done

NZone_1:   test   ax,0fh     ;Zone 0: 1 - 16
        jz     get_y_done

        mov    cl,4+1
        and    ax,0fh
        shl    ax,cl      ; * 16
        mov    bx,ax      ;For an Index into LogTable

        mov    ax,_logtable[bx] ; LogTable[Count << 4]

get_y_done: and    ax,0ffh    ;Return Adjusted Y Coordinate in Ax
        neg    ax
        add    ax,SPEC_Y + 255

        pop    bp
        ret

_Get_Plot_Y endp

```

---

```

/*
 *          Clr_Input()
 *
 * Clear the PCA's Channel Count Data for a Specified Input
 *
 * void Clr_Input(int input)
 *
*****_Clr_Input proc far
    push    bp           ;Linkage for C++
    mov     bp,sp
    mov     ax,[bp+6]      ;Get the Input Number
    mov     cx,BYTES_PER_INPUT ;Multiply by # of Bytes per Input
    mul     cx
    mov     bx,ax          ;Address of First Byte of Input Data
    mov     dx,PCAPORT+2    ;Load Dx with Port Number

    clr_inp_lp:
    mov     al,bl          ;Output Address Low Byte
    out    dx,al
    dec    dx
    mov     al,bh          ;Output Address High Byte
    out    dx,al
    dec    dx
    xor    al,al          ;Zero the Data Byte
    out    dx,al
    inc    bx
    add    dx,2            ;Move to Next Data Byte
    ;Restore Dx
    loop   clr_inp_lp    ;Do All the Bytes for this Input
    pop    bp
    ret
_Clr_Input endp
*****_Read_Input()
 *
 * Read the PCA's Channel Count Data for a Specified Input
 *
 * void Read_Input(int input, long far *data)
 *
*****_Read_Input proc far
    push    bp           ;Linkage for C++
    mov     bp,sp
    push   es
    push   di           ;Save Important C++ Registers

```

---

```

les    di,[bp+8]           ;Get Long Pointer into Es:Di
mov    ax,[bp+6]
mov    cx,CHANS_PER_INPUT
mul    cx
shl    ax,1
shl    ax,1
mov    bx,ax
;Address of First Byte of Input Data */

mov    dx,PCAPORT+2
;Load Dx with Port Number

read_lp:
    mov    al,bl
    out   dx,al
    dec   dx

    mov    al,bh
    out   dx,al
    dec   dx

    in    al,dx
;Read the PCA Data Byte
    stosb
;Put Low Byte into Data Array
    inc   add
    bx   dx,2
;Move to Next Data Byte
;Restore Dx

    mov    al,bl
    out   dx,al
    dec   dx

    mov    al,bh
    out   dx,al
    dec   dx

    in    al,dx
;Put Middle Byte into Data Array
    stosb
    inc   add
    bx   dx,2

    mov    al,bl
    out   dx,al
    dec   dx

    mov    al,bh
    out   dx,al
    dec   dx

    in    al,dx
;Put High Byte into Data Array
    stosb

```

```

add    bx,2
add    dx,2

xor    al,al      ;Clear ROI Byte
stosb

loop   read_lp     ;Do All the Channels for this Input
pop    di          ;Restore Important C++ Registers
pop    es
pop    bp
ret

_Read_Input endp

*****
*           Write_Input()          *
* Write the Counts Buffer to the PCA Card for a Specified Input *
* void Write_Input(int input, long far *data)                    *
*****


_Write_Input proc  far
    push   bp          ;Linkage for C++
    mov    bp,sp
    push   ds          ;Save Important C++ Registers
    push   si
    lds   si,[bp+8]    ;Get Long Pointer into Ds:Si
    mov    ax,[bp+6]    ;Get the Input Number
    mov    cx,CHANS_PER_INPUT ;Multiply by # of Channels per Input */
    mul    cx
    shl    ax,1         ;And Multiply By 4 to get
    shl    ax,1
    mov    bx,ax        ;Address of First Byte of Input Data */

    mov    dx,PCAPORT+2 ;Load Dx with Port Number

write_lp:
    mov    al,bl        ;Output Address Low Byte
    out    dx,al
    dec    dx

    mov    al,bh        ;Output Address High Byte
    out    dx,al
    dec    dx

    lodsb             ;Get the Low Byte from Data Array
    out    dx,al        ;Write the PCA Data Byte
    inc    bx           ;Move to Next Data Byte

```

---

```

add    dx,2          ;Restore Dx

mov    al,bl
out   dx,al
dec   dx

mov    al,bh
out   dx,al
dec   dx

lodsb      ;Get Middle Byte from Data Array

out   dx,al
inc   bx
add   dx,2

mov    al,bl
out   dx,al
dec   dx

mov    al,bh
out   dx,al
dec   dx

lodsb      ;Get High Byte from Data Array

out   dx,al
inc   bx
add   dx,2

mov    al,bl
out   dx,al
dec   dx

mov    al,bh
out   dx,al
dec   dx

lodsb      ;Get ROI Byte from Data Array

out   dx,al
inc   bx
add   dx,2

loop  write_lp       ;Do All the Channels for this Input
pop   si
pop   ds             ;Restore Important C++ Registers
pop   bp

```

```
    ret  
_Write_Input endp  
PUBLIC  _Get_Plot_Y  
PUBLIC  _Clr_Input  
PUBLIC  _Read_Input  
PUBLIC  _Write_Input  
end
```

```

*****
*          VIEWSAVD.C
*
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains View_Saved() which allows saved spectra to be loaded
* and viewed or printed.
*
*****/




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

#include <soil.h>
#include <pca.h>

*****/
*          GLOBAL DATA REFERENCES
*
*****/




extern char invalid[NUM_INPUTS];      /* Data Altered Status */
extern long far *counts;             /* Buffer for Count Data */
extern int insert;                  /* Insert Mode for GetEstr */
extern char *spect_tname[NUM_INPUTS]; /* Temporary Spectra Names */
extern char *spect_fname[NUM_INPUTS]; /* Spectra Names */

*****/
*          LOCAL DEFINITIONS
*
*****/




#define VMENU_COL 3
#define VMENU_ROW 3
#define VBOX_W 11
#define VBOX_H 3

#define PMENU_ROW 3
#define PCOL_COORD 34
#define PBOX_W 12
#define PBOX_H 3

*****/
*          View_Saved()
*
* Allow saved spectra to be loaded in and viewed on the screen, or printed
* to a LaserJet or a TI-855 printer.
*
*****/

```

```
*****
View_Saved()
{
    int choice = 0;
    int printer = 0;

    int col_crd[VMENU_COL] = {13, 36, 59};
    int row_crd[VMENU_ROW] = {5, 8, 11};

    char *vmenu_text[VMENU_COL][VMENU_ROW] = {"View 1",
                                                "Load 1",
                                                "Print 1",
                                                "View 2",
                                                "Load 2",
                                                "Print 2",
                                                "View 3",
                                                "Load 3",
                                                "Print 3"};

    int pmow_crd[PMENU_ROW] = {6, 9, 12};
    char *pmenu_text[PMENU_ROW] = {"TI - 855",
                                   "LaserJet",
                                   "Quit"};

    int input;
    int i,j;
    int errchk;

    while (choice != -1) {
        clrscr();
        gotoxy(19,2);
        puts("V I E W   S A V E D   S P E C T R A   M E N U");
        gotoxy(69,25);
        puts("Esc to Quit");

        for (i=0; i<VMENU_ROW; i++)
            for (j=0; j<VMENU_COL; j++)
                Draw_Menu_Box(col_crd[j], row_crd[i], VBOX_W, VBOX_H, vmenu_text[j][i]);

        choice = Menu(choice, VMENU_ROW, VMENU_COL, col_crd[0], row_crd[0], VBOX_W, VBOX_H, 12, 0);

        switch (choice) {
            case 0:
            case 1:
            case 2:
                View_Spectrum(choice);
                break;
            case 3:

```

```

case 4:
case 5:
    input = choice-3;           /* Which Input */
    Acquire_Off(input);         /* Turn off the acquire */
    gotoxy(1,18);              /* Input File Name */
    printf("Input the File Name: ");
    GetEStr(spect_tname[input],18,22,&insert,1,1);

    /* Load Spectrum */
    errchk = SpLoad(spect_tname[input], input, counts);

    if (errchk) {               /* Couldn't Load Spectrum */
        gotoxy(1,20);
        printf("Cannot Load Spectrum File: %s", spect_tname[input]);
        gotoxy(39,22);
        puts("Okay");
        Menu(0, 1, 1, 39, 22, 4, 1, 0, 0);
        strset(spect_fname[input], ' '); /* Clear Spectrum Name */
    } else {                    /* Did Load Spectrum */
        strcpy(spect_fname[input], spect_tname[input]);
        invalid[input] = 0;
    }
    break;

case 6:
case 7:
case 8:
    input = choice-6;
    Valid_Read(input, counts);
    clrscr();

    gotoxy(27,4);
    puts("C H O O S E   P R I N T E R");

    for (i=0; i<VMENU_COL; i++)
        Draw_Menu_Box(PCOL_COORD, prow_crd[i], PBOX_W, PBOX_H, pmenu_text[i]);

    printer = Menu(0, PMENU_ROW, 1, PCOL_COORD, prow_crd[0], PBOX_W, PBOX_H, 0, 0);

    gotoxy(26,18);
    puts("Printing, Please Stand By...");

    switch (printer) {
        case 0:
            Prn_Spec_TI(input);
            break;
        .
        case 1:
            Prn_Spec_LJ(input);
            break;
    }
}

```

```
        break;  
    }  
    return 0;  
}
```

```

*****
*          VIEWSPEC.C
* Last Modified: 07/31/91
* Written By: John E. Wilson
* Compiler: Borland C++ V2.0
* Memory Model: Large
*
* This module contains the routines needed for viewing and controlling the
* display of a spectrum. The main routine is View_Spectrum(). Also here are
* Prn_Spec_LJ() and Prn_Spec_TL() which print a spectrum to the printers.
* Numerous support routines needed for display and setup are here also.
*
*****/
```

```

#include <stdio.h>
#include <graphics.h>
#include <math.h>

#include <soil.h>
#include <scancode.h>
#include <soilasm.h>
#include <pca.h>

*****
*          GLOBAL DATA REFERENCES
* ****
extern char acquire[NUM_INPUTS];      /* Acquire Status for the Inputs */
extern char invalid[NUM_INPUTS];     /* Data Altered Status */
extern long *counts;                 /* Buffer for Count Data */
extern long timer[NUM_INPUTS];       /* Live Timer Counters */
extern int vscale[NUM_INPUTS];        /* Spectra Vertical Scale */
extern int hscale[NUM_INPUTS];        /* Spectra Horizontal Scale */
extern int roi_beg[NUM_INPUTS][MAX_ROI]; /* Spectra ROI Beginnings */
extern int roi_end[NUM_INPUTS][MAX_ROI]; /* Spectra ROI Ends */
extern char *spect_fname[NUM_INPUTS]; /* Spectra Names */

*****
*          LOCAL DEFINITIONS
* ****
#define DOT    0
#define FILL   1
#define SPEC_X 116
#define SPEC_Y 8

*****
*          LOCAL DATA DEFINITIONS
* ****

```



```
"32K",
"64K",
"128K",
"256K",
"512K",
"1M",
"2M",
"4M",
"8M",
"16M");

/*********************************************
*                                         *
*             Clr_ROI(input)           *
*                                         *
* This routine clears the regions of interest for a particular input. *
*                                         *
*****************************************/
Clr_ROI(input)

int input; /* DMR Input: 0,1,2 */

{
    int i;

    for (i=0; i < MAX_ROI; i++) {
        roi_beg[input][i] = -1;
        roi_end[input][i] = -1;
    }
    return 0;
}

/*********************************************
*                                         *
*             Fill_LogTable()          *
*                                         *
* This routine fills the log table of y coordinates for the log scale *
* display to the screen and also to the TI printer.                     *
*                                         *
*****************************************/
void Fill_LogTable()
{
    int i;
    double temp;

    for (i=0; i < 16; i++)           /* first 16 entries are 0 */
        logtable[i] = 0;

    for (i=16; i < 256; i++) {
        temp = log(i / 16.0);      /* log(1..16 [in 1/16 steps]) */
        temp = temp / log(256 / 16.0); /* Range from 0 to 1 */
        temp = temp * 42.0;         /* Range from 0 to 42 */
    }
}
```

```

temp = floor(temp + 0.5);      /* Round Up */
logtable[i] = (int)temp;       /* Put it into Table */
}

*****
*           Init_Plot()          *
* This routine initializes all the variables needed for the screen display. *
*****
void Init_Plot(input)
int input; /* DMR Input: 0,1,2 */
{
    int x, half;

    if (plot_mode[input] == DOT) {           /* DOT MODE */
        for (x=0; x < CHANS_PER_INPUT; x++) /* Put at Top of Box */
            plot[x] = SPEC_Y;
    } else {                                /* FILL MODE */
        for (x=0; x < CHANS_PER_INPUT; x++) /* Put at Bottom of Box */
            plot[x] = SPEC_Y+255;
    }

    cursor_x = SPEC_X;                      /* Default Cursor Position */
    cursor_top = SPEC_Y;
    cursor_bot = SPEC_Y;

    last_count = 0xffff000000;                /* Force Channel Count to Print */
    last_time = -1;                         /* Force Timer Count to Print */
    last_acquire = -1;                      /* Force Acquire Status to Print */
    last_sum = -1;                          /* Force ROI Sum to Print */

    ***** Calculate new Screen Beginning and Ending Channels *****/
    half = 512 >> (hscale[input] + 1);     /* Half of Screen */

    ***** Start at Nearest 1/2 Screen Chunk *****/
    scr_beg[input] = cursor[input] - (cursor[input] % half);

    ***** Set Last Channel *****/
    scr_end[input] = scr_beg[input] + (half * 2);

    ***** Clip at End of Spectrum *****/
    while (scr_end[input] > 512) {

```

```

scr_beg[input] -= half;           /* Backup 1/2 Screen */
scr_end[input] -= half;
}

/*
*          Clear_Spec_Box()
*
* This routine clears the spectrum box and calls Init_Plot().
*/
void Clear_Spec_Box(input)

int input; /* DMR Input: 0,1,2 */

{
    int spec_box[8] = {SPEC_X-1,      SPEC_Y-1,
                      SPEC_X+512,   SPEC_Y-1,
                      SPEC_X+512,   SPEC_Y+256,
                      SPEC_X-1,      SPEC_Y+256};

    setcolor(WHITE);
    setfillstyle(SOLID_FILL, BLACK);
    fillpoly(4, spec_box);

    Init_Plot(input);           /* Initialize the Plot Array */
}

/*
*          Blank()
*
* This routine blanks out a box on the screen (where text will be drawn).
*/
void Blank(box)

int *box; /* Pointer to array of box coordinates */

{
    setcolor(BLUE);
    setfillstyle(SOLID_FILL, BLUE); /* Draw a Solid Blue Rectangle */
    fillpoly(4, box);
}

/*
*          Draw_Acquire()
*
* Draw what the F1 key will do to acquire (STOP or START).
*/

```

```

void Draw_Acquire(input)
int input; /* DMR Input: 0,1,2 */
{
    int box[8] = {36,32, 76,32, 76,40, 36,40};
    char *str;

    if (last_acquire != acquire[input]) {
        Blank(box);
        if (acquire[input]) {
            str = "STOP";
        } else {
            str = "START";
        }
        setcolor(WHITE);
        outtextxy(36, 32, str);
        last_acquire = acquire[input];
    }
}

/*********************************************
*          Draw_Plot_Mode()           *
* Draw what the FB key will do to draw mode (DOT or FILL).   *
*****************************************/
void Draw_Plot_Mode(input)
int input; /* DMR Input: 0,1,2 */
{
    int box[8] = {36,144, 68,144, 68,152, 36,152};
    char *str;

    Blank(box);
    if (plot_mode[input] == DOT) {
        str = "FILL";
    } else {
        str = "DOT";
    }
    setcolor(WHITE);
    outtextxy(36, 144, str);
}

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

```

```

/*
* Draw_Channel()
*
* Draw the channel number under the cursor.
*
*****void Draw_Channel(input)
int input; /* DMR Input: 0,1,2 */
{
    int box[8] = {76,272, 100,272, 100,280, 76,280};

    Blank(box);
    setcolor(WHITE);
    sprintf(buffer,"%3d",cursor[input]);
    outtextxy(76, 272, buffer);
}

/*
* Draw_Counts()
*
* Draw the counts of the channel under the cursor.
*
*****void Draw_Counts(input)
int input; /* DMR Input: 0,1,2 */
{
    unsigned long count;
    int box[8] = {68,288, 132,288, 132,296, 68,296};

    count = counts[cursor[input]] & 0xffff; /* Get Counts for Channel */
    if (count != last_count) /* If it Changed, Redraw */
    {
        Blank(box);
        setcolor(WHITE);
        sprintf(buffer,"%8ld",count);
        outtextxy(68, 288, buffer);
        last_count = count;
    }
}

/*
* Draw_Cursor()
*/

```

```

/*
* Draw the cursor in the spectrum box. *
***** */
void Draw_Cursor(input)

int input; /* DMR Input: 0,1,2 */

{
    int new_x,new_bot,new_top;

    /*** Calc Coords of Current Cursor ***/
    new_x = SPEC_X + (cursor[input] - scr_beg[input]) * (1 << hscale[input]);
    new_bot = Get_Plot_Y(input,cursor[input]) - 1;
    if (new_bot < SPEC_Y)
        new_bot = SPEC_Y;
    new_top = new_bot - 16;
    if (new_top < SPEC_Y)
        new_top = SPEC_Y;

    /*** If it Moved then Redraw It ***/
    if ((new_x != cursor_x) || (new_bot != cursor_bot)) {
        setcolor(BLACK);
        line(cursor_x, cursor_top, cursor_x, cursor_bot);

        setcolor(LIGHTGREEN);
        line(new_x, new_top, new_x, new_bot);

        cursor_x = new_x;
        cursor_bot = new_bot;
        cursor_top = new_top;
    }
}

/*
*          Draw_Time() *
* Draws the number of seconds acquires so far. *
***** */
void Draw_Time(input)

int input; /* DMR Input: 0,1,2 */

{
    int time;
    int box[6] = {92,312, 144,312, 144,320, 92,320};

    time = (int)(timer[input] / 100L); /* 100 Ticks per Second */
}

```

```

if (time != last_time) {           /* If it Changed, Redraw */
    Blank(box);
    setcolor(WHITE);
    sprintf(buffer,"%5d", time);
    outtextxy(92, 312, buffer);
    last_time = time;
}

*****
*          Draw_VScale()          *
* Draws the vertical scaling factor. *
*****
void Draw_VScale(input)
int input; /* DMR Input: 0,1,2 */
{
    int box[8] = {308,272, 340,272, 340,280, 308,280};

    Blank(box);
    setcolor(WHITE);
    outtextxy(308, 272, vsc_label[vscale[input]+1]);
}

*****
*          Draw_HScale()          *
* Draws the horizontal scaling factor. *
*****
void Draw_HScale(input)
int input; /* DMR Input: 0,1,2 */
{
    int box[8] = {564,272, 571,272, 571,280, 564,280};

    Blank(box);
    setcolor(WHITE);
    sprintf(buffer,"%1d", (1 << hscale[input]));
    outtextxy(564, 272, buffer);
}

```

```

*****
*          Draw_FName()          *
*          *          *
* Draws the file name of the spectrum.  *
*          *          *
*****
void Draw_FName(input)

int input; /* DMR Input: 0,1,2 */

{
    int box[8] = {100,332, 340,332, 340,340, 100,340};

    Blank(box);
    setcolor(WHITE);
    outtextxy(100, 332, spect_fname[input]);
}

*****
*          Draw_Cur_ROI()          *
*          *          *
* Draws a > next to the ROI that will be affected by the F3 and F4 keys.  *
*          *          *
*****
void Draw_Cur_ROI(input)

int input; /* DMR Input: 0,1,2 */

{
    int box[8] = {348,308 , 355,308 , 355,340 , 348,340};

    Blank(box);
    setcolor(WHITE);
    outtextxy(348, ((cur_roi[input] * 12) + 308), ">");
}

*****
*          Draw_ROI_Beg()          *
*          *          *
* Draws the first channel number of the ROI.  *
*          *          *
*****
void Draw_ROI_Beg(input, roi)

int input; /* DMR Input: 0,1,2 */
int roi; /* Region Of Interest: 0,1,2 */

```

```

{
    int box[8] = {412,0 , 436,0 , 436,0 , 412,0};

    box[1] = box[3] = (roi * 12) + 308; /* Set Box Y Coordinates */
    box[5] = box[7] = box[1] + 8;

    Blank(box);

    if (roi_beg[input][roi] >= 0) {
        sprintf(buffer,"%3d",roi_beg[input][roi]);
        setcolor(WHITE);
        outtextxy(412, box[1], buffer);
    }
}

/*********************************************
*                                         *
*             Draw_ROI_End()           *
*                                         *
* Draws the last channel number of the ROI. *
*                                         *
*****************************************/
void Draw_ROI_End(input, roi)

int input; /* DMR Input: 0,1,2 */
int roi;   /* Region Of Interest: 0,1,2 */

{
    int box[8] = {452,0 , 476,0 , 476,0 , 452,0};

    box[1] = box[3] = (roi * 12) + 308; /* Set Box Y Coordinates */
    box[5] = box[7] = box[1] + 8;

    Blank(box);

    if (roi_end[input][roi] >= 0) {
        sprintf(buffer,"%3d",roi_end[input][roi]);
        setcolor(WHITE);
        outtextxy(452, box[1], buffer);
    }
}

/*********************************************
*                                         *
*             Draw_ROI_Sum()          *
*                                         *
* Draws the sum of all counts in the ROI. *
*                                         *
*****************************************/

```

```
*****
void Draw_ROI_Sum(input, roi)

int input; /* DMR Input: 0,1,2 */
int roi; /* Region Of Interest: 0,1,2 */

{
    long sum=0;
    int i;
    int box[8] = {492,0 , 572,0 , 572,0 , 492,0};

    box[1] = box[3] = (roi * 12) + 308; /* Set Box Y Coordinates */
    box[5] = box[7] = box[1] + 8;

    if (roi_beg[input][roi] >= 0 &&
        roi_end[input][roi] <= roi_end[input][roi]) {
        for(i=roi_beg[input][roi]; i <= roi_end[input][roi]; i++)
            sum += counts[i];
        sprintf(buffer,"%10ld",sum);
        Blank(box);
        setcolor(WHITE);
        outtextxy(492, box[1], buffer);
    } else {
        Blank(box);
    }
}

*****
*          Draw_MCA_Screen()          *
*                                     *
* Puts up the initial display of the view spectrum screens.      *
*****
void Draw_MCA_Screen(input)

int input; /* DMR Input: 0,1,2 */

{
    int i;
    int screen_box[8] = {0,0, 639,0, 639,349, 0,349};

    setcolor(WHITE);
    setfillstyle(SOLID_FILL, BLUE);
    fillpoly(4, screen_box);

    Clear_Spec_Box(input);
}
```

```

outtextxy(4, 8, label[0]);      /* INPUT # */
sprintf(buffer,"%1d",input+1);
outtextxy(60, 8, buffer);

for (i=1; i < 10; i++)
    outtextxy(4, (i*16)+16, label[i]);

outtextxy(4, 272, label[10]);    /* CHANNEL: */
outtextxy(4, 288, label[11]);    /* COUNTS: */
outtextxy(4, 312, label[12]);    /* LIVE TIME: */
outtextxy(180, 272, label[13]);  /* VERTICAL SCALE: */
outtextxy(420, 272, label[14]);  /* HORIZONTAL SCALE: */
outtextxy(4, 332, label[15]);    /* FILE NAME: */

outtextxy(412, 296, label[16]);  /* BEG END SUM */
for (i=17; i < 20; i++)
    outtextxy(356, (i-17) * 12 + 308, label[i]);

Draw_Acquire(input);
Draw_Plot_Mode(input);

Draw_Channel(input);
Draw_Counts(input);

Draw_Time(input);

Draw_VScale(input);
Draw_HScale(input);

Draw_FName(input);

Draw_Cur_ROI(input);

for (i=0; i < MAX_ROI; i++)
    Draw_ROI_Beg(input,i);

for (i=0; i < MAX_ROI; i++)
    Draw_ROI_End(input,i);

for (i=0; i < MAX_ROI; i++)
    Draw_ROI_Sum(input,i);
}

/*********************************************
*           Draw_Spec()
* Draw the dots or lines representing the count data in the spectrum box.
*****/void Draw_Spec(input)

int input; /* DMR Input: 0,1,2 */

{
    int hsc, dot;

```

```

int start, stop, i;
int x, xstep, y;

hsc = hscale[input];
dot = (plot_mode[input] == DOT);
start = scr_beg[input];
stop = scr_end[input];
/* Transfer Variables */
/* From Arrays to Temps */
/* For Quick Access */

x = SPEC_X;
xstep = (1 << hsc);
/* Beginning X Coord */
/* How Far to Next X */

for (i=start; i < stop; i++) {
    y = Get_Plot_Y(input, i);
    /* Calc New Dot */
    if (y != plot[i]) {
        /* If Dot has Moved */
        if (dot) {
            /* DOT MODE */
            putpixel(x, plot[i], BLACK);
            plot[i]=y;
            putpixel(x, y, roi_color[i]);
        } else {
            /* FILL MODE */
            setcolor(roi_color[i]);
            line(x, y, x, plot[i]);
            plot[i]=y;
        }
        x += xstep;
    }
}

/*********************************************
*          Init_ROI()
*
* Fills the roi_color array with colors for each channel to represent the
* ROIs.
*
********************************************/
void Init_ROI(input)
int input; /* DMR Input: 0,1,2 */
{
    int i,j;

    for (j=0; j < CHANS_PER_INPUT; j++)
        roi_color[j] = YELLOW; /* Default all channels to yellow */
    for (i=0; i < MAX_ROI; i++)

```

```

    if (roi_beg[input][i] >= 0) { /* Channels in ROIs are red */
        for (j = roi_beg[input][i]; j <= roi_end[input][i]; j++)
            roi_color[j] = LIGHTRED;
    }
}

//*********************************************************************
*          View_Spectrum()
*
* This is the main routine for viewing a spectrum. It allows for changing
* the vertical or horizontal scaling, starting and stopping the acquire,
* changing from dot to fill mode, clearing the accumulated counts, and
* changing the regions of interest.
*
*****/View_Spectrum(input)

int input; /* DMR Input: 0,1,2 */

{
    int gdriver = VGA;      /* VGA Graphics Card */
    int gmode = VGAMED;   /* 640 * 350 Pixel Mode */
    int gerror;           /* Graphics Error Indicator */
    int i;                 /* Temp Integer */
    char key;              /* User Command Keypress Code */

    Fill_LogTable();           /* Initialize the Log Table */
    Init_ROI(input);          /* Initialize the ROI Colors */
    initgraph(&gdriver, &gmode, ""); /* Initialize the Graphic Driver */
    gerror = graphresult();
    if (gerror != grOk) {      /* Abort for Graphics Error */
        printf("Graphics Error");
        while (GetKey_If_Ready() == 0);
        return;
    }

    Valid_Read(input, counts); /* Read the Count data */
    Draw_MCA_Screen(input);   /* Draw the Screen */
    key = GetKey_If_Ready();
    while (key != ESC) {
        ***** Get a Valid Read of Count Data *****/
        Valid_Read(input, counts);
    }
}

```

```

***** Update the Display *****/
if (cursor[input] < scr_beg[input]      /* If moved to next screen */
|| cursor[input] >= scr_end[input])    /* Then reset the screen */
  Clear_Spec_Box(input);

Draw_Acquire(input);
Draw_Cursor(input);
Draw_Counts(input);
Draw_Time(input);
Draw_Spec(input);

if (last_sum != (int)(timer[input] / 100)) /* Once Per Second */
  last_sum = (int)(timer[input] / 100);

  if (last_sum == ACQ_TICKS / 100) { /* Make sure have latest */
    /* when acquire is done. */
    Valid_Read(input, counts);
  }

  for (i=0; i < MAX_ROI; i++)          /* Redraw all ROI SUMs */
    Draw_ROI_Sum(input,i);
}

***** Carry out user commands *****/
key = GetKey_If_Ready();

switch (key) {
  case F1:                           /* Turn Acquire On / Off */
    if (acquire[input] == 0) {
      Acquire_On(input);
    } else {
      Acquire_Off(input);
    }
    invalid[input] = 1;
    strset(spect_fname[input], ' ');
    Draw_FName(input);
    break;

  case F2:                           /* Clear Out the Count Data */
    Valid_Clr(input);
    Clear_Spec_Box(input);
    invalid[input] = 1;
    strset(spect_fname[input], ' ');
    Draw_FName(input);
    break;

  case F3:                           /* Set ROI beginning */
    roi_beg[input][cur_roi[input]] = cursor[input];
    Draw_ROI_Beg(input, cur_roi[input]);
    Init_ROI(input);
    Clear_Spec_Box(input);
    break;
}

```

```
case F4: /* Set ROI ending */
    roi_end[input][cur_roi[input]] = cursor[input];
    Draw_ROI_End(input, cur_roi[input]);
    Init_ROI(input);
    Clear_Spec_Box(input);
    break;

case F5: /* Previous ROI */
    cur_roi[input]--;
    if (cur_roi[input] < 0)
        cur_roi[input] = MAX_ROI-1;
    Draw_Cur_ROI(input);
    break;

case F6: /* Next ROI */
    cur_roi[input]++;
    if (cur_roi[input] >= MAX_ROI)
        cur_roi[input] = 0;
    Draw_Cur_ROI(input);
    break;

case F7: /* Clear the ROIs */
    Clr_ROI(input);
    for (i=0; i < MAX_ROI; i++)
        Draw_ROI_Beg(input, i);
    for (i=0; i < MAX_ROI; i++)
        Draw_ROI_End(input, i);
    for (i=0; i < MAX_ROI; i++)
        Draw_ROI_Sum(input, i);
    Init_ROI(input);
    Clear_Spec_Box(input);
    break;

case F8: /* Change the PLOT MODE */
    if (plot_mode[input] == DOT) {
        plot_mode[input] = FILL;
    } else {
        plot_mode[input] = DOT;
    }
    Clear_Spec_Box(input);
    Draw_Plot_Mode(input);
    break;

case KEY_LEFT: /* Move Cursor Left */
    if (cursor[input] > 0) {
        cursor[input]--;
    }
    Draw_Channel(input);
    break;

case KEY_RIGHT: /* Move Cursor Right */
    if (cursor[input] < (CHANS_PER_INPUT-1)) {
        cursor[input]++;
    }
    Draw_Channel(input);
    break;
```

```

case PGDN:           /* Move Cursor Left 16 */
    cursor[input] -= 16;
    if (cursor[input] < 0) {
        cursor[input] = 0;
    }
    Draw_Channel(input);
    break;

case PGUP:           /* Move Cursor Right 16 */
    cursor[input] += 16;
    if (cursor[input] > CHANS_PER_INPUT-1) {
        cursor[input] = CHANS_PER_INPUT-1;
    }
    Draw_Channel(input);
    break;

case HOME:           /* Move Cursor to Channel 0 */
    cursor[input] = 0;
    Draw_Channel(input);
    break;

case END:             /* Move Cursor to Channel 511 */
    cursor[input] = CHANS_PER_INPUT-1;
    Draw_Channel(input);
    break;

case KEY_DOWN:        /* Increase Vertical Scale */
    if (vscale[input] < 16) {
        vscale[input]++;
        Draw_VScale(input);
        Clear_Spec_Box(input);
    }
    break;

case KEY_UP:          /* Decrease Vertical Scale */
    if (vscale[input] > -1) {
        vscale[input]--;
        Draw_VScale(input);
        Clear_Spec_Box(input);
    }
    break;

case PLUS:            /* Increase Horizontal Scale */
case PLUS_KP:
    if (hscale[input] < 3) {
        hscale[input]++;
        Draw_HScale(input);
        Clear_Spec_Box(input);
    }
    break;

case MINUS:           /* Decrease Horizontal Scale */
case MINUS_KP:
    if (hscale[input] > 0) {
        hscale[input]--;
        Draw_HScale(input);
        Clear_Spec_Box(input);
    }
    break;

```

```

        }
    break;
}
closegraph();
return;
}

/*********************+
*          Make_Prn_Num()
* Converts the long int to a string. Gives K or M ending if possible.
*
+********************/
void Make_Prn_Num(str, num)
char *str; /* String to hold result */
long num; /* Number to convert */
{
    sprintf(str,"%ld",num);
    if (num != 0L && num % 1024L == 0L)
        sprintf(str,"%3ldK",(num >> 10));
    if (num != 0L && num % 1048576L == 0L)
        sprintf(str,"%3ldM",(num >> 20));
}

/*********************+
*          Get_LJ_Plot()
* Returns the Y Coordinate of a channel count adjusted for the LaserJet.
*
+********************/
int Get_LJ_Plot(input, chan)
int input; /* DMR Input: 0,1,2 */
int chan; /* Channel Number */
{
    int v;
    int y;
    int i;
    long l;

    v = vscale[input];
    l = counts[chan];
}

```

```

if (v < 0) /* LOG SCALE */
{
    y=0;                                /** Log Scale is Broken up into 6 **/
                                                /** Zones of 341 Pixels Each **/
    i = (int)l;
    if (i & 0xf) {                      /** Zone 0: 1 - 16 **/
        v = (int)(l<<4) & 0xff;
        y = logtable[v] + 0*340;
    }
    if (i & 0xf0) {                     /** Zone 1: 16 - 256 **/
        v = (int)l & 0xff;
        y = logtable[v] + 1*340;
    }
    if (i & 0xf00) {                    /** Zone 2: 256 - 4K **/
        v = (int)(l>>4) & 0xff;
        y = logtable[v] + 2*340;
    }
    if (i & 0xf000L) {                  /** Zone 3: 4K - 64K **/
        v = (int)(l>>8) & 0xff;
        y = logtable[v] + 3*340;
    }
    i = (int)(l>>16);
    if (i & 0xf) {                     /** Zone 4: 64K - 1M **/
        v = (int)(l>>12) & 0xff;
        y = logtable[v] + 4*340;
    }
    if (i & 0xf0) {                   /** Zone 5: 1M - 16M **/
        v = (int)(l>>16) & 0xff;
        y = logtable[v] + 5*340;
    }
} else /* BINARY SCALE */
{
    if (v >= 3) {
        y = (int)((l >> (v-3)) & 0x7ffL);
    } else {
        y = (int)((l << (3-v)) & 0x7ffL);
    }
}
return (2208-y);
}

```

---

```

*****
*          LJ_Line()
*
* Draws a vertical line to the LaserJet printer.
*
```

```

/*
*****Prn_Spec_LJ*****
void LJ_Line(prn,x,y,len)
FILE *prn; /* Printer Stream */
int x; /* X Coordinate for Start of Line */
int y; /* Y Coordinate for Start of Line */
int len; /* Length of line */

{
    int i;
    int nbytes;
    char lastbyte;

    char bytes[8] = { 0, 128, 192, 224, 240, 248, 252, 254 };

    nbytes = len / 8;
    lastbyte = bytes[len - nbytes*8];

    fprintf(prn,"%c*p%dx",27,x); /* Cursor X */
    fprintf(prn,"%c*p%dy",27,y); /* Cursor Y */

    fprintf(prn,"%c*r1A",27); /* Start Gfx at Cursor */
    fprintf(prn,"%c*b%dw",27,nbytes+1); /* Start Byte Transmission */

    for (i=0; i < nbytes; i++)
        fprintf(prn,"%c",255); /* Transmit Bytes */
    fprintf(prn,"%c",lastbyte);

    fprintf(prn,"%c*rB",27); /* End Graphics */
}

/*
*****Prn_Spec_LJ*****
* Prints the spectrum to the LaserJet printer.
*/
Prn_Spec_LJ(input)

int input; /* DMR Input: 0,1,2 */

{
    FILE *prn;
    int i,j;
    int x,y,h,xstep;
    long l,m;
    long sum;

    double temp;
    char tempstr[12];
}

```

```

/*** Initialize the ROI Plot array ***/
Init_ROI(input);

/*** Fill in the Laser-Jet Log Table ***/
for (i=0; i < 16; i++)
    logtable[i] = 0;
for (i=16; i < 256; i++) {
    temp = log(i / 16.0);          /* Divide by 16 to Step Slower */
    temp = temp / log(256 / 16.0); /* Range from 0 to 1 */
    temp = temp * 340.0;           /* Range from 0 to 340 */
    temp = floor(temp + 0.5);     /* Round Up */
    logtable[i] = (int)temp;        /* Put it into Table */
}

/*** Open the Stream to Printer ***/
prn = fopen("PRN","wb");
if (prn == NULL) {
    return 1;
} else {
    /*** Put Printer into Proper Mode ***/
    fprintf(prn,"%cE",27);        /* Reset */
    fprintf(prn,"%c*t75P",27);    /* 75 dpi Graphics */
    fprintf(prn,"%c&l10",27);    /* Landscape */
    fprintf(prn,"%c&k2S",27);    /* 16.66 Pitch */
    fprintf(prn,"%c*rB",27);      /* End Graphics */

    /*** Left Line ***/
    LJ_Line(prn, 100, 160, 515);

    /*** Right Line ***/
    LJ_Line(prn, 3178, 160, 515);

    /*** Top and Bottom Lines ***/
    for (x=102; x <= 3178; x+=4) {
        fprintf(prn,"%c*p%dX",27,x); /* Cursor X */
        fprintf(prn,"%c*p160Y",27);   /* Cursor Y */
    }
}

```

```

fprintf(prn,"%c*r1A",27); /* Start Gfx at Cursor */
fprintf(prn,"%c*b1W%c",27,128); /* Byte Transmission */
fprintf(prn,"%c*rB",27); /* End Graphics */

fprintf(prn,"%c*p%dx",27,x); /* Cursor X */
fprintf(prn,"%c*p2216y",27); /* Cursor Y */
fprintf(prn,"%c*r1A",27); /* Start Gfx at Cursor */
fprintf(prn,"%c*b1W%c",27,128); /* Byte Transmission */
fprintf(prn,"%c*rB",27); /* End Graphics */
}

/** Dots **/

x = 106; /* Beginning X Coord */
xstep = (6 << hscale[input]); /* How Far to Next X */

for (i=scr_beg[input]; i < scr_end[input]; i++) {
    y = Get_LJ_Plot(input, i);

    if (roi_color[i] == YELLOW) {
        LJ_Line(prn, x, y, 1); /* Dot */
    } else {
        LJ_Line(prn, x, y, (2216+3-y)/4); /* Fill in ROI */
    }

    x += xstep;
}

/** Cursor **/

x = 106 + (cursor[input] - scr_beg[input]) * (6 << hscale[input]);
y = Get_LJ_Plot(input, cursor[input]) - 64;
h = 15;

if (y < 160) {
    h = (y - (160 - 60) + 3) / 4; /* Adjust Length */
    y = 160; /* Clip at Top */
}

LJ_Line(prn, x, y, h); /* Draw Cursor Line */

/** Channel Markers **/

i = scr_beg[input]; /* First Channel Number */
j = (32>> hscale[input]); /* # Channels in 32 Steps */

for (x=106; x < 3178; x+=32*6) {
    LJ_Line(prn, x, 2220, 3); /* Draw Tick Below Box */

    h = x - 27;
    if (i < 100)
}

```

```

    h = x - 18;
    if (i < 10)
        h = x - 9;
    fprintf(prn,"%c*p%dx",27,h);      /* Cursor X */
    fprintf(prn,"%c*p2260Y",27);      /* Cursor Y */
    fprintf(prn,"%d",i);              /* Print Channel Number */

    i += j;                          /* Next Channel Number */
}

/** Vertical Markers **/
if (vscale[input] >= 0) {           /* Binary Scale Markers */
    l = 0L;                         /* First Vertical Number */
    m = (16L << vscale[input]);    /* # Counts in 16 Steps */

    for (y=2208; y > 160; y-=16*8) {
        LJ_Line(prn, 100-4, y, 1);   /* Draw Tick Mark Left of Box */
        LJ_Line(prn, 100-8, y, 1);
        Make_Prn_Num(tempstr, l);   /* Make Vertical Number */
        fprintf(prn,"%c*p0X",27);    /* Cursor X */
        fprintf(prn,"%c*p%dy",27,y+12); /* Cursor Y */
        fprintf(prn,"%4s",tempstr);   /* Print Vertical Number */

        l += m;                      /* Next Vertical Number */
    }
} else {                           /* Log Scale Markers */
    l = 1L;                         /* First Vertical Number */
    for (y=2208; y > 168; y-=340) {
        Make_Prn_Num(tempstr, l);   /* Make Vertical Number */
        fprintf(prn,"%c*p0X",27);    /* Cursor X */
        fprintf(prn,"%c*p%dy",27,y+12); /* Cursor Y */
        fprintf(prn,"%4s",tempstr);   /* Print Vertical Number */

        l <<= 4;                    /* Next Vertical Number */

        for (i=16; i < 256; i+=16) {
            h = logtable[i];
            LJ_Line(prn, 100-4, y-h, 1); /* Draw Tick Mark Left of Box */
            LJ_Line(prn, 100-8, y-h, 1);
        }
    }
}

/** File Name ***/

```

```

fprintf(prn,"%c&a0R",27);           /* Set Cursor */
fprintf(prn,"%c&a5C",27);           /* Print Spectrum File Name */
fprintf(prn,"File Name: %s",spect_fname[input]);

/** Channel and Counts **/
fprintf(prn,"%c&a2R",27);
fprintf(prn,"%c&a5C",27);
fprintf(prn,"Channel: %d Counts: %ld",
cursor[input], counts[cursor[input]] & 0xffffffff);

/** ROIs **/
for (i=0; i<MAX_ROI; i++) {
    fprintf(prn,"%c&a%dr",27,i);      /* Cursor Row */
    fprintf(prn,"%c&a60c",27);        /* Cursor Column */

    if (roi_beg[input][i] >= 0 &&
        roi_beg[input][i] <= roi_end[input][i]) {

        sum = 0L;
        for(j=roi_beg[input][i]; j <= roi_end[input][i]; j++)
            sum += counts[j];

        fprintf(prn,"ROI%1d: %3d %3d %9ld",
               i+1, roi_beg[input][i], roi_end[input][i], sum);
    } else {
        fprintf(prn,"ROI%1d:",i+1);
    }
}

/** Vertical Scale **/
fprintf(prn,"%c&a0R",27);
fprintf(prn,"%c&a110c",27);
fprintf(prn,"Vertical Scale: %s", vsc_label[vscale[input]+1]);

/** Horizontal Scale **/
fprintf(prn,"%c&a2R",27);
fprintf(prn,"%c&a110c",27);
fprintf(prn,"Horizontal Scale: %ld", (1 << hscale[input]));

/** Reset Printer **/
fprintf(prn,"%cE",27);
fclose(prn);

```

```
        return 0;
    }

/*************************************************************************
 *          Prn_Spec_TI()
 *          Prints the spectrum to the Texas Instruments Model 855 printer.
 *************************************************************************/
Prn_Spec_TI(input)
int input; /* DMR Input: 0,1,2 */
{
    FILE *prn;
    int h,i,j,k;
    int y;
    int curbot, curtop;
    long l,m;
    long sum;
    char tempstr[12];
    char log_tick[32];

    /*** Initialize the ROI Plot array ***/
    Init_ROI(input);

    /*** Set Up for Log Scale ***/
    Fill_LogTable();           /* Fill in the Log Table */
    for (i=0; i<32; i++)      /* Clear the Log Scale Ticks */
        log_tick[i]=0;
    for (i=0; i < 42*6; i+=42) {
        for (j=16; j < 256; j+=16) {
            h = 255-(i+logtable[j]);
            log_tick[(h>>3)] |= (128 >> (h&7)); /* Set Bits for Log Ticks */
        }
    }
}
```

```

/** Set up the Count Plot Data ***/
for (i=0; i < CHANS_PER_INPUT; i++) /* Get Dot Y-Coordinate */
    plot[i] = Get_Plot_Y(input, i) - SPEC_Y; /* Remove Screen Adjustment */

/** Cursor Coordinates ***/
curtop = plot[cursor[input]] - 10;
curbot = plot[cursor[input]] - 2;

/** Open the Stream to Printer ***/
prn = fopen("PRN","wb");
if (prn == NULL) {
    return 1;
} else {
    /** Setup Printer Modes **/
    fprintf(prn,"%ca",27);           /* Reset to DP Mode */
    fprintf(prn,"%cu",27);           /* Unidirectional Printing */
    fprintf(prn,"%cz",27);           /* Set 12 CPI Print */
    fprintf(prn,"%c0",27);           /* 8 Lines / Inch */

    /** File Name **/
    fprintf(prn,"File Name: %s\n\n",spect_fname[input]);

    /** Channel and Counts **/
    fprintf(prn,"Channel: %d Counts: %ld\n\n",
cursor[input], counts[cursor[input]] & 0xffff);

    /** ROIs **/
    for (i=0; i<MAX_ROI; i++) {
        if (roi_beg[input][i] >= 0 &&
            roi_beg[input][i] <= roi_end[input][i]) {
            sum = 0L;
            for(j=roi_beg[input][i]; j <= roi_end[input][i]; j++)
                sum += counts[j];
            fprintf(prn,"ROI%1d: %3d %3d %9ld\n",
i+1, roi_beg[input][i], roi_end[input][i], sum);
        } else {
}
}

```

```

        fprintf(prn,"ROI%1d:\n",i+1);
    }

    /*** Vertical Scale ***/
    fprintf(prn,"\nVertical Scale: %s  ", vsc_label[vscale[input]+1]);

    /*** Horizontal Scale ***/
    fprintf(prn,"Horizontal Scale: %1d(\n\n",(1 << hscale[input]));

    fprintf(prn,"%cA%c",27,8);           /* Set Vertical Spacing */

    /*** Top Edge of Box ***/
    fprintf(prn,"%cH%c%c",27,5,2);      /* 6 Spaces, 517 Graphics */

    for (i=0; i < 3; i++)
        fputc(0,prn);
    for (i=0; i < 514; i++)
        fputc(1,prn);

    fputc('\n',prn);

    /*** Middle of Box ***/
    if (vscale[input] >= 0) {
        l = (256L << vscale[input]);          /* BINARY SCALE */
        m = (16L << vscale[input]);           /* First Vertical Number */
        /* # Counts in 16 Steps */
    } else {
        l = (1L << 24);                     /* LOG SCALE */
    }
    y=0;                                     /* Y Coordinate */

    for (i=0; i < 32; i++) {
        /*** Vertical Scale Markers ***/
        if (vscale[input] >= 0) {             /* BINARY SCALE MARKERS */
            if (i & 1) {
                l -= m;                      /* Next Vertical Number */
                Make_Prn_Num(tempstr, l);     /* Make Vertical Number */
                fprintf(prn,"%5s ",tempstr);  /* Print Vertical Number */
            } else {

```

```

        fprintf(prn,"      ");
    }

    fprintf(prn,"%c%c%c",27,5,2);

    if (i & 1) {
        for (j=0; j < 3; j++)
            fputc(1,prn); /* Tick Mark */
    } else {
        for (j=0; j < 3; j++)
            fputc(0,prn); /* No Tick Mark */
    }

} else { /* LOG SCALE MARKERS */

    if ((i<20 && i%5 == 0)|| (i>20 && (i-1)%5 == 0)) {
        Make_Prn_Num(tempstr, l); /* Make Vertical Number */
        fprintf(prn,"%5s ", tempstr); /* Print Vertical Number */
        l >>= 4; /* Next Vertical Number */
    } else {
        fprintf(prn,"      "); /* No Vertical Number */
    }

    fprintf(prn,"%c%c%c",27,5,2);
    for (j=0; j < 3; j++)
        fputc(log_tick[i],prn); /* Tick Marks */
}

/** Left Edge of Box ***/

fputc(255,prn);

/** Dots, Lines, and Cursor ***/

for (j=scr_beg[input]; j < scr_end[input]; j++) {
    if (roi_color[j] == YELLOW) {
        if (plot[j] >= y && plot[j] < (y+8)) {
            h = (128 >> (plot[j]-y)); /* Dot */
        } else {
            h = 0; /* No Dot */
        }
    } else {
        if (plot[j] < y) {
            h = 255; /* Full Line */
        } else {
    }
}

```

```

        if (plot[j] >= y && plot[j] < (y+8)) {
            h = (255 >> (plot[j]-y)); /* Partial Line */
        } else {
            h = 0; /* No Line */
        }
    }

    if (cursor[input] == j) { /* Add in Cursor */
        if (curtop >= y && curtop < (y+8))
            h |= (255 >> (curtop-y));
        if (curbot >= y && curbot < (y+8))
            h |= (255 << (y+7-curbot));
    }

    fputc(h, prn); /* Print Character */

    /*** Blanks between Channels for Horizontal Scaling ***/
    for (k=0; k < ((1<<hscale[input])-1); k++)
        fputc(0,prn);
}

/*** Right Edge of Box ***/
fputc(255,prn);
fputc('\n',prn); /* Line Feed */
y += 8; /* Move Y Coordinate Down */
}

/*** Bottom of Box ***/
fprintf(prn," %c%c%c",27,5,2);
for (i=0; i < 3; i++)
    fputc(0,prn);

fputc(128,prn); /* Bottom Left Corner */
for (i=0; i < 16; i++) {
    fputc(240,prn); /* Tick Mark */
    for (j=0; j < 31; j++)
        fputc(128,prn); /* Bottom Edge */
}
fputc(128,prn); /* Bottom Right Corner */

fprintf(prn,"\n      ");

/*** Channel Markers ***/
j = scr_beg[input]; /* First Channel Number */
j = (32->> hscale[input]); /* # Channels in 32 Steps */

```

```
for (k=0; k < 16; k++) {
    fprintf(prn,"%cN%c",27,2);           /* Graphics Characters for */
    for (h=0; h < 3; h++)
        fputc(0,prn);                  /* Extra Spacing */

    fprintf(prn,"%3d ",i);              /* Print Channel Number */
    i += j;                           /* Next Channel Number */

    fprintf(prn,"%c2",27);             /* 6 Lines per Inch */
    fprintf(prn,"%c",12);              /* Form Feed */

    fclose(prn);
    return 0;
}
```

**APPENDIX B**

**DERIVATION OF THE THREE-ROI METHOD**

**FOR ESTIMATING CONCENTRATION OF  $^{226}\text{Ra}$**



## DERIVATION OF THE THREE-ROI METHOD FOR ESTIMATING CONCENTRATION OF $^{226}\text{Ra}$

First implemented by Doane and others of Oak Ridge National Laboratory (1984), the three-ROI method of estimating the concentration of  $^{226}\text{Ra}$  in soil samples uses gamma spectra of  $^{40}\text{K}$ ,  $^{226}\text{Ra}$ , and  $^{232}\text{Th}$  (KRT). Arthur S. Rood of ORNL later modified the software, creating a program called KRTCOM to run on an IBM® personal computer. The following explanation is based on an unpublished paper documenting the three-ROI method (Rood 1986), and no credit is claimed for the author of this memorandum.

The three-ROI method is based on the principle that a linear relationship exists between the amount of a radionuclide in a sample and the number of counts from the sample observed in a gamma detection system. Amount is here defined as the concentration of the radionuclide in the sample, expressed as units per weight, multiplied by the weight of the sample multiplied by the observed counting time. The relationship between the observed counts and the amount of the radionuclide can be expressed as:

$$N = AC$$

where

$N$  is the number of observed counts in a region of interest in the gamma spectrum,

$A$  is a proportionality constant (dictated by the relationship), and

$C$  is the amount of the radionuclide in the sample.

For this relationship to hold true, the following assumptions about the sample detection system are necessary:

- 1) All samples have the same geometry (size and shape), and
- 2) All samples are observed at the same distance from the detector.

Using the observed counts from a known amount of the radionuclide, the proportionality constant can be calculated:

$$A = NC^{-1}$$

where

$N$  is the number of observed counts in a region of interest in the gamma spectrum,

$A$  is the proportionality constant, and

$C^{-1}$  is the inverse of the known amount of the radionuclide.

Once the proportionality constant is known for a counting system, an unknown amount of the radionuclide can be calculated by:

$$C = NA^{-1}$$

where

$C$  is the amount of the radionuclide in the sample,

$N$  is the number of observed counts in a region of interest in the gamma spectrum, and

$A^{-1}$  is the inverse of the proportionality constant.

The relationship holds true for a sample material that contains only one radionuclide. When a sample material contains several radionuclides, it is possible that each of the radionuclides will contribute counts to the region of interest. To account for several radionuclides, the relationship between observed counts and concentrations can be expressed as:

$$N = A_1 C_1 + A_2 C_2 + A_3 C_3 + \dots + A_n C_n$$

where

$N$  is the number of observed counts in a region of interest in the gamma spectrum,

$C_x$  is the amount of the radionuclide present in the sample, and

$A_x$  is the proportionality constant corresponding to the radionuclide.

For the KRT method, three different ROIs are observed. Each ROI is chosen to contain a characteristic peak in the gamma spectrum for one of the radionuclides. The gamma energy spectrum is divided into 512 channels (numbered 0 to 511), with each channel representing 6 KeV in the energy spectrum. The ROI chosen for  $^{226}\text{Ra}$  (referred to as ROI<sub>1</sub>) is channels 273 through 412 (1638 to 2472 KeV). The ROI for  $^{40}\text{K}$  (referred to as ROI<sub>2</sub>) is channels 226 through 259 (1356 to 1554 KeV). The ROI for  $^{232}\text{Th}$  (referred to as ROI<sub>3</sub>) is channels 426 through 511 (2556 to 3066 KeV).

For the KRT method, three different reference materials are used to establish the proportionality constants between the counts and the amounts of the radionuclide. Each of the three reference materials is composed mainly of one of the radionuclides used. Reference material one is composed mainly of radium, reference material two is composed mainly of potassium, and reference material three is composed mainly of thorium. (It should be noted that traces of the other two radionuclides are present in all three of the reference materials.) The concentrations of the three radionuclides in each of the three reference materials are known, as are the weights of the reference materials.

Because counts from three radionuclides are observed in three regions of interest from three reference materials, the relationship between counts and amounts of the radionuclides can best be described in terms of matrices:

$$\text{ref} = \text{pro} * \text{con}^{-1}$$

where

$$\text{ref} = \begin{bmatrix} \text{ref}_{11} & \text{ref}_{12} & \text{ref}_{13} \\ \text{ref}_{21} & \text{ref}_{22} & \text{ref}_{23} \\ \text{ref}_{31} & \text{ref}_{32} & \text{ref}_{33} \end{bmatrix}, \quad \text{con} = \begin{bmatrix} \text{con}_{11} & \text{con}_{12} & \text{con}_{13} \\ \text{con}_{21} & \text{con}_{22} & \text{con}_{23} \\ \text{con}_{31} & \text{con}_{32} & \text{con}_{33} \end{bmatrix},$$

and

*pro* is a three by three matrix representing the proportionality constants,  
 $\text{ref}_{11}$  = Counts in ROI<sub>1</sub> from reference 1 (radium),  
 $\text{ref}_{21}$  = Counts in ROI<sub>2</sub> from reference 1 (radium),

$ref_{31}$  = Counts in ROI<sub>3</sub> from reference 1 (radium),  
 $ref_{12}$  = Counts in ROI<sub>1</sub> from reference 2 (potassium),  
 $ref_{22}$  = Counts in ROI<sub>2</sub> from reference 2 (potassium),  
 $ref_{32}$  = Counts in ROI<sub>3</sub> from reference 2 (potassium),  
 $ref_{13}$  = Counts in ROI<sub>1</sub> from reference 3 (thorium),  
 $ref_{23}$  = Counts in ROI<sub>2</sub> from reference 3 (thorium),  
 $ref_{33}$  = Counts in ROI<sub>3</sub> from reference 3 (thorium),  
 (Net counts, background removed)  
 $con_{11}$  = Concentration of radium in reference 1  $\times$  weight of reference  
 1  $\times$  time,  
 $con_{21}$  = Concentration of potassium in reference 1  $\times$  weight of reference  
 1  $\times$  time,  
 $con_{31}$  = Concentration of thorium in reference 1  $\times$  weight of reference  
 1  $\times$  time,  
 $con_{12}$  = Concentration of radium in reference 2  $\times$  weight of reference  
 2  $\times$  time,  
 $con_{22}$  = Concentration of potassium in reference 2  $\times$  weight of reference  
 2  $\times$  time,  
 $con_{32}$  = Concentration of thorium in reference 2  $\times$  weight of reference  
 2  $\times$  time,  
 $con_{13}$  = Concentration of radium in reference 3  $\times$  weight of reference  
 3  $\times$  time,  
 $con_{23}$  = Concentration of potassium in reference 3  $\times$  weight of reference  
 3  $\times$  time,  
 $con_{33}$  = Concentration of thorium in reference 3  $\times$  weight of reference  
 3  $\times$  time.

The matrix  $pro$  can be determined by:

$$pro = ref * con^{-1}$$

where  $con^{-1}$  is the inverse of matrix  $con$ .

Now that the proportionality constant matrix has been determined, the

B-5

amounts of the radionuclides can be determined from counts in the ROIs of a gamma energy spectrum obtained from a sample using the formula:

$$result = pro^{-1} * samp$$

where

$$result = \begin{bmatrix} result_1 \\ result_2 \\ result_3 \end{bmatrix}, \quad samp = \begin{bmatrix} samp_1 \\ samp_2 \\ samp_3 \end{bmatrix},$$

and

$pro^{-1}$  is the inverse of matrix  $pro$ ,

$result_1$  = Concentration of radium in sample  $\times$  weight of sample  $\times$  time,

$result_2$  = Concentration of potassium in sample  $\times$  weight of sample  $\times$  time,

$result_3$  = Concentration of thorium in sample  $\times$  weight of sample  $\times$  time,

$samp_1$  = Counts in ROI<sub>1</sub> of sample,

$samp_2$  = Counts in ROI<sub>2</sub> of sample, and

$samp_3$  = Counts in ROI<sub>3</sub> of sample.

To determine the concentrations of each of the radionuclides in the sample, each  $result_x$  must be divided by sample weight and time.

**APPENDIX C**  
**ALGORITHMIC DEFINITIONS**  
**OF MATRIX OPERATIONS**



## ALGORITHMIC DEFINITIONS OF MATRIX OPERATIONS

The following are algorithmic definitions of matrix operations. These definitions are the basis for the computer algorithms used to manipulate matrices in the gamma analysis program.

### **Matrix Addition:**

The sum of matrices ( $C = A + B$ ) may be found by adding the individual elements of two matrices. In general:

$$C_{ij} = A_{ij} + B_{ij}$$

where  $A$  and  $B$  are matrices of identical size.

### **Matrix Products:**

The product of two matrices ( $C = AB$ ) may be found by summing the products of each element in the row of the first matrix and each element in the column of the second matrix. In general:

$$C_{ij} = \sum_{k=1}^n A_{ik} * B_{kj}$$

where  $A$  is a matrix with  $n$  columns and  $B$  is a matrix with  $n$  rows.

### **Scalar Multiples:**

A matrix may be scaled ( $B = cA$ ) by multiplying each of its elements by a scalar. In general:

$$B_{ij} = c * A_{ij}$$

where

$A$  and  $B$  are matrices and  $c$  is a scalar.

### Matrix Transpose:

The transpose of a matrix ( $B=A^T$ ) can be found by replacing the row entries of a matrix by its column entries. In general:

$$B_{ij} = A_{ji}$$

where  $A$  is a matrix of size  $m,n$  and  $B$  is a matrix of size  $n,m$ .

### Inverse Matrix:

The inverse of a matrix ( $B=A^{-1}$ ) can be found by scaling the adjoint of the matrix with the reciprocal of the determinant of the matrix. In general:

$$B = \frac{1}{\det A} \text{adj} A$$

where  $\det A$  is the determinant of matrix  $A$  and  $\text{adj} A$  is the adjoint of matrix  $A$ .

### Adjoint of a Matrix:

The adjoint of a matrix ( $B=\text{adj} A$ ) can be found by replacing each element by its cofactor and then transposing the matrix (Boas 1983). In general:

$$B_{ij} = \text{cof} A_{ji}$$

where  $A$  is a square matrix and  $\text{cof} A_{ji}$  is the cofactor of the element in the  $j^{th}$  row and the  $i^{th}$  column of matrix  $A$  (Shields 1981). The cofactor in general is:

$$\text{cof} A_{ij} = (-1)^{i+j} \det S$$

where  $S$  is the submatrix found by deleting the  $i^{th}$  row and the  $j^{th}$  column from matrix  $A$  (Boas 1983).

**Determinate of a Matrix:**

The determinate of a matrix is a number, and can be found by the recursive definition:

$$\det A = \sum_{i=1}^n (-1)^{i+1} A_{ii} \det S_{ii}$$

where  $A$  is a square matrix of size  $n$  and  $S_{ii}$  is the submatrix found by deleting the  $i^{th}$  row and the  $1^{st}$  column of  $A$  (Shields 1981). The determinant of a matrix of size 1 is the element itself.

**APPENDIX D**  
**CONTENT DESCRIPTIONS OF FILES USED**



**CONTENT DESCRIPTIONS OF FILES USED****Spectrum File**

Each spectrum file contains the information for a single gamma energy spectrum. This information is stored in the format specified by Nucleus, Inc., manufacturer of the PCAT™ and DMR™ hardware. The header file containing the C language description of a spectrum file is DMR.H (Appendix A). A semantic description of a spectrum file is:

**File Header:**

Consists of 512 bytes of data describing various attributes of the spectrum. All these attributes are define in the structure called DMRHEADER. The program specifications of this function are in the source code file called PCA.C (see Appendix A). The "SpSave" function sets all header variables to zero except for the following:

ltimeflag	Live Time Flag is set to -1 to indicate that the spectrum was collected in live time mode.
convergain	Conversion Gain is set to 4 to indicate a conversion gain of 512 channels.
group	Group Number is set to 8.
phemode	PHA Mode Flag is set to -1 to indicate that the spectrum was collected in pulse height analysis mode.
numchanspm	Number of Channels per Spectrum is set to 512.
majvers	Major Version number is set to 2 to indicate spectrum is compatible with DMR™ software Version 2.
minvers	Minor Version number is set to 10 to indicate spectrum is compatible with DMR™ software Version 2.10.
endheader	End Header is set to 210 to indicate spectrum is compatible with DMR™ software Version 2.10.

idcodestr	ID Code String is set to the spectrum file name (without the drive specification).
pca_date	PCA™ Date is set to the date the spectrum file is saved. Format: <b>MMM DD YYYY.</b>
pca_time	PCA™ Time is set to the time the spectrum file is saved. Format: <b>HH:MM:SS xm.</b>

**Count Data:**

There are 512 channel counts, consisting of four bytes each (a long integer, in C terminology). The most significant byte of each channel count is used as a flag to indicate ROIs. The least three significant bytes of each channel count contains the count data: a number from 0 to 16,777,215. The 4 bytes are arranged in the file as follows:

- Byte 0: least significant byte of count data
- Byte 1: middle byte of count data
- Byte 2: most significant byte of count data
- Byte 3: ROI byte: (0 for no ROI, 1 for ROI #1, etc.)

**File Footer:**

PCA™-compatible spectra files may contain footer data that describes start and stop dates for spectra files that contain multiple spectra. The spectra files that the SpSave function writes do not contain any footer data.

The total length of the spectra files written by the SpSave function is 2560 bytes. This consists of 512 bytes for the header, and 2048 bytes for the count (and ROI) data ( $512 \times 4$ ).

**Reference File**

The reference file contains the contents of 3 matrices used in the concentration estimation algorithm. The first matrix (called *ref* in the program)

contains the net counts (background removed) in the ROIs from the reference standards. The second matrix (called *con* in the program) holds the known amounts of the reference radionuclides in the reference standards. The amount is the concentration of the radionuclide multiplied by the weight of the reference material multiplied by the observed counting time. The third matrix (called *bkg* in the program) contains the background counts in each of the ROIs. These matrices can be described as follows:

$$\text{ref} = \begin{bmatrix} \text{ref}_{11} & \text{ref}_{12} & \text{ref}_{13} \\ \text{ref}_{21} & \text{ref}_{22} & \text{ref}_{23} \\ \text{ref}_{31} & \text{ref}_{32} & \text{ref}_{33} \end{bmatrix},$$

where

- $\text{ref}_{11}$  = Counts in ROI<sub>1</sub> from reference 1 (radium),
  - $\text{ref}_{21}$  = Counts in ROI<sub>2</sub> from reference 1 (radium),
  - $\text{ref}_{31}$  = Counts in ROI<sub>3</sub> from reference 1 (radium),
  - $\text{ref}_{12}$  = Counts in ROI<sub>1</sub> from reference 2 (potassium),
  - $\text{ref}_{22}$  = Counts in ROI<sub>2</sub> from reference 2 (potassium),
  - $\text{ref}_{32}$  = Counts in ROI<sub>3</sub> from reference 2 (potassium),
  - $\text{ref}_{13}$  = Counts in ROI<sub>1</sub> from reference 3 (thorium),
  - $\text{ref}_{23}$  = Counts in ROI<sub>2</sub> from reference 3 (thorium),
  - $\text{ref}_{33}$  = Counts in ROI<sub>3</sub> from reference 3 (thorium).
- (net counts, background removed)

$$\text{con} = \begin{bmatrix} \text{con}_{11} & \text{con}_{12} & \text{con}_{13} \\ \text{con}_{21} & \text{con}_{22} & \text{con}_{23} \\ \text{con}_{31} & \text{con}_{32} & \text{con}_{33} \end{bmatrix},$$

where

- $\text{con}_{11}$  = Concentration of radium in reference 1  $\times$  weight of reference 1  $\times$  time,

$con_{21}$  = Concentration of potassium in reference 1  $\times$  weight of reference  
 1  $\times$  time,

$con_{31}$  = Concentration of thorium in reference 1  $\times$  weight of reference  
 1  $\times$  time,

$con_{12}$  = Concentration of radium in reference 2  $\times$  weight of reference  
 2  $\times$  time,

$con_{22}$  = Concentration of potassium in reference 2  $\times$  weight of reference  
 2  $\times$  time,

$con_{32}$  = Concentration of thorium in reference 2  $\times$  weight of reference  
 2  $\times$  time,

$con_{13}$  = Concentration of radium in reference 3  $\times$  weight of reference  
 3  $\times$  time,

$con_{23}$  = Concentration of potassium in reference 3  $\times$  weight of reference  
 3  $\times$  time,

$con_{33}$  = Concentration of thorium in reference 3  $\times$  weight of reference  
 3  $\times$  time.

$$bkg = \begin{bmatrix} bkg_1 \\ bkg_2 \\ bkg_{31} \end{bmatrix},$$

where

$bkg_1$  = Background counts in ROI<sub>1</sub>,

$bkg_2$  = Background counts in ROI<sub>2</sub>,

$bkg_3$  = Background counts in ROI<sub>3</sub>.

Matrices  $ref$ ,  $con$ , and  $bkg$  are actually arrays of 3 matrices, one from each of the detector inputs. In total there are 63 numbers. The reference file is an ASCII text file that contains one number per line arranged in the following order:

$ref_{11}$  (Input Number One)

$ref_{12}$

*ref<sub>13</sub>*

*ref<sub>21</sub>*

*ref<sub>22</sub>*

*ref<sub>23</sub>*

*ref<sub>31</sub>*

*ref<sub>32</sub>*

*ref<sub>33</sub>*

*ref<sub>11</sub>* (Input Number Two)

.

.

*ref<sub>33</sub>*

*ref<sub>11</sub>* (Input Number Three)

.

.

*ref<sub>33</sub>*

*con<sub>11</sub>* (Input Number One)

*con<sub>12</sub>*

*con<sub>13</sub>*

*con<sub>21</sub>*

*con<sub>22</sub>*

*con<sub>23</sub>*

*con<sub>31</sub>*

*con<sub>32</sub>*

*con<sub>33</sub>*

*con<sub>11</sub>* (Input Number Two)

.

.

*con<sub>33</sub>*

*con<sub>11</sub>* (Input Number Three)

.

.

*con<sub>33</sub>*

*bkg<sub>1</sub>* (Input Number One)

*bkg<sub>2</sub>*

*bkg<sub>3</sub>*

*bkg<sub>1</sub>* (Input Number Two)

*bkg<sub>2</sub>*

*bkg<sub>3</sub>*

*bkg<sub>1</sub>* (Input Number Three)

*bkg<sub>2</sub>*

*bkg<sub>3</sub>*

## **ERFILE**

This table stores data for the determination of the error of concentration estimates, based on empirically derived amounts of error. The information is stored in this table, rather than in the program itself, to facilitate easy modification when new error amounts are deemed necessary.

Six columns of information are stored in this file: Type, Pig, Analyte, Low Concentration, High Concentration, and Error. The columns of information are separated by spaces. The Type column contains a C or P: constant error or proportional error. The Pig column contains 0, 1, or 2, corresponding to the detector from which the spectra are collected. The Analyte column contains 0, 1, or 2, corresponding to the analyte compound: radium, potassium, thorium. The Low and High concentration columns contain numbers from 0 to 9e+9, corresponding to the inclusive range of concentrations to be used with this error amount. The Error column contains a number. If the Type column contains P,

this column entry will be treated as a fraction (0.050 will be +/- 5%). If the Type column contains C, this column entry will be treated as a constant number (0.050 will be +/- .050 pCi/g).

The following table is the way ERFILE is currently set up for our system. It produces a 7% error estimate for all analytes for all pigs. It sets a minimum error estimate of 0.3 pCi/g for radium and thorium and a minimum error estimate of 0.5 pCi/g for potassium. The reason the crossover points of 4.29 and 7.14 were chosen is that they are the concentration levels where 7% produces the minimum error for those analytes. In other words, 7% of 4.29 is 0.3 and 7% of 7.14 is 0.5. We want to trap all concentrations below those levels and force them to the constant error estimation.

C	0	0	0.0	4.29		0.300
P	0	0	4.29	9e+9	0.070	
C	0	1	0.0	7.14		0.500
P	0	1	7.14	9e+9	0.070	
C	0	2	0.0	4.29		0.300
P	0	2	4.29	9e+9	0.070	
C	1	0	0.0	4.29		0.300
P	1	0	4.29	9e+9	0.070	
C	1	1	0.0	7.14		0.500
P	1	1	7.14	9e+9	0.070	
C	1	2	0.0	4.29		0.300
P	1	2	4.29	9e+9	0.070	
C	2	0	0.0	4.29		0.300
P	2	0	4.29	9e+9	0.070	
C	2	1	0.0	7.14		0.500
P	2	1	7.14	9e+9	0.070	
C	2	2	0.0	4.29		0.300
P	2	2	4.29	9e+9	0.070	

**LATEST**

Found in the program subdirectory, this file contains the file extension for the next spectrum file that will be written. It is an ASCII text file and contains three upper case letters. The three letters must occur at the beginning of the file (no carriage returns, line feeds, embedded spaces, or any other extra characters). The three letters must occur consecutively: no spaces between them. Spaces, carriage returns, line feeds, or other characters after the three characters do not affect the operation of the program.

**REFER\LATEST**

This LATEST file is located in the reference subdirectory and contains the full pathname from the program subdirectory to the latest reference file. The format is: REFER\XXXXXXX.XXX where REFER\ is the reference subdirectory name and XXXXXX.XXX is the name of the reference file. The format of the reference file name is described in Appendix E.

**TEMPDB.DAT and TEMPDB.BAD**

These files are BASIC-compatible ASCII text files located in the program subdirectory. The two files are identical in format. The program itself creates and fills these table with analysis results. Each sample analysis is placed on a single line of this file. The operator chooses which of these two files will receive the results of the analysis. There are 14 string type fields on each line of the file. Since this is a BASIC compatible file, each of the fields is enclosed in double quotes and separated by commas. The fourteen fields in order are:

Reference Name: The name of the reference file used for the analysis.

Sample Number: The sample number entered by the operator at the time the sample spectrum was collected.

D-9

Spectrum Name:	The name of the spectrum file for this sample.
Sample Weight:	The sample weight entered by the operator at the time the sample spectrum was collected.
Sample Days:	The number of days the sample has been in the container. Entered by operator at the time the sample spectrum was collected.
Radium Conc:	The estimated concentration of $^{226}\text{Ra}$ in the sample (followed by an asterisk if the concentration is below the minimum detectable activity level).
Radium Error:	The estimated error of the concentration estimate of $^{226}\text{Ra}$ .
Potassium Conc:	The estimated concentration of $^{40}\text{K}$ in the sample (followed by an asterisk if the concentration is below the minimum detectable activity level).
Potassium Error:	The estimated error of the concentration estimate of $^{40}\text{K}$ .
Thorium Conc:	The estimated concentration of $^{232}\text{Th}$ in the sample (followed by an asterisk if the concentration is below the minimum detectable activity level).
Thorium Error:	The estimated error of the concentration estimate of $^{232}\text{Th}$ .
Time:	The time of day that the sample spectrum was analyzed.

**D-10**

Date: The date that the sample spectrum was analyzed.

Barrel Number: The barrel number where the soil sample will be stored.  
This was entered by the operator when the sample  
spectrum was collected.

**APPENDIX E**  
**FILE-NAMING CONVENTIONS**



## FILE-NAMING CONVENTIONS

### Reference file names:

A reference file name has the following format:

**YYMMDDX.REF**

where

**YY** are two digits representing the current year (92 for 1992).

**MM** are two digits representing the current month (01 for January, 02 for February, etc.).

**DD** are two digits representing the current day of the month (01 for the first day).

**X** is a letter (A....Z) that makes this reference file name unique. If the first reference file of the day is named 920301A.REF, the second reference file of the day would be named 920301B.REF.

.REF is a constant file extension for all reference files.

### Reference spectra file names:

A background spectrum file that was saved while making a reference file has the following format:

**RRRRRRRLBK**

where

**RRRRRRR** are the first seven characters of the associated reference file name.

**I** is the input number from which this spectrum was collected.

**.BK** is a constant file extension for background spectra files collected for references.

A spectrum file that was saved while making a reference file has the following format:

**RRRRRRRLRFX**

where

**RRRRRRR** are the first seven characters of the associated reference file name.

**I** is the input number from which this spectrum was collected.

**.RF** is a constant part of the file extension for spectra files collected for references.

**X** is a the reference material source (0 = radium, 1 = potassium, 2 = thorium).

#### Spectra file names:

A spectrum file that was saved while analyzing a sample has the following format:

**RRRRRRRLXXX**

where

**RRRRRRR** are the first seven characters of the associated reference file name.

**I** is the input number from which this spectrum was collected.

**XXX** is a set of three letters that uniquely identifies this spectrum.

These three letters provide 17,576 unique combinations. Each new spectrum file steps through the sequence of letter combinations. The sequence is AAA, AAB, ... AAZ, ABA, ABB, ... ABZ, ... AZZ, BAA, BAB, ... ZZZ. When ZZZ is reached the sequence starts again at AAA, but since a new reference file should be in use by this time, the entire spectrum file name is still unique.

**APPENDIX F**  
**PRINTED SPECTRA**



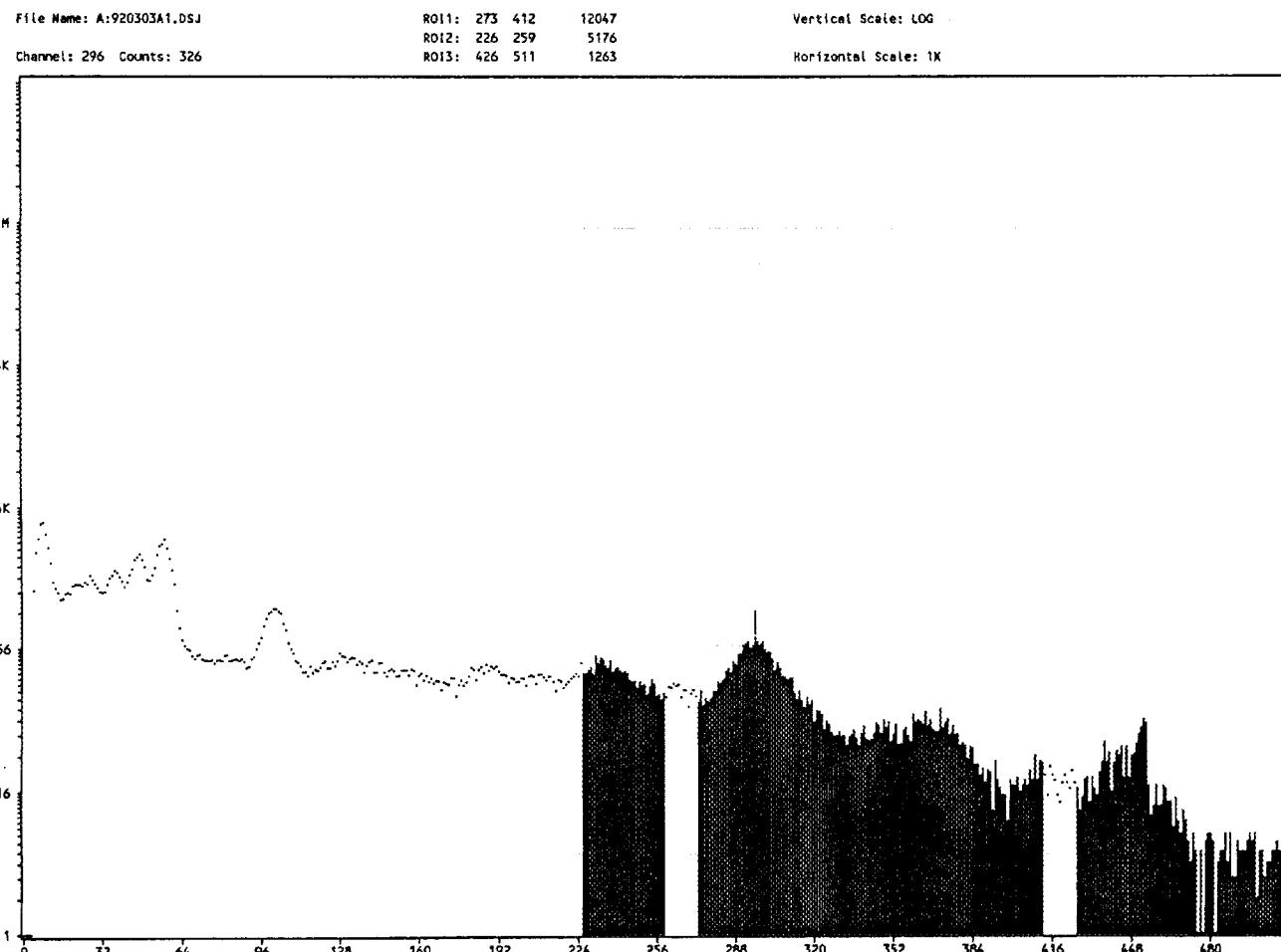


Fig. F.1. Spectrum printed on Hewlett<sup>®</sup> - Packard LaserJet<sup>®</sup> - II printer.

File Name: A:920303A1.DSJ

Channel: 296 Counts: 326

ROI1: 273 412 12047  
ROI2: 226 259 5176  
ROI3: 426 511 1263

Vertical Scale: LOG Horizontal Scale: 1X

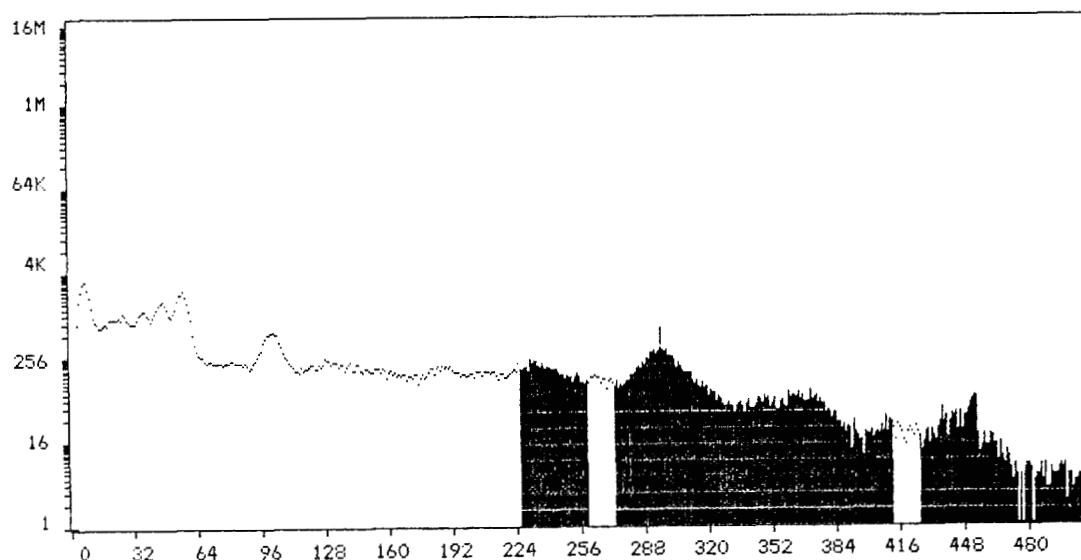


Fig. F.2. Spectrum printed on Texas Instruments<sup>TM</sup> Model 855 printer.

**APPENDIX G**

**SAMPLE ANALYSIS, CALIBRATION, AND**

**QUALITY CONTROL PROCEDURES**



## G.1 SAMPLE ANALYSIS PROCEDURE

### G.1.1 PURPOSE

This procedure outlines the steps necessary for the system operator to analyze a soil sample (or multiple soil samples). The purpose of sample analysis is to determine the concentrations of  $^{226}\text{Ra}$ ,  $^{40}\text{K}$ , and  $^{232}\text{Th}$  in a soil sample.

### G.1.2 MATERIALS NEEDED

The most important item is the sample itself. The sample must have been prepared to meet granularity and homogeneous specifications (which are outside the context of this memorandum). The sample must be in a container of the same shape and size as those used in the reference calibration. The weight of the sample must be known. For consistent results, all samples should be filled to the same level in the container. This is important because the number of radioactive counts recorded by the detector is in direct proportion to how close the radioactive source is to the detector. A sample container half-filled would have its material proportionally closer to the detector than a sample container completely filled. A half-filled container would have more counts in proportion to its weight and would, therefore, result in a higher radionuclide concentration estimation.

A floppy disk for recording sample gamma spectra is needed in the A: drive. Another formatted disk should be kept on hand so that, if the disk should become full during analysis, the new disk can be placed into the drive without exiting the analysis program to format another.

### G.1.3 PROCEDURE DESCRIPTION

- 1) Turn on the computer and printer. If using a TI-855 printer, set it to print at 12 characters per inch; otherwise, printing of results will wrap

around to two lines instead of one. If the computer is using a shared printer on a local area network, make sure the computer is properly spooled to that printer.

- 2) Make sure the amplifiers and DMR™ unit are turned on and up to operating temperature. Changes in temperature affect the operation of the amplifiers, creating changes in energy calibration.
- 3) Perform an energy calibration, described below, to ensure that gamma analysis system is acquiring gamma energies into the proper channels.
- 4) Execute the program by changing to the program directory (e.g., C:\NSOIL) and typing SOIL <Enter>.
- 5) At the Main Menu choose the "Analyze" option.
- 6) From the Analysis Menu choose the "Load Reference" option.
- 7) From the Load Reference Menu choose either the "Use Most Recent" or the "Type In Ref Name" option. Generally, the most recent reference is the one to use. If the most recent one is not used, a prompt is given to type in the name of reference file (file name only, not the full path name). This reference file must be located in the reference subdirectory.
- 8) If found and properly loaded, the name of the reference is displayed at the bottom of the screen. Choose "Quit" from the Load Reference Menu to return to the Analysis Menu.
- 9) Choose "Analyze Sample" from the Analysis Menu.
- 10) At this point, column headings are output to printer. If the program appears to lock up, check the printer for power and that it is on-line and has paper.
- 11) The Analyze Sample Menu has three columns of choices, one column for each input. The "View" option brings up the MCA screen for that input. The "Start" option triggers count collection for that input for a sample. The "Analyze" option determines the concentration of the radionuclides in a spectrum.
- 12) Place the sample into one of the pigs.

- 13) Choose the "Start" option for the input corresponding to the sample. Input the following at the prompt: a) the weight of the sample, b) the number of days the sample has been in the container, c) the sample number (to uniquely identify each sample), and d) the barrel number (to identify the storage location of the sample once analysis is completed). The program then begins to acquire counts for the sample spectrum. A plus sign (+) appears in the menu on each side of the word "Start" to indicate that counts are being acquired for the input. When 300 live seconds have elapsed, counting stops for the input, and an asterisk (\*) replaces the plus sign on the menu to indicate that the sample may now be analyzed.
- 14) Choose the "Analyze" option. The concentration estimates for each of the three analytes (radium, potassium, and thorium) are displayed on the screen along with the error estimate and the minimum detectable activity flag for each.
- 15) The operator next chooses whether or not to save the analysis results.
- 16) If "Yes", the operator is asked which data base to use:  
TEMPDB.DAT for a "good" analysis or TEMPDB.BAD for an inadequate analysis.
- 17) The results are then printed.
- 18) The gamma spectrum is written to a spectrum file on the A: drive. If an error occurs, it is probably due to a full disk. When this happens, insert an empty formatted disk into the A: drive and choose "Analyze" again.
- 19) The analysis results are now written to the file chosen by the operator (TEMPDB.DAT or TEMPDB.BAD). These are holding files for the results. The results will be uploaded from these files to a database, normally once per day.
- 20) Finally, the name of the spectrum file is displayed on the screen until the operator presses the Enter key to return to the ANALYZE SAMPLE MENU.

- 21) Steps 12 through 20 are repeated for each additional sample. All three of the inputs may be used concurrently to analyze samples.

## G.2 CALIBRATION PROCEDURES

### G.2.1 QUARTERLY LINEARITY CHECK

#### G.2.1.1 Purpose

In an MCA system, there must exist a linear relationship between a pulse height (in volts) and the channel in which the pulse is stored. For example, if a pulse of 8 volts is stored, in channel 500 and a pulse of 4 volts is stored in channel 250, then a pulse of 0 volts would be stored at channel 0. Periodically (once every three months), the system is checked to make sure this relationship is linear. The process for setting the zero intercept (the channel for a pulse of 0 volts) is different and is described in the hardware installation section above.

#### G.2.1.2 Equipment Needed

A pulser is attached to the input of the DMR-II™ unit.

#### G.2.1.3 Procedure Description

Execute the soil program by changing to the program subdirectory (e.g., C:\NSOIL) and typing SOIL <Enter>. From the Main Menu of the program, choose the "Calibrate" option.

The following procedure must be repeated for each DMR™ input:

1. Attach the pulser to the DMR™ input.

2. Choose the "View" option from the Calibration Menu for the corresponding input. This will bring up the MCA screen.
3. Change the vertical scale to "Log Scale" by pressing the up arrow.
4. Make sure the Screen is in FILL mode by pressing F8 until the DOT prompt appears.
5. Begin acquiring counts by pressing F1 until the STOP prompt appears.
6. Adjust the pulse output height on the pulser until the counts are accumulating near the top of the spectrum (channel 500+).
7. Once the counts are accumulating near the top of the spectrum, clear the spectrum by pressing the F2 key.
8. Begin stepping the pulse height down in equal steps, pausing at each step to allow a recognizable spike to appear on the MCA screen. Approximately ten steps should be adequate.
9. Once the bottom of the spectrum has been reached, press the F1 key to stop acquiring counts.
10. Put the cursor on each of the spikes in the spectrum and record the channel number at that point.
11. Calculate the difference in channels between the spikes. This distance should be constant (within a channel) all the way up the spectrum.
12. If the distance does not remain constant, i.e., grows steadily closer or farther as it progresses up the spectrum, a problem exists somewhere in the MCA hardware. In this unlikely event, an electronic specialist should check out the system to determine the exact cause of the problem.

## G.2.2 QUARTERLY REFERENCE CALIBRATION

### G.2.2.1 Purpose

The three-ROI sample analysis method (described in Appendix B) relies on a linear relationship between the number of counts in ROIs in a gamma energy spectrum and the amount of radionuclide(s) present in a sample. This linear relationship (sometimes referred to as the counting efficiency) must be quantified periodically by obtaining gamma spectra from known amounts of the radionuclides, summing the counts in the ROIs, and storing the results in a reference file. The process, also referred to in the context of this program as building a reference file, is conducted for all input since each input has a different counting efficiency. The process must be performed each time there is a hardware change to the system (any change other than a gain change on the amplifiers) or at least quarterly following the linearity check.

### G.2.2.2 Materials needed

An empty sample can (for background counts) and samples of  $^{226}\text{Ra}$ ,  $^{40}\text{K}$ , and  $^{232}\text{Th}$  are needed to perform this procedure. Referred to as reference standards, these samples contain known concentrations of each of the three radionuclides. The reference calibration procedure can be accelerated by using one container for each input so that spectra can be obtained for all inputs concurrently.

### G.2.2.3 Procedure description

1. Execute the soil program by changing to the program subdirectory and typing SOIL <Enter>.
2. From the Main Menu choose the "Analyze" option.
3. From the Analysis Menu choose the "Make New Reference" option.  
The New Reference Menu has three columns of options, one for each

input. The bottom row of choices, "Save", "Quit", and "Reset", refers to the whole reference file.

4. Load a background container into each pig.
5. Choose the "Background" option for each input. A plus sign (+) appears on the sides of the option on the menu to indicate that background spectra are being collected. The "View" options may be chosen at any time to see the background spectra. All reference spectra accumulate for 3000 seconds (50 minutes), ten times as long as for sample spectra. The timer counts from minus 2700 to 300 seconds. When the timer reaches 300 seconds, the spectrum has been collected, and asterisks (\*) appear in place of the plus signs to indicate completion.
6. Following acquisition of background spectra, remove the background containers from the pigs. Load the radium reference containers into the pigs, noting the weights of the reference materials.
7. Choose the "Radium" option from the menu.
8. Input the following at the prompt: a) the weight of the radium reference; b) the concentrations of each of the reference radionuclides in the radium reference (pressing Enter will input the default concentrations for our reference standards); and c) "Change", "Okay", or "Cancel". If something was entered wrong, choose "Change" to correct it. Choose "Cancel" to abort. If everything is correct, select "Okay", and the reference spectrum is collected.
9. Repeat steps 6 through 8 for each input.
10. When all spectra have been collected for the radium references, repeat the above steps for the potassium references and then for the thorium references.
11. If at any point all spectra need to be reacquired, choose the "Reset" option from the New Reference Menu.
12. After all spectra have been collected, choose the "Save" option from the New Reference Menu. The reference file is written to the reference subdirectory and the name of the reference file saved in the

REFER\LATEST file. Choosing the "Use Most Recent" option from the Load Reference Menu loads this reference file. The name of the latest reference file is displayed at the bottom of the screen.

13. Choose the "Quit" option to go back to the Analysis Menu.

## **G.2.3 DAILY ENERGY CALIBRATION**

### **G.2.3.1 Purpose**

The gamma analysis program relies on the fact that the MCA correlates a specific gamma energy to a particular channel of count data. For example, gamma rays emitted by  $^{137}\text{Cs}$  have 661.64 KeV of energy; if the MCA is properly calibrated, at 6 KeV per channel, these gamma rays are counted in the 110th channel (numbered 109 on our system because the first channel is number 0). The purpose of the daily energy calibration is to ensure that the MCA is placing gamma ray counts in the proper channels. This is accomplished by using standard radionuclides that emit gamma rays of a known energy and adjusting the gain on the linear amplifiers until those gamma rays are being counted in the proper channels. This process should be performed at the start of each sample analysis session. It should also be performed when the operational check (described below) fails.

### **G.2.3.2 Materials needed**

Samples of  $^{137}\text{Cs}$  and  $^{40}\text{K}$  are needed to perform this calibration. One sample per input, allowing simultaneous calibration, speeds up the process.

### **G.2.3.3 Procedure Description**

1. Execute the program by changing to the program subdirectory and typing SOIL <Enter>.

2. From the Main Menu choose the "Calibrate" option. The Calibration Menu has three columns of options, one for each input.
3. Place a  $^{137}\text{Cs}$  sample into each pig.
4. Choose the "Cesium" option for each input, initiating count collection. This also sets an ROI for the cesium energy peak on the MCA screen and places the cursor at channel 109, which should be the channel at the top of the cesium energy peak.
5. Choose the "View" option to see the energy spectrum being collected. Because the cesium energy peak grows rapidly, it is easiest to view on the log vertical scale.
6. If the top of the peak is below (to the left of) channel 109, the amplifier gain needs to be increased. If the top of the peak is above (to the right of) channel 109, the amplifier gain needs to be decreased.
7. Adjust the fine gain setting on the corresponding TC-240 amplifier by turning the knob counterclockwise to decrease and clockwise to increase. Note that there is a locking mechanism that must be released before turning the fine gain knob. This should be relocked after making adjustments.
8. Press the F2 key from the MCA screen to clear counts in the spectrum, and watch the energy peak being formed following amplifier gain adjustment.
9. Repeat steps 6 through 9 until the top of the energy peak is at channel 109. If desired, change the horizontal scaling factor to stretch out the energy spectrum for viewing the top of the peak more clearly.
10. Repeat steps 2 through 9 using the  $^{40}\text{K}$  sample. Choose the "Potassium" option from the Calibration Menu to start the process. The top of the potassium peak should be at channel 242.
11. Because the potassium energy peak is smaller (not so many counts accumulate), it is easier to view on the 256, 512, or 1K vertical scale. The fact that fewer counts are accumulated coupled with the scattering effect of the NaI(Tl) detectors causes the peak to be more poorly

defined than that of the cesium. Once a spectrum has finished accumulating, choose "Smooth" option from the Calibration Menu to minimize irregularities in the spectrum. The Smooth option removes the randomness from the spectrum by averaging adjacent channels. Each time the Smooth option is selected, the averaging process occurs again on the already smoothed spectrum. The Smooth option should be used a maximum of three times per spectrum to keep peaks from "creeping".

### **G.3 QUALITY CONTROL PROCEDURES**

To ensure that the sample analysis system is operating properly (i.e., providing reliable results for the samples), it is necessary to perform continuing operational checks and to cross-check our results with another laboratory.

#### **G.3.1 OPERATIONAL CHECK**

##### **G.3.1.1 Purpose**

The operational check is an ongoing process during the analysis session to see if a known concentration of  $^{226}\text{Ra}$  results in the correct concentration estimation. The underlying effect of this process is to make sure that the energy calibration is still correct, that the gain of the linear amplifiers has not changed.

##### **G.3.1.2 Materials Needed**

Samples of  $^{226}\text{Ra}$  in concentrations of 5 pCi/g and 15 pCi/g are needed for this procedure. Three samples of each help speed up the process by checking all inputs at the same time.

**G.3.1.3 Procedure Description**

1. At the beginning of each analysis session, the three samples of 5 pCi/g  $^{226}\text{Ra}$  are analyzed as samples (one in each input). If the concentration estimates from the analysis do not match the known concentration of the material, recheck the energy calibration. If the energy calibration is correct, the anomaly can be attributed to the random nature of the radioactive decay process. If the readings continue to be wrong and the energy calibration is correct, the problem lies with the counting hardware, with gamma radiation contamination from other sources, or with the sample standard itself. If the sample standard container is not completely filled, a higher concentration estimate may result.
2. Repeat step 1 for the 15 pCi/g  $^{226}\text{Ra}$ .
3. After seven samples have been analyzed in each input during the analysis session, analyze the 15 pCi/g  $^{226}\text{Ra}$  standard, and follow the procedure outlined in step 1. These quality control samples may be analyzed more or less frequently at the operator's discretion.

**G.3.2 CROSS CHECK WITH INDEPENDENT LABORATORY**

To further verify that analysis results are correct, samples are sent monthly to another laboratory for analysis. Three samples are chosen from each analysis session, one for each of the three inputs. The results from the independent laboratory are compared to the results from our analysis. Statistical methods are used to evaluate the comparisons.



## INTERNAL DISTRIBUTION

1.	K. M. Woynowskie	14.	S. V. Kaye
2.	J. R. Davidson	15.	P. S. Rohwer
3.	S. J. Wallace	16.	P. T. Owen
4.	J. A. Rice	17 - 21.	ORNL/GJ Library
5.	G. A. Pierce	22.	ORNL Technical Library, Y-12
6.	M. K. Jensen	23.	ORNL Patent Section
7.	M. J. Wilson	24.	Central Research Library
8.	C. A. Little	25 - 26.	Laboratory Records
9 - 13.	J. E. Wilson	27.	Laboratory Records - RC

## EXTERNAL DISTRIBUTION

- 28. Office of Assistant Manager, Energy Research and Development, Oak Ridge Operations Office, P.O. Box 2001, Oak Ridge, Tenn. 37831-8600
- 29 - 38. Office of Scientific and Technical Information, U.S. Department of Energy, P.O. Box 62, Oak Ridge, Tenn. 37831