7-1-1989

Implementation of a research goniospectrophotometer for appearance research

R. Mitchell Miller

IMPLEMENTATION OF A RESEARCH GONIOSPECTROPHOTOMETER FOR APPEARANCE RESEARCH

by

R. Mitchell Miller

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in the
Center for Imaging Science in the
College of Graphic Arts and Photography
of the Rochester Institute of Technology

July, 1989

Signature of the Author R. Mitchell Miller
Center for Imaging Science

Accepted by _____
Coordinator, M.S. Degree Program

College of Graphic Arts and Photography
Rochester Institute of Technology
Rochester, New York

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of R. Mitchell Miller
has been examined and approved
by the thesis committee as satisfactory
for the thesis requirement for the
Master of Science degree

Dr. Roy Berns, Thesis Advisor

Mr. Mark Fairchild

Mr. Nestor Rodriguez

July 28, 1995
Date

THESIS RELEASE PERMISSION FORM

ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF GRAPHIC ARTS AND PHOTOGRAPHY

Title of Thesis: Implementation of a Research Goniospectrophotometer for Appearance Research

I, R. Mitchell Miller, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date 7-28-89

IMPLEMENTATION OF A
RESEARCH GONIOSPECTROPHOTOMETER
FOR APPEARANCE RESEARCH

by

R. Mitchell Miller

Submitted to the
Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science degree
at the Rochester Institute of Technology

ABSTRACT

A research-grade goniospectrophotometer was modified to improve its measurement capabilities for appearance research. Hardware, optical, and environmental modifications produced a device capable of measuring the spectral reflectance factors of a sample over the wavelength range from 380 to 700 nanometers at user-selectable illuminating and viewing geometries. Several extensive Pascal programs were written to provide a user-friendly interface to facilitate the control of the instrument, as well as, the collection, reduction, and display of data.

ACKNOWLEDGEMENTS

Successful completion of this thesis owes recognition to support from many sources. Valuable and timely assistance and advice was appreciated from the following:

Dr. Roy Berns of the R.I.T. Munsell Color Science Laboratory who kindly consented to act as thesis advisor after the tragic death of Dr. Franc Grum, the initial advisor;

Eastman Kodak Company, Rochester, NY, who loaned the goniospectrophotometer that made this thesis possible;

Dr. Jack Hsia of the National Institute for Standards and Technology, who provided absolute calibrations of several standard tiles;

Mr. Nestor Rodriguez of the Eastman Kodak Company, who provided many materials and advise;

Mr. Denis Daoust of Rochester Institute of Technology, who provided artwork of the modified goniospectrophotometer; and,

Mr. George Fazekas, Program Coordinator of Computer Technology at Monroe Community College, who assisted in debugging electronic circuit problems.

The support of a C.I.A. grant is acknowledged with appreciation.

DEDICATION

This thesis is dedicated in memory of

Dr. Franc Grum

and

Dr. Ronald Francis

whose guidance and expertise

inspired my interest in color

and imaging science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(CONTINUED)

INTRODUCTION

Hunter[1] states that the appearance of an object involves not only the object's color, but geometric attributes as well. He continues by emphasizing that the design of devices to measure appearance attributes must consider the physical source of the attributes and the human responses to them. This thesis will discuss the implementation of a goniospectrophotometer, a device that measures the spectral reflectance factors* of an object as a function of both illuminating and viewing geometries. From these data, the color of an object can be determined as a function of geometry.

To measure reflectance**, a device typically incorporates an integrating sphere. The sphere is used in the illuminating or viewing geometry to provide or collect diffuse, non-directional radiation. For example, an integrating sphere in the collection geometry would combine radiation reflected at all angles from the sample surface

* "Reflectance factor is the ratio of the radiant flux actually reflected by a sample surface to that which would be reflected into the same reflected-beam geometry by an ideal (lossless) perfect diffuse (lambertian) standard surface irradiated in exactly the same way as the sample[2]." The term becomes spectral reflectance factor when measured as a function of wavelength.

** "Reflectance is the ratio of reflected to incident flux[2]." The term becomes spectral reflectance when measured as a function of wavelength.

and provide the integrated radiation to an appropriately placed detector. The integrating sphere is required since the definition of reflectance requires that radiation reflected from a sample be collected at all angles. Reflectance factor, however, compares the amount of flux reflected at a particular angle to the amount that would have been reflected at the same angle by a similarly irradiated lambertian surface. Instruments that measure reflectance factors typically measure what is known as bidirectional reflectance at specified viewing and illuminating angles. Bidirectional 0/45 reflectance factors would be measured by irradiating a sample at 0 degrees and collecting at 45 degrees. A goniospectrophotometer measures bidirectional reflectance at selectable illuminating and viewing angles.

Traditional spectrophotometers measure spectral reflectance and spectral reflectance factors with fixed illuminating and viewing geometries. Though the geometries can vary from diffuse to directional, such instruments can only be used to describe the color of an object for the geometry used. If the geometry used does not relate to the object's intended use, the color measured may be of little value. A goniospectrophotometer, however, can be used to determine the color of a sample with illuminating and viewing geometries selected to relate to the object's use.

Standard reference materials (SRMS) should only be selected after considering a material's goniospectrophotometric and other properties[3,4]. Geometry differences among instruments could create errors in transferring photometric calibrations. For example, the CIE definition of bidirectional 0/45 geometry[5] states that the illuminating angle can be up to 10 degrees off axis and the collection angle in the range of 40 to 50 degrees. Fairchild[6] has shown that the limits of these recommendations produce unacceptable variations in the measured reflectance factors. Therefore, instruments meeting CIE specifications may not be able to accurately transfer photometric scale calibrations.

Eastman Kodak Company's generous long-term loan of a custom-built research goniospectrophotometer to the RIT Munsell Color Science Laboratory has made this thesis possible. This thesis concentrates on the modifications made to this goniospectrophotometer, in order to improve the instrument's capabilities. The procedure required hardware, optical, and environmental modifications. In addition, several extensive Pascal programs were required to provide a user-friendly interface to facilitate the control of the instrument, as well as, the collection, reduction, and display of data. These programs are fully documented in Appendices A - E.

EXPERIMENTAL

The original configuration of the goniospectrophotometer is shown schematically in Figure 1. The system did not include the gonio arm drive, counters, plotter, and computer. Several hardware and software modifications were made throughout this project in an attempt to improve the instrument for specific applications. Component diagrams of the modified instrument are shown in Figures 2 and 3, with an optical diagram in Figure 4.

The Eastman Kodak configuration of the goniospectrophotometer included a light source, monochromator, sample holder, radiometer, and intermediate optics. A Leitz model XLS-1A-M10 150 watt Xenon lamp assembly and regulated power supply provided white light to the entrance slit of the Schoffel model GM-250 holographic grating which, in turn, provided narrow-band radiation of the selected bandwidth at the exit slit. Micro-adjustable entrance and exit slits selected the bandwidth of radiation that would be delivered to the sample surface by a lens. The sample could be manually rotated about two axes to adjust its orientation during measurements. A Photoresearch model 1500 Spectra-Spotmeter radiometer was mounted on a gonio arm that rotated about the sample. A folding prism (shown in Figure 4) permitted collection geometries as close as five degrees to the irradiating beam. A Hewlett Packard HP-85 computer

GONIOSPECTROPHOTOMETER

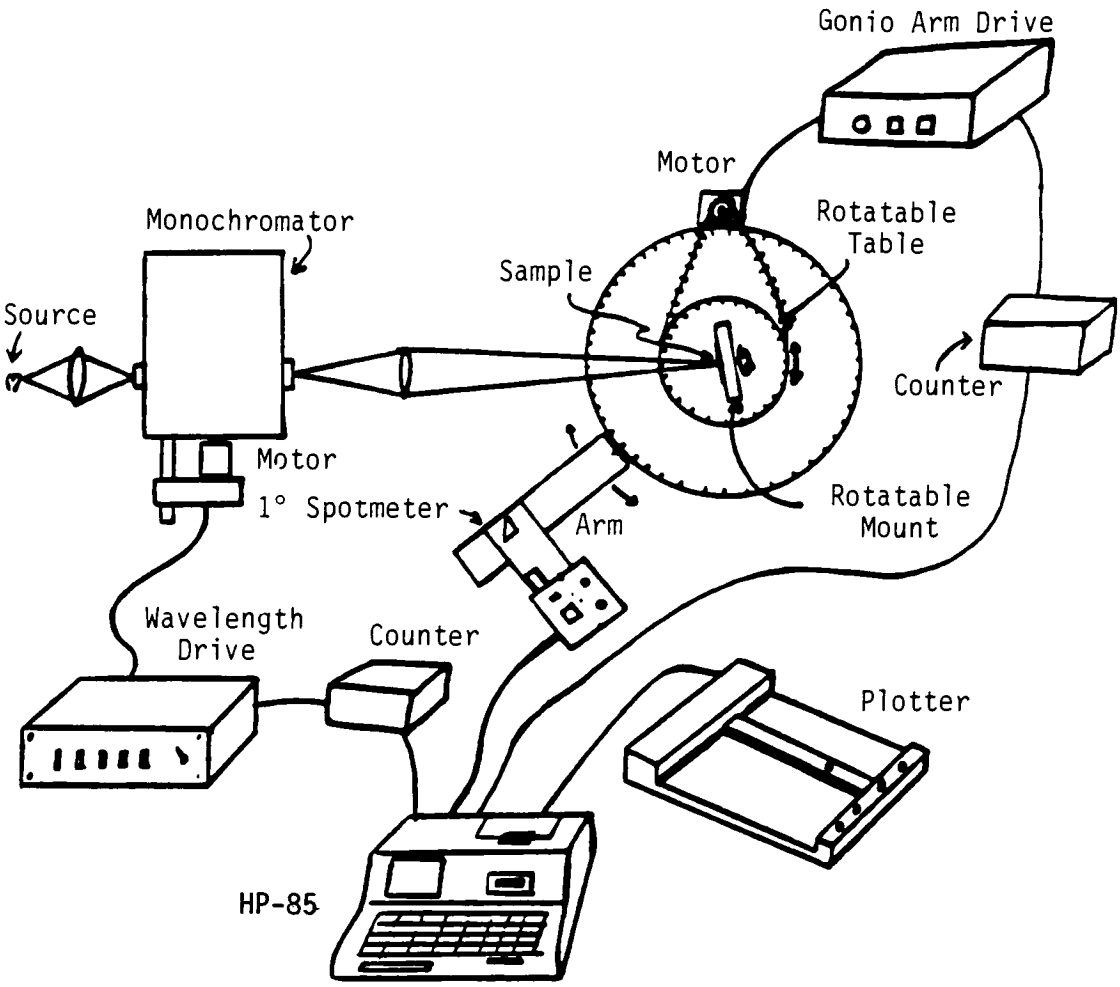
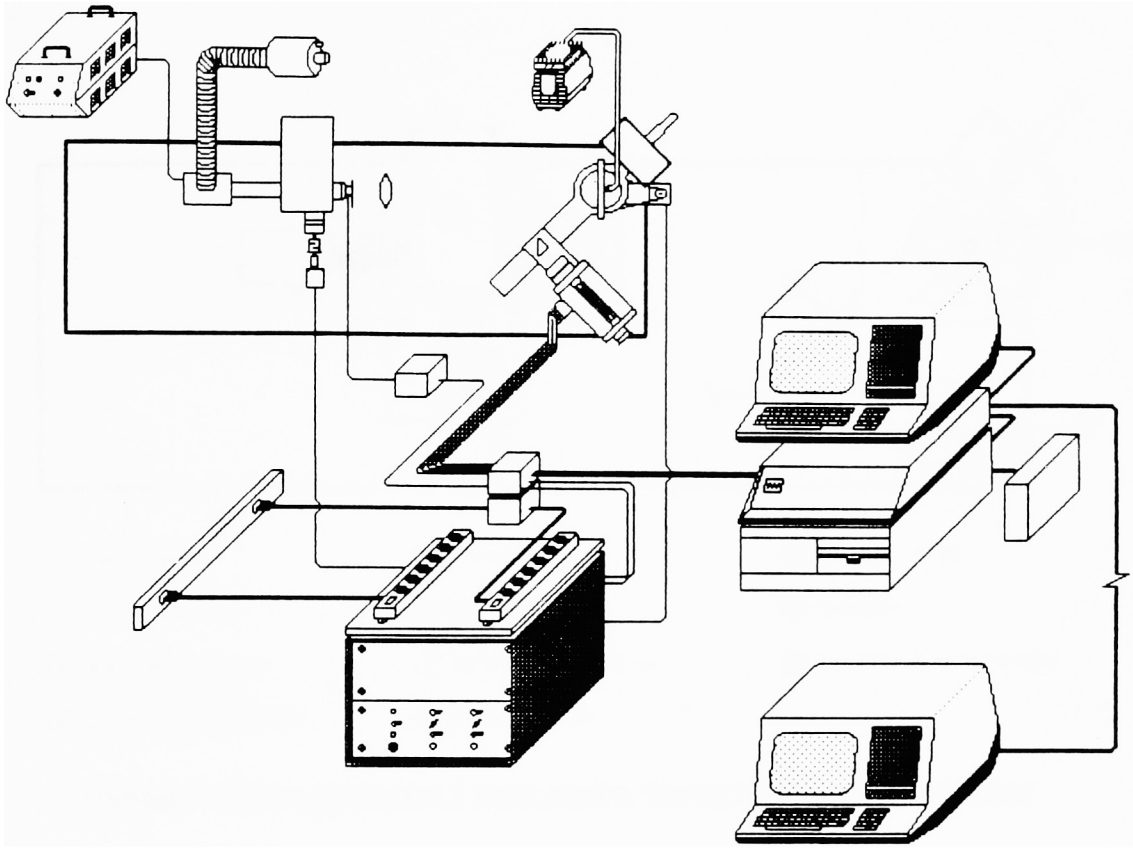


Figure 1. Diagram of the Eastman Kodak Goniospectrophotometer



MCSL Goniospectrophotometer - Component Diagram

Figure 2.

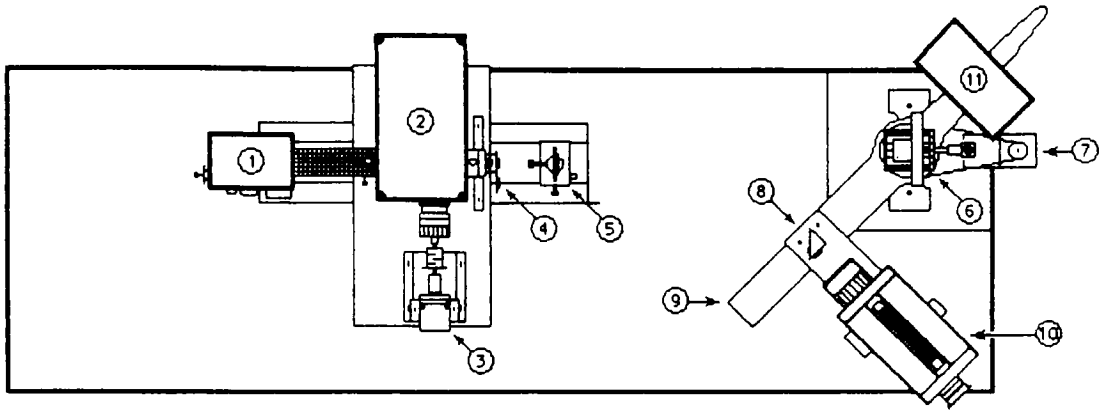


Figure 3. Diagram of the MCSL Goniospectrophotometer

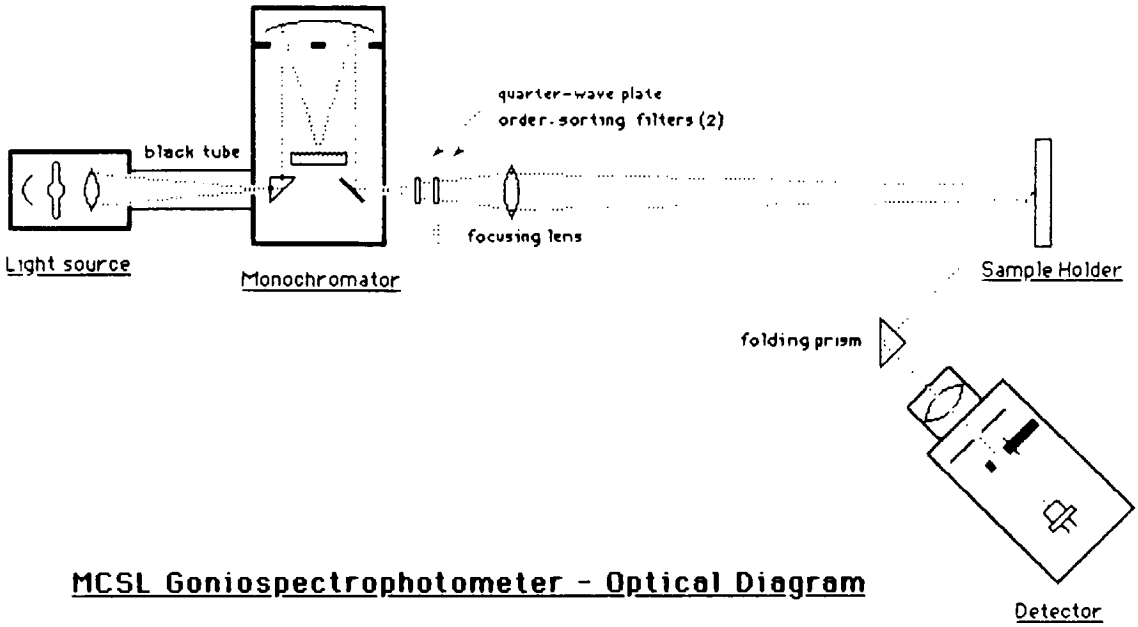

quarter-wave plate
order-sorting filters (2)
black tube
focusing lens
Light source
Monochromator
Sample Holder
folding prism
Detector
MCSL Goniospectrophotometer - Optical Diagram

Figure 4.

controlled the gonio arm, wavelength positioning, and sample measurements in the original configuration.

The following describes the modifications made to the goniospectrophotometer and software developed:

A. Light Sources

The spatial and temporal uniformities of the irradiated beam from the Xenon lamp were poor. The arc appeared to wander in the quartz envelope of the Xenon lamp which translated into non-uniformities during lengthy measurement times. Due to limited funds and the high cost of alternate illuminating systems, the Xenon lamp was replaced with an inexpensive tungsten lamp for some experiments. The tungsten lamp provided higher uniformity, but proportionately less power at wavelengths in the blue region of the spectrum than the Xenon arc.

An exhaust fan and hose were connected between the lamp assembly and plenum above the laboratory ceiling to remove unwanted ozone and heat for safety considerations.

Figure 4 shows the location of a black tube inserted between the light source and monochromator. The tube and appropriately applied black tape reduce stray light from the lamp housing significantly.

The lamp assembly was aligned for maximum throughput and uniformity at the sample plane prior to making measurements.*

B. Monochromator

The optical path of the Schoffel holographic grating monochromator is shown in Figure 4. A Micro-adjustable slit at the exit aperture controls the bandpass of the radiation reaching the sample plane. The exit slit size and the location of a focusing lens determine the illuminated sample area. The parameters were chosen to irradiate a one square centimeter area of the sample positioned at zero degrees. Wavelength calibration was performed and later verified by examination of emissions at known wavelengths for a Mercury arc lamp and Helium-Neon laser. A Tracor Northern diode array spectroradiometer was used to confirm the calibration of exit slit size to half-height wavelength bandpass.

The Schoffel monochromator was originally controlled by a SPEX stepping motor and controller. Analysis of the drive system showed a random error of one to two nanometers for every fifty nanometers of

* A two-dimensional control was adjusted on the rear of the lamp housing until the illuminated sample area (illuminated with 555 nanometer light) appeared to have maximum brightness and uniformity.

wavelength travel. This error was most likely due to worn parts in a gear reducer connecting the drive to the monochromator. Figure 3 shows the connection of a new Superior Electric Company Slo-Syn stepping motor model M061-FD08 and coupling that proved reliable over the full wavelength range. The motor was controlled by a Superior Electric Company Slo-Syn Translator model STM-101.

As noted in the literature[3,7-14], instrument polarization can cause errors in reflectance and reflectance factor measurements. Reflectance factors can be measured for the p and s states of polarization of the illuminating and viewing geometries. Radiation is polarized parallel to the plane of incidence in the p state, and the radiation is polarized perpendicular to the plane of incidence in the s state. Instruments utilizing an integrating sphere in the illuminating geometry would measure two reflectance factors for the p and s states of polarization of the viewing geometry. Likewise, two measurements of reflectance factors would be required to characterize the p and s states of polarization of the illuminating geometry where the measuring instrument had a integrating sphere in the viewing geometry instead of the illuminating geometry. Without an integrating sphere, four measurements of

reflectance factors are required to characterize the p and s states of polarization for both the illuminating and viewing geometries. The average of reflectance factors measured for each case would equal the reflectance factors for the unpolarized state. Since the polarized method requires two or four times the measurements, and only the unpolarized case of reflectance factors are desired for our application, the reflectance factors were directly measured by depolarizing the incident radiation as much as possible[3].

The polarization of the incident beam was checked by rotating a linear polarizer in the optical path between a barium sulphate sample and detector. Signal changes of plus and minus fifty percent indicated a high degree of polarization in the light source and monochromator. A circular polarizer, or polarization scrambler, was positioned at the exit slit of the monochromator and rotated until the above test showed essentially no signal change, indicating a depolarized state. The circular polarizer was secured and the same test was repeated later during this experimentation to confirm its proper alignment.

The second-order effects of the monochromator were absorbed using cutoff filters which were mounted between the exit slit of the monochromator and the lens shown in Figures 3 and 4. An orange-red filter absorbed the shorter wavelengths, and a blue-green filter absorbed the longer wavelengths. In the original design the filters were manually interchanged at a green wavelength during a spectral scan. The process was automated by fabricating an assembly that held the filters to a computer-controlled solenoid.

C. Sample Holders

Figure 3 shows the location of the sample holder. The holder was free to rotate about two axes thus permitting the sample surface to rotate about the axis of the irradiating beam and about a vertical axis. Angles could be positioned with a precision of 5 minutes of arc.

The original sample holder used a vacuum to secure a sample during measurement. Since many samples could not be held by the vacuum system*, c-clamps were often used to secure samples to the holder. Small pieces of rubber were used to protect the sample surface from abrasion.

* The system would not hold porcelain tiles having a metal rim, samples that were too small, or any sample with an irregular rear surface.

The original design included a micrometer to set the distance between the detector and sample surface accurately. The micrometer was adjusted for the thickness differences of samples measured. The following technique was developed for setting this micrometer quickly and easily. If the micrometer was correctly adjusted, the measuring aperture (as seen through the tele-radiometer) would not deviate from the center of the illuminated sample area during gonio arm rotation between the angles of 10 and 80 degrees. The micrometer was adjusted until the above condition was met.

The illuminating angle was controlled by the rotation of the sample holder. At zero degrees the sample surface is parallel to the exit slit of the monochromator. Please refer to Figure 5 for the definitions of illuminating and viewing angles. The sample could be rotated through 360 degrees though the angles used rarely exceeded plus-or-minus 80 degrees. As the angle theta increased from zero degrees, the illuminated sample area increased at the rate of 1/cos(theta). At the limiting case of 90 degrees and beyond, the sample blocked the irradiating beam from striking the sample surface.

ILLUMINATING AND VIEWING ANGLES

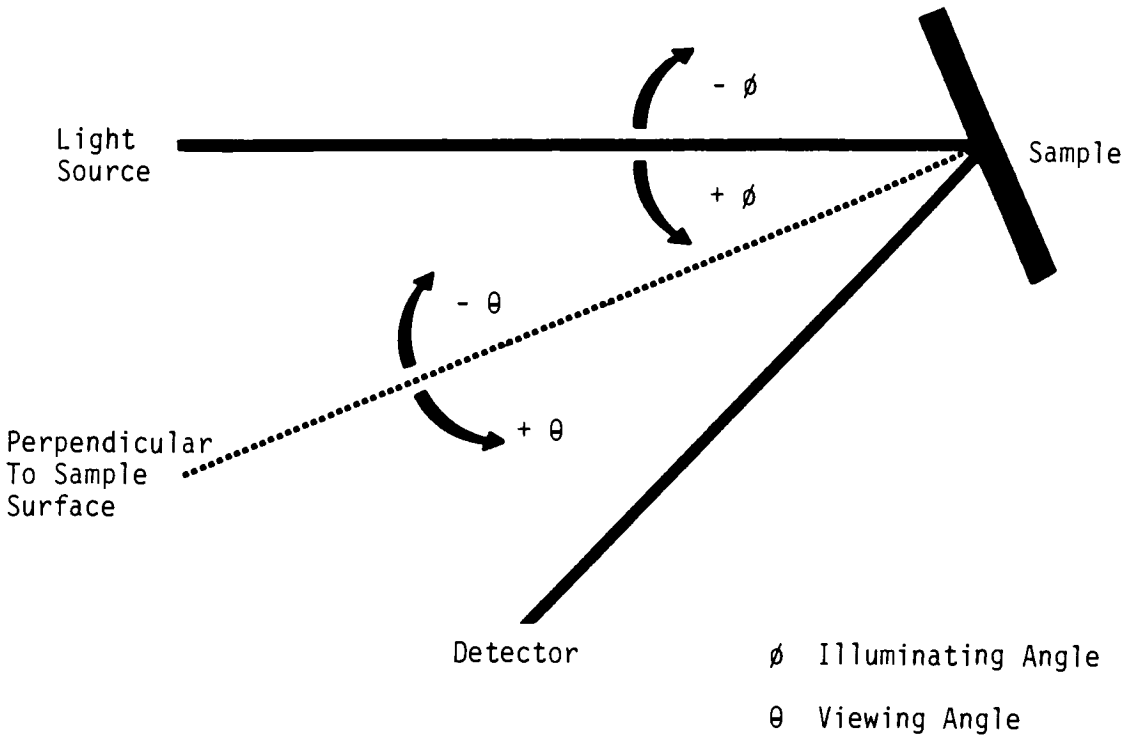


Figure 5. Definition of Illuminating and Viewing Angles

A Borg-Warner temperature controller and thermoelectric heat pump, model numbers TC-108 and AHP-300 respectively, controlled the temperature of the sample during measurements. A mount was fabricated to hold the heat pump at the sample plane. The heat pump surface could be controlled to within 0.1 degree Centigrade over the range of -30 to 55 degrees Centigrade.

D. Gonio Arm

The location of the counter-weighted gonio arm is shown in Figure 3. The arm was free to rotate over 360 degrees, but was seldom used beyond the range of 10 to 150 degrees from the irradiating beam. Angles could be positioned with a precision of 5 minutes of arc. At angles less than five degrees the radiometer would block the irradiated beam, and at large angles, the sample holder would block the irradiating beam.

A stepping motor provided computer control of the arm. The angle of the arm minus the angle of the sample holder equals the viewing angle. Figure 5 shows the illuminating and viewing angles.

The stepping motor controller for the gonio arm drive was not available, so the stepping motor was replaced with the same model motor and controller which was used to control the monochromator.

The weight of the detector and counter-weighted arm was so heavy that the arm positioning was accurate only when continually positioned in the same direction. Backlash of the gear and drive belt caused positioning errors when directions were changed. This problem was solved only by positioning the arm manually when the direction changed.

E. Detector Systems

A Photoresearch model 1500 Spectra-Spotmeter radiometer was mounted on the gonio arm as shown in Figure 3. Figure 4 shows the optical path from the sample, through a folding prism, to the detector. This tele-radiometer is a broad-band instrument that measures the radiance of an object through a 1.0 degree angle while simultaneously providing the operator with a view of the sample field excluding the measuring aperture.

The following changes were made to the radiometer system. The low sensitivity of the tele-radiometer's silicon detector to blue wavelengths led to the pursuit of a PMT detector. A similar meter with a 0.25 degree measurement and PMT detector was borrowed from the Imaging and Photographic Science Division of RIT. The PMT tele-radiometer did not have a computer interface, but examinations of the original tele-radiometer revealed the interface was a series of wire connections

from the rear of a panel meter in the radiometer to an externally mounted DB25 connector. A hole was cut in the side of the PMT tele-radiometer for the installation of a similar connector and wires. The modified tele-radiometer's interface worked successfully. The PMT tele-radiometer had more sensitivity at the shorter wavelength than the silicon detector, but the PMT was very low in sensitivity at the longer wavelengths. The PMT tele-radiometer's low sensitivity to red light was appropriate for use with the tungsten source which was rich in red output. Neither detector performed well over the complete visible wavelength range, but the PMT tele-radiometer gave acceptable results over the wavelength range from 380 to 700 nanometers. The silicon tele-radiometer's low sensitivity in the blue region of the spectrum limited precise measurements at wavelengths below 420 nanometers, however, there was good sensitivity which peaked beyond 700 nanometers. It was not practical to switch the tele-radiometers during a measurement due to geometry differences between the instruments. The PMT tele-radiometer was selected for the remaining measurements. The radiometric filter was never used during measurements with either tele-radiometer since spectrally-flat measurements were not required.

F. Measurement Standards

Dr. Jack Hsia of the National Institute for Standards and Technology (NIST) calibrated seven BCRA tiles and one porcelain tile using 45/0 geometry and a 10 nanometer bandpass. These samples served as reference standards throughout this thesis.

G. Measurement Techniques

The tele-radiometer used with the goniospectrophotometer measures the radiance* of a sample viewed. Radiance would be constant for all viewing angles of an irradiated lambertian surface. Since radiance measurements are not sensitive to the viewing angle for lambertian surfaces, the goniospectrophotometer can be calibrated at one viewing angle and used to measure reflectance factors at other viewing angles. Any deviations in the radiance measurements at other viewing angles indicate a sample surface is non-lambertian.

* "The radiance L is the flux per unit projected area per unit solid angle leaving a source or, in general, any reference surface."[15]

Radiance values are converted to reflectance factors using the following equation:

$$Ri = \frac{Mi}{Ms} Rs$$

where

Mi = measured radiometer reading of the sample
Ms = measured radiometer reading of the standard
Ri = spectral reflectance factor of the sample
Rs = spectral reflectance factor of the standard

As a sample is rotated from 0 degrees the illuminated sample area increases by a factor of $1/\cos\phi$. Measurements at illumination angles other than 0 degrees must be multiplied by this factor to correct for the decreased radiance of area viewed. This correction is shown in the following equation:

$$Ri = \frac{Mi}{Ms} Rs \frac{1}{\cos\phi}$$

where

Mi = measured radiometer reading of the sample
Ms = measured radiometer reading of the standard
Ri = spectral reflectance factor of the sample
Rs = spectral reflectance factor of the standard
ϕ = the illuminating angle

H. Computer Systems

To maintain compatibility with the laboratory's Digital Equipment Corporation's (DEC) LSI-11/73 computer system, a Heath H11 computer containing a DEC LSI-11/02 processor, was selected to control the goniospectrophotometer. The H11 computer utilized the RT-11 operating system and could execute programs written in Oregon Software's Pascal-2 compiler when compiled on the main laboratory system. The system consisted of a Heath H11 chassis, LSI-11/02 processor card with a KEV-11 floating point chip, two serial interfaces, a DEC DRV11-J parallel interface, 56KB of memory, a Data Technology 10MB fixed disk drive and controller, a Shugart 801 8 inch floppy drive, and a DEC VT52 terminal.

The parallel interface card was used to control the radiometer, stepping motor controllers, and filter-switching solenoid. In some cases solid-state relays were controlled by the parallel interface which, in turn, controlled particular devices.

The H11 computer and main laboratory computer were connected via serial interfaces allowing data to be transferred to the main computer after collection.

I. Goniospectrophotometry Software

The goniospectrophotometry program controls all automatic operations of the data collection process. The computer was programmed to control the wavelength drive of the monochromator, the position of the gonio arm, and the selection of the blue-green or red-orange cutoff filters. The computer was able to read the value present on the radiometer data port, however, the radiometer's range switch was manually operated. The program would stop and instruct the operator to increase or decrease the meter's range switch setting when values indicated an over-range condition for the meter, or when an additional digit of data could be obtained by switching to the next lower setting.

Spectral data could be collected, displayed, and saved for a specified geometry and sample. The system also operated in an abridged goniophotometer mode allowing a range of collection angles to be selected and measured for one wavelength. The main menu and a program parameters menu of this software are shown in Figure 6.

J. Plot Program

The plot program can read data from binary spectral data files and ascii text files, solicit operator input for the names of various axis labels and scaling, and

```
Munsell Color Science Laboratory Spectrogoniophotometer Version 4.0




        C: Spectrogoniophotometer Mode Data Collection
        G: Goniophotometer Mode (White Light) Data Collection
        R: Retrieve Data from Disk
        D: DISPLAY values from Photometer
        W: test WAVELENGTH stepping motor
        A: test GONIO ARM stepping motor
        P: change program Parameters
        X: EXIT program


Select C,G,R,D,W,A,P or X :
```

```
Munsell Color Science Laboratory Spectrogoniophotometer Version 4.0
            Program Parameters                Current Value      Default Value

W:                  Wavelength Delay                    350                350
A:                   Gonio Arm Delay                    200                200
N:          Number of Reads Averaged                     10                 10
S:                 Delay after steps                  5000               5000
R:            Ramped Number of Steps                     5                  5
M:             Delay Ramp Multiplier                     3                  3
T:              Radiometer Tolerance               0.00100            0.00100
C:       White calibration reference     DKO:HALONA.REF     DKO:HALONA.REF
I:              Wavelength Increment                     5                  5
1:               Starting wavelength                   380                380
2:                 Ending wavelength                   780                780
3:                     Starting scale                    0                  0
4:          Rewind wavelength at EOR                  TRUE               TRUE
5:    Delay between radiometer reads                   100                100
6:                Debug Mode Enabled                 FALSE              FALSE

D:  change all parameters to their default values

Select W,A,N,S,R,M,T,C,I,1..6,D or Q(uit :
```

Figure 6. Gonio Program Menus
Main Menu (top)
Program Parameters Menu (bottom)

produce plots on a Hewlett Packard HP7475A plotter.

Data points can be unconnected, connected with lines, or connected by a smooth curve using a taut cubic interpolation routine[16]. The tautness, lack of flexibility of the curve, is controlled by a parameter called gamma. The main menu and scale selection menu of this software are shown in Figure 7, and a plot produced by this software is shown in Figure 8.

K. Convert Program

The convert program permits data to be transferred from a binary format to an ascii format. The binary format is very compact and efficient for storage on the computer system, but it is not readable as text, whereas a standard ascii coded file may be sent to a terminal screen or a printer. The program also converts data in the reverse direction to binary format from ascii text format.

The program also calculates the mean and standard deviation of several spectral data files, and performs curve smoothing[17] on spectral data. The main menu of this software is shown in Figure 9.

L. First Color Program

The color program reads data files from the gonio-spectrophotometer, requests an operator to select the desired light source and observer functions, and prints

```
            RIT Munsell Color Science Laboratory Plotting Package

File  1  "STU:h35.24" :   H35.24
File  2  "STU:h40.24" :   H40.224
File  3  "STU:h45.24" :   H45.24
File  4  "STU:h50.24" :   H50.24
File  5  "STU:h55.24" :   H55.24


  Current File Format :  COLOR PROGRAM Data

Data types that can be plotted with this program


          C:  Color data file format
          O:  Optronics data file format
          X:  X,Y data (with description on first line)


enter type of data files you will plot (C,O,X) [C] :
```

```
            RIT Munsell Color Science Laboratory Plotting Package


File            X                          Y              Description
 No.  Minimum      Maximum      Minimum      Maximum
  1    380.0000     760.0000       0.0000       1.0394  H35.24
  2    380.0000     760.0000       0.0000       1.0126  H40.224
  3    380.0000     760.0000       0.0000       1.0059  H45.24
  4    380.0000     760.0000       0.0000       0.9891  H50.24
  5    380.0000     760.0000       0.0000       0.9723  H55.24
 20    380.0000     760.0000       0.0000       1.0394  OVERALL Min/Max of ab
 30  Enter your own Min/Max values for X and Y
 40  Enter your own Min/Max values for X {use OVERALL for Y}
 50  Enter your own Min/Max values for Y {use OVERALL for X}
 60  Use  0..1   for Y {OVERALL for X}
 70  Use  0..100 for Y {OVERALL for X}

Select one of the above :
```

Figure 7. Plotting Program Menus
Input File Selection Menu (top)
Scale Selection Menu (bottom)

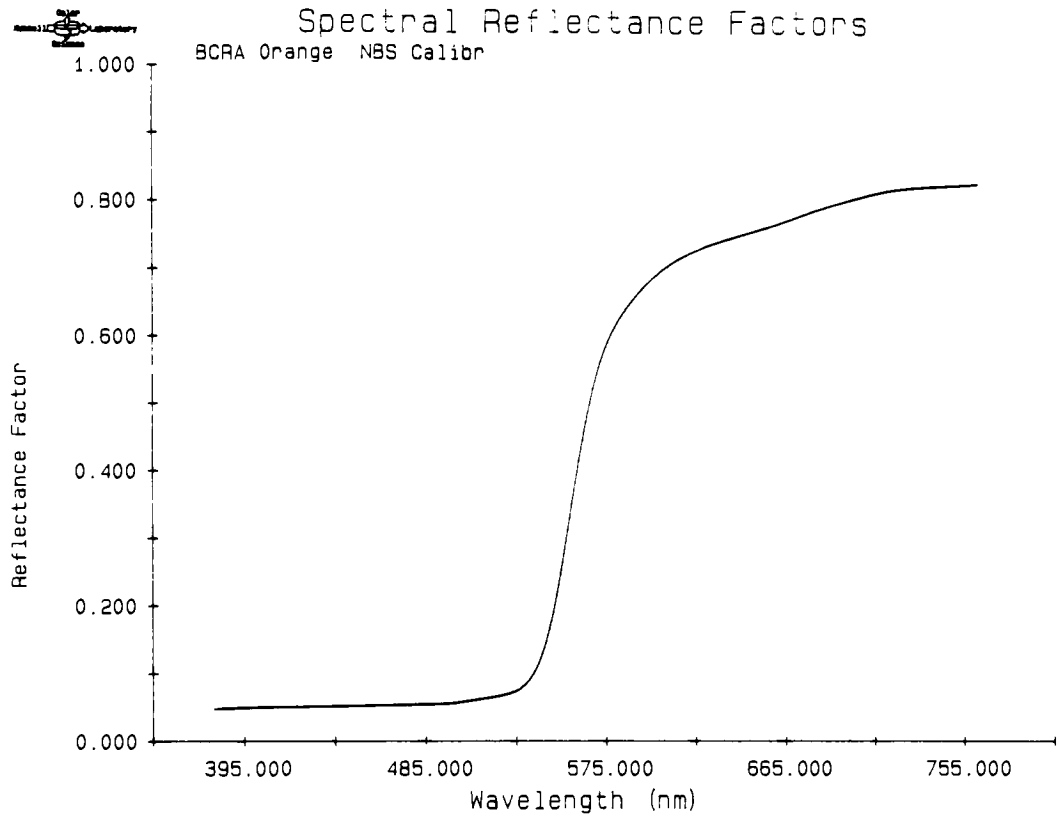


Figure 8. Plot Produced by the Plotting Program
Data Collected by Dr. Jack Hsia of NIST

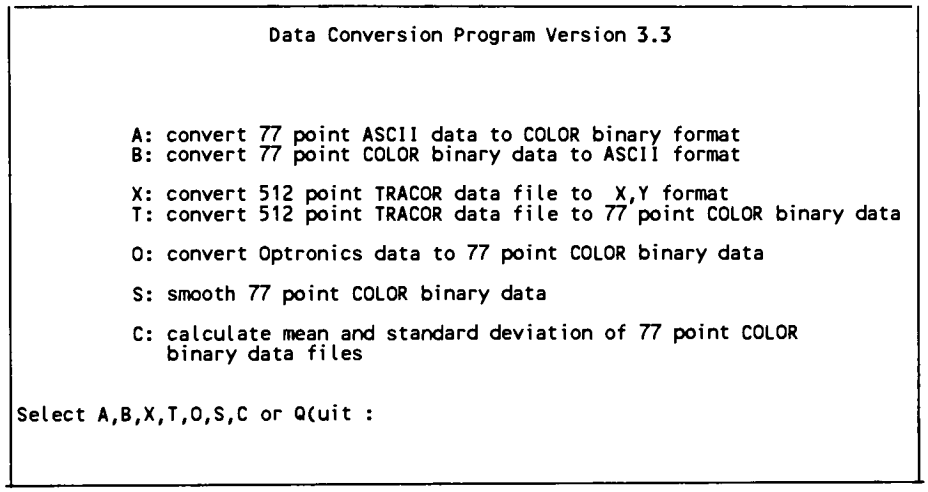


Figure 9. Data Conversion Program Menu

a report of the desired calculations on the terminal screen or printer. Spectral data can be displayed, manually entered, corrected, saved to disk, read from disk, printed and plotted. Other features include the creation of daylight illuminants using characteristic vectors[18], creation of blackbody illuminants using Planck's radiation law[19], calculation of correlated color temperature[23] and distribution temperature[23].

The color program can perform the following calculations:

1. CIE 1931 tristimulus values X,Y, and Z[20]
2. CIE 1931 chromaticity coordinates x,y and z[20]
3. CIE 1931 dominant wavelength and excitation purity[20]
4. CIE 1960 UCS chromaticity coordinates u and v[21]
5. CIE 1976 UCS chromaticity coordinates u' and v'[21]
6. CIE 1976 CIELAB coordinates L*, a*, b*, C*, h, and color differences[21]
7. CIE 1976 CIELUV coordinates L*, u*, v*, C*, h, and color differences[21]

The main and alternate menus, sample screen output for a color calculation report, and a plot produced by the software are shown in Figures 10 through 12.

```
                    Munsell Color Software Version 3.0

E:  enter data file                      T:  calculate D.T. of source
R:  retrieve data file                   U:  calculate C.C.T. of source
C:  calculate chromaticities             D:  display source or sample
L:  CIELab/CIELuv  Color Differences     O:  correct data file
B:  dominant wavelength/ exc. purity     P:  print/plot data file
X:  exit from this program               S:  change to supplemental menu


          select one of the above :
```

```
                    Munsell Color Software Version 3.0

V:  calculate R/B ratio values           G:  spec. radiant excitance calc.
M:  multiply data files                  A:  select 2 or 10 degree observer
K:  create blackbody source              H:  change program options
Z:  zero source and sample arrays        J:  autoprint of calculations
N:  change user name                     S:  change to main menu


          select one of the above :
```

```
                    Munsell Color Software Version 3.0
                   Software Options :

0:  Authorization not needed to EXIT :    TRUE
1:  DEC LA100 printer is the system printer :    TRUE
2:  The system printer is a spooled device :    TRUE
3:  Red/Blue calculations are authorized :    TRUE
4:  Default Devices      System: DAT    Data: STU
5:  Default Extensions   System: DAT    Data: DAT
6:  Printed plots are authorized :    TRUE
7:  Formfeeds enabled :    TRUE

enter authorization code to change values :
```

Figure 10. Original Color Program Menus
Main Menu (top)
Supplemental Menu (middle)
Program Options Menu (bottom)

```
                Munsell Color Software Version 3.0

        Tristimulus Values :X :    0.46801
      02                   Y :    0.35699
                           Z :    0.05885
    Chromaticity Coordinates :x :  0.5295135        u :  0.3120379
                              y :  0.4039031        v :  0.3570251
                           z :  0.0665834
 Sample Description :  BCRA Orange  NBS Calibr.
 Source Description :  CIE ILLUMINANT D65

                        press any key to continue
```

```
                Munsell Color Software Version 3.0

Chromaticity Coordinates :      u' :  0.3120379          v' :  0.5355377
CIELAB Coordinates
    L* :   66.28902              a* :   40.13136              b* :   66.26318
  C*ab :   77.46828             Hab :   58.79940 degrees
CIELUV Coordinates
    L* :   66.28902              u* :   98.41176              v* :   57.92052
  C*uv :  114.19130             Huv :   30.47906 degrees


                        press any key to continue
```

```
                Munsell Color Software Version 3.0

 Chromaticity coordinates :
       Source       x : 0.3127     y : 0.3290
       Sample       x : 0.5295     y : 0.4039

    Dominant Wavelength :  590.59 nm

    Excitation Purity :  81.54 %


 the coordinate information can be printed on the system printer
  P(rint these values -or- any other key to continue (P) :
```

Figure 11. Original Color Program
Color Calculation Report

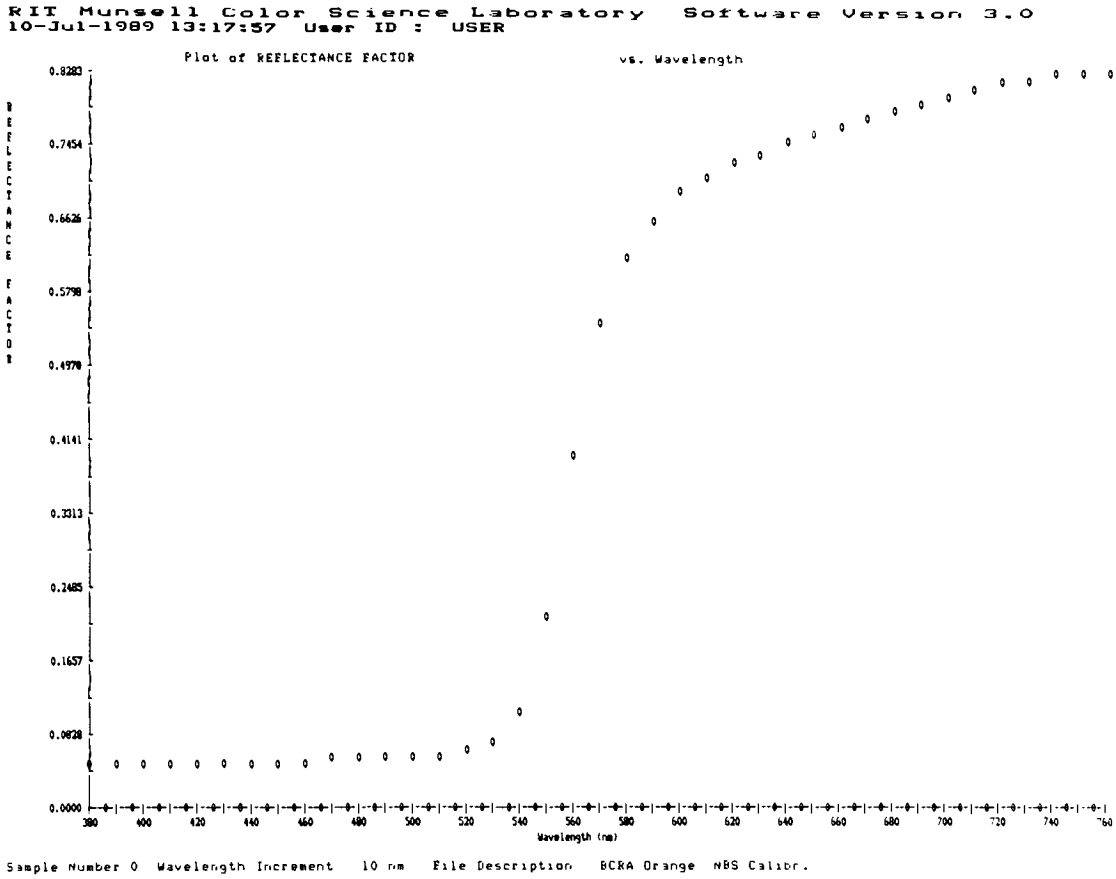


Figure 12. Original Color Program
Spectral Reflectance Factor Plot

M. New Color Program

Before completion of this work, the Munsell Color Science Laboratory replaced their DEC computers and did not have a way to execute any of the above software. Consideration was given to modifying the old color program to work with the VAX/VMS operating system, however, the number of changes required were so extensive that a complete rewrite was performed. The new color program has all the features of the original with many new additions including the following:

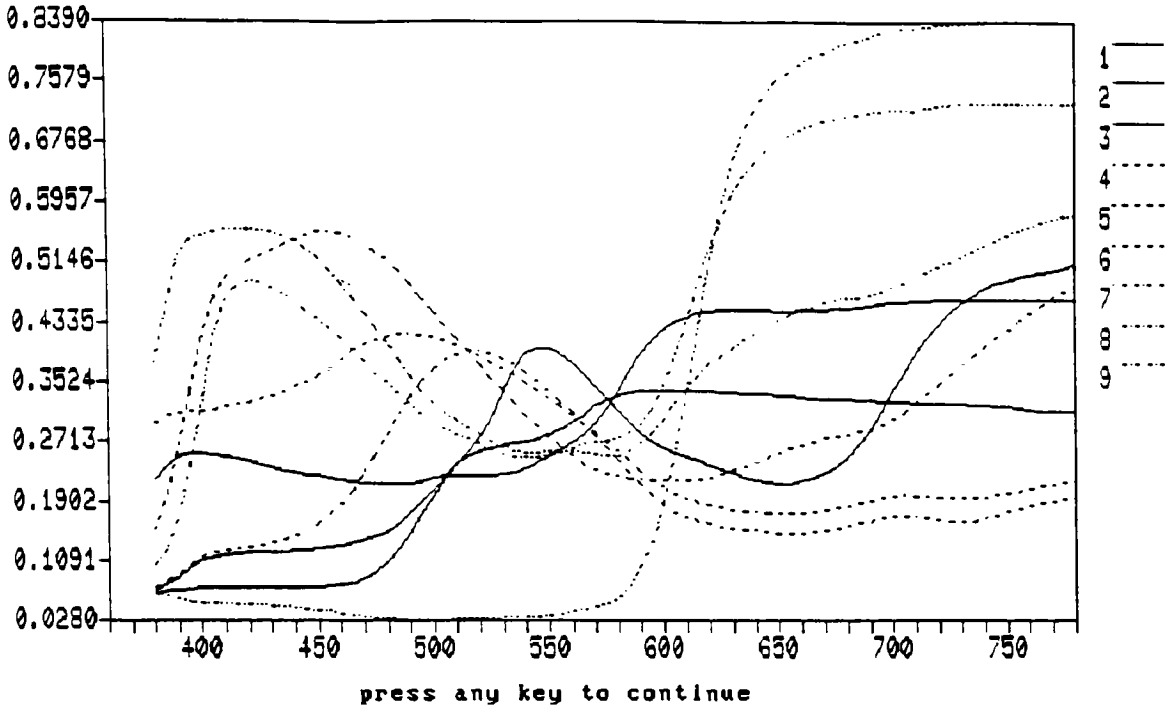
1. ASTM weights[24] were used to calculate the color coordinates of samples under standard illuminants
2. ASTM type weights can be calculated[25] for a light source
3. the color rendering index of a light source[26] can be calculated
4. screen plots of spectral data are available when the software is executed using a terminal that supports REGIS graphics
5. spectral data handling routines were added to allow the folding and truncating of spectral data

Menus and sample output for this software are shown in Figures 13 through 16.

```
                              Color Program V1.0


                              W:  Weights Menu

                              S:  Samples Menu

                              L:  Light Sources Menu

                              F:  File Math Menu

                              C:  Configuration Menu

                              H:  Help Menu

     select one of the above -or- Q(uit [S] :  <_>

                          ENTER AN ACCEPTABLE CHARACTER

**  REMINDER: type a "?" in response to any prompt to return to this menu  **
```

```
                              S A M P L E S   M E N U
No.  1 SPECTRA : Munsell Patches/ 1
No.  2 SPECTRA : Munsell Patches/ 2
No.  3 SPECTRA : Munsell Patches/ 3
No.  4 SPECTRA : Munsell Patches/ 4
No.  5 SPECTRA : Munsell Patches/ 5
No.  6 SPECTRA : Munsell Patches/ 6
No.  7 SPECTRA : Munsell Patches/ 7
No.  8 SPECTRA : Munsell Patches/ 8
No.  9 SPECTRA : Munsell Patches/ 9
No. 10 SPECTRA : Munsell Patches/ 10
ILLUMINANT :  D65
        D(isplay E(nter C(orrect R(ead S(ave P(lot L(ist Z(ero  spectra
                 ---------------------------------------------------
      1: select source for color calc  2: enter coordinates of samples
      3: calculate color of samples    4: calculate color differences
     5: calculate dominant wavelength 6: change display of coordinates

select one of the above -or- Q(uit to the main menu :  <_>

                          ENTER AN ACCEPTABLE CHARACTER
```

Figure 13. New Color Program Menus
Main Menu (top)
Samples Menu (bottom)



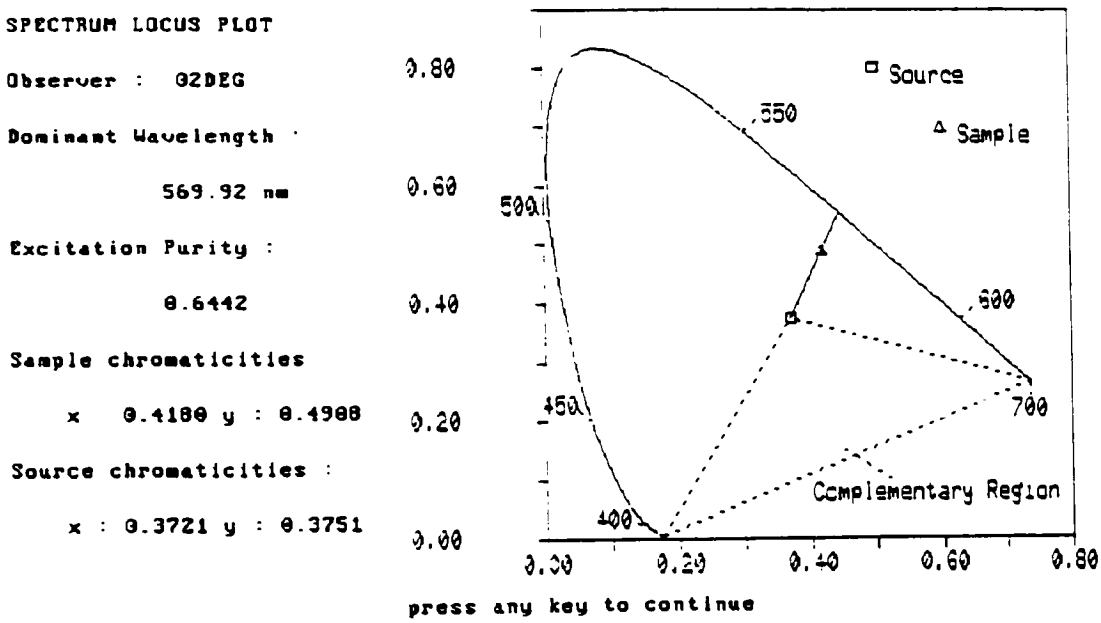
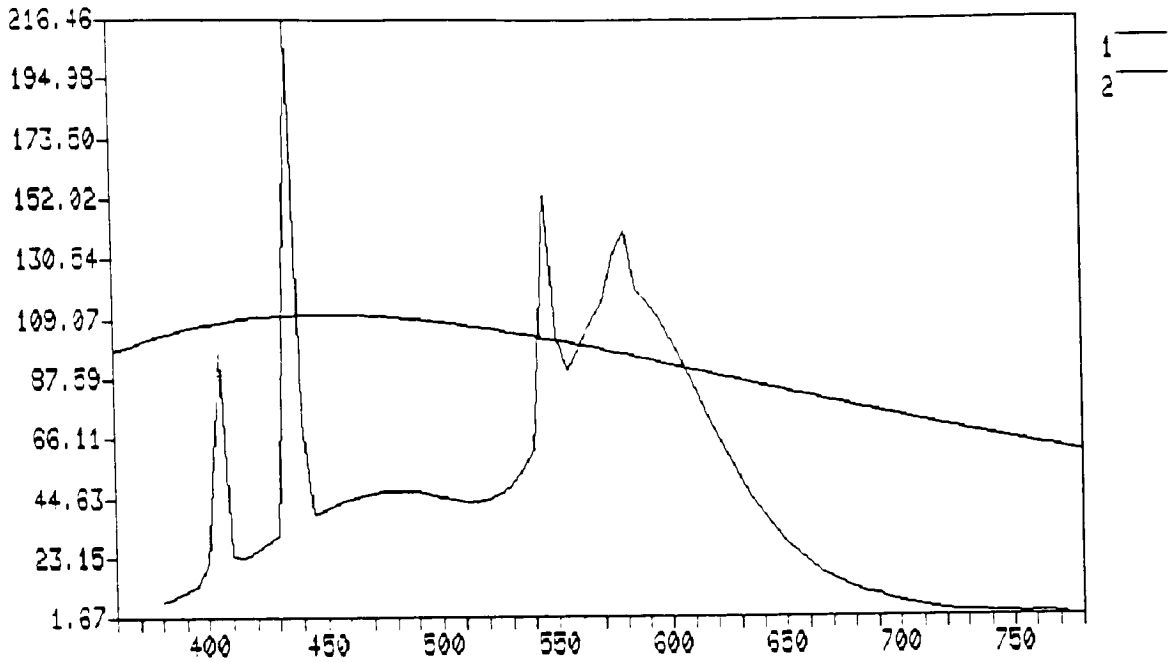


Figure 14. New Color Program Screens
Spectra Reflectance Curves (top)
Dominant Wavelength (bottom)




```
T -  6435.7910K    RMS Error -  3.71300371245986e+01
T    6435.5469K    RMS Error -  3.71300371331111e+01
T =  6435.9131K    RMS Error -  3.71300371308823e+01
T =  6435.7910K    RMS Error -  3.71300371245986e+01
T =  6435.6689K    RMS Error -  3.71300371253411e+01
T -  6435.8521K    RMS Error -  3.71300371268623e+01
T -  6435.7910K    RMS Error -  3.71300371245986e+01
T =  6435.7300K    RMS Error -  3.71300371240915e+01
T =  6435.6689K    RMS Error -  3.71300371253411e+01
T =  6435.7605K    RMS Error -  3.71300371241255e+01
T -  6435.7300K    RMS Error -  3.71300371240915e+01
T -  6435.6995K    RMS Error    3.71300371244967e+01
T =  6435.7452K    RMS Error    3.71300371240536e+01
T =  6435.7300K    RMS Error    3.71300371240915e+01

Distribution Temperature :   6435     K

      Source Description :  CIE Illuminant F2

Your source can be compared to a  6435     K blackbody spectra

Plot the spectra for comparison on your terminal ? [Y] :  <_>
```

Figure 15. New Color Program DT Screens

ILLUMINANT : D65
Sample 1 SPECTRA : Munsell Patches/ 1

Tristimulus Values :

X	: 32.99	Y	: 29.78	Z	: 24.53

Chromaticity Coordinates :

x	: 0.3779	y	: 0.3411	z	: 0.2810
u	: 0.2385	v	: 0.3229	v'	: 0.4844

Dominant Wavelength and Excitation Purity :

DW	: 597.25	Pe	: 0.2123

CIELAB coordinates :

L*	: 61.47	a*	: 17.48	b*	: 11.90
c*_ab:	21.15	h_ab	: 34.23		

CIELUV coordinates :

L*	: 61.47	u*	: 32.51	v*	: 12.90
c*_uv:	34.97	s_uv	: 0.57	h_uv	: 21.65

SPECIAL COLOR RENDERING INDICES
TEST SOURCE : F Illuminants/ F7

Sample Number	Munsell Notation	Color Appearance Under Daylight	Special CRI
1	7.5R 6/4	Light greyish red	89
2	5Y 6/4	Dark greyish yellow	92
3	5GY 6/8	Strong yellow green	91
4	2.5G 6/6	Moderate yellowish green	91
5	10BG 6/4	Light bluish green	90
6	5PB 6/8	Light blue	89
7	2.5P 6/8	Light violet	92
8	10P 6/8	Light reddish purple	87
9	6.5R 4/13	Strong red	61
10	5Y 8/10	Strong yellow	79
11	4.5G 5/8	Strong green	89
12	3PB 3/11	Strong blue	87
13	5YR 8/4	Light yellowish pink	90
14	5GY 4/4	Moderate olive green	94

GENERAL COLOR RENDERING INDEX : 90

Figure 16. New Color Program Screens
Color Calculation Report (top)
Color Rendering Index (bottom)

RESULTS

Data for tele-radiometer measurements at two reciprocal geometries, 0/45, 45/0, 0/30, and 30/0, are shown in Table 1. Reciprocal geometries will result in similar reflectance factor measurements, if the tele-radiometer properly measures radiance.

A hand-pressed sample of barium sulphate was measured photometrically at several collection angles. These data were measured using the silicon detector with white light illumination. The reflectance factors are shown in Figure 17. The values are normalized to 1.0 at 45 degrees to permit comparison with Clarke's data for the reflectance factors of barium sulphate[7].

The spectral reflectance factors of Halon[27] are shown in Figure 18 for a series of geometries from 0/35 to 0/55. These data were measured using the silicon detector, a half-height bandpass of 6.6 nanometers, and the tungsten light source. A BCRA pale gray tile calibrated by NIST was used as the reference standard. The luminous reflectance factors, tristimulus value Y, and the CIE psychometric lightness value, L*, for each geometry, are shown in Figure 19. The spectral reflectance factors of each measurement are shown in Table 2. The CIELAB color coordinates and tristimulus value Y for each geometry are shown in Table 3. All color calculations used CIE Illuminant D65 and the two

Table 1. Comparison of 45/0 and 0/45 Measurements

Illum. Angle*	Viewing Angle*	Measured Value**	1/cos ø factor***	Corrected Value****	Difference between
0	45	11.60	1.000	11.60	0.0
45	0	8.20	1.414	11.6	
0	30	11.87	1.000	11.87	0.02
30	0	10.26	1.155	11.85	

* Illuminating and viewing angles in degrees

** Measured values were read from the Photometer display while measuring hand-pressed barium sulphate at a wavelength of 550 nm using a 6.6 nm bandpass

*** the inverse cosine factors represent the increased sample area illuminated at illumination angles greater than 0 degrees

**** Corrected value = measured value * factor

Table 2. Goniospectrophotometric Properties of Halon (at selected geometries)

Wavelength (nm)	0/35 degrees	0/40 degrees	0/45 degrees	0/50 degrees	0/55 degrees
400	0.999	0.995	0.979	0.968	0.954
420	1.009	1.000	0.986	0.976	0.960
440	0.994	0.986	0.975	0.961	0.950
460	1.021	1.019	0.999	0.989	0.976
480	1.015	1.009	0.994	0.983	0.966
500	1.016	1.004	0.992	0.984	0.970
520	1.015	1.005	0.998	0.987	0.969
540	1.016	1.008	0.996	0.985	0.969
560	1.016	1.008	0.997	0.987	0.972
580	1.017	1.008	0.999	0.987	0.973
600	1.033	1.029	1.014	0.998	0.983
620	1.032	1.024	1.011	0.998	0.985
640	1.031	1.026	1.011	1.000	0.985
660	1.032	1.025	1.014	0.995	0.986
680	1.030	1.019	1.009	0.993	0.977
700	1.024	1.014	1.005	0.990	0.980

NOTES:

1) The above data were measured using a 6.6 nm bandpass following photometric calibration using a pale gray BCRA tile calibrated by NIST.

Table 3. CIELAB Coordinates as a Function of Viewing Angle for Halon

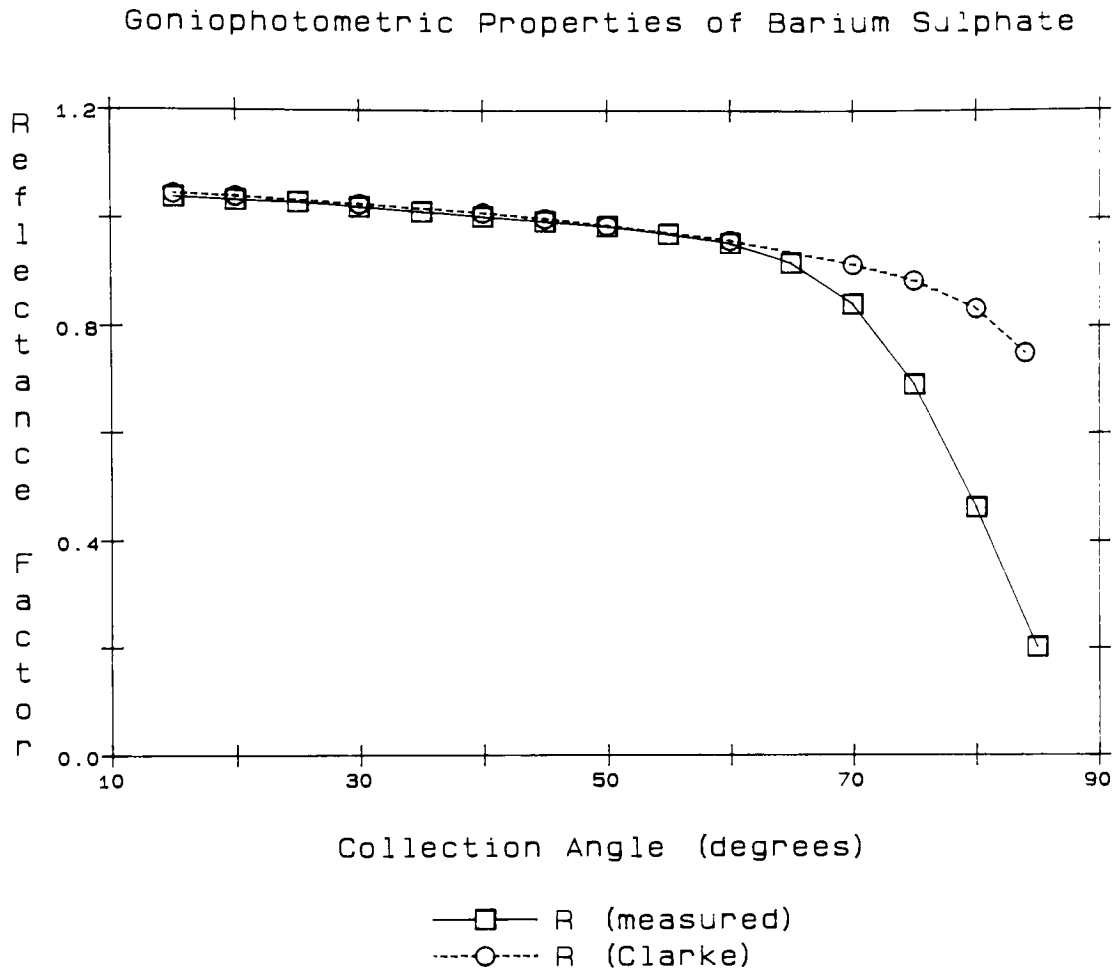
Viewing Angle	Y	L*	a*	b*	dE
35	102.2	100.9	1.38	0.16	0.91
40	101.4	100.5	1.56	0.02	0.78
45	100.0	100.0	1.12	0.40	----
50	99.1	99.7	1.15	0.28	0.37
55	97.6	99.1	1.37	0.16	0.99

NOTES:

1) Illuminating angle was 0 degrees, viewing angles are expressed in degrees.

2) CIELAB coordinates and luminous reflectance, Y, were calculated for CIE Illuminant D65 and the 2 degree observer.

3) CIELAB color differences are calculated between the 0/45 measurement and all other viewing angles.

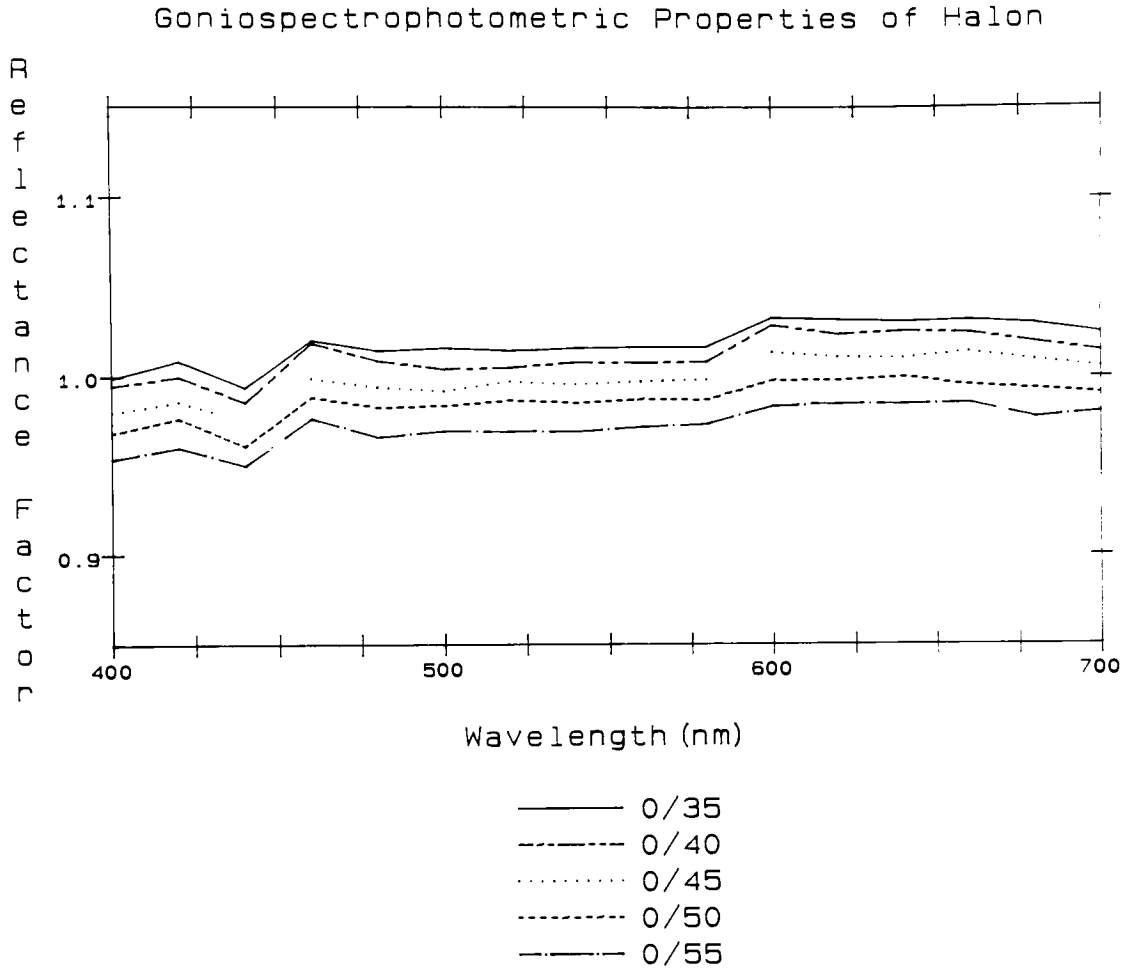


NOTES:

1) Illuminating angle was 0 degrees.

2) Luminous reflectance factors are shown for hand-pressed barium sulphate normalized to a value of 1.0 at a geometry of 0/45 for comparison with Clarke's data[7].

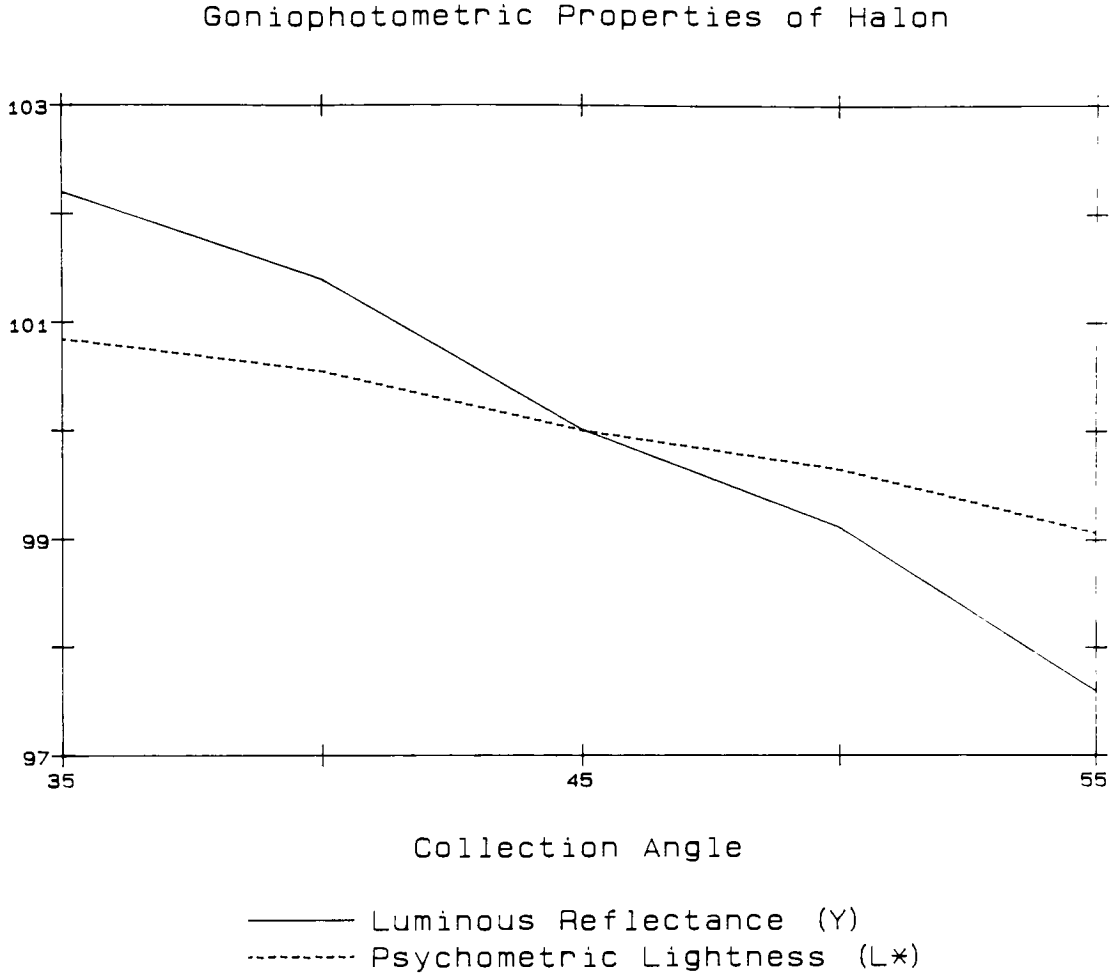
Figure 17. Goniophotometric Properties of Barium Sulphate



NOTES:

1) Spectral reflectance factors are shown for Halon measured using a 6.6 nm bandpass following photometric calibration using a pale gray BCRA tile calibrated by NIST.

Figure 18. Goniospectrophotometric Properties of Halon



NOTES:

1) Illuminating angle was 0 degrees, viewing angles are expressed in degrees.

2) CIELAB coordinates and luminous reflectance, Y, were calculated for CIE Illuminant D65 and the 2 degree observer.

3) CIELAB color differences are calculated between the 0/45 measurement and all other viewing angles.

Figure 19. Goniophotometric Properties of Halon

degree observer.

The 0/45 spectral reflectance factors of five BCRA tiles were measured four times using the PMT detector, a half-height bandpass of 6.6 nanometers, and the xenon arc light source. The average of four measurements for each tile and the standard deviations are shown in Figures 20 through 29. Reflectance factors measured by NIST are plotted with the averaged values for comparison. Spectral reflectance factor averages, NIST calibration values, standard deviations, and the difference between the averages and NIST values, are shown in Tables 4 through 8. CIELAB color coordinates and color differences are shown in Table 9 using CIE Illuminant D65 and the two degree observer.

Table 4. Spectral Reflectance Factors of a Cyan BCRA Tile

Wavelength (nm)	NIST* VALUES	MEASURED** (N=4)	SDEV*** (N=4)	NIST MINUS MEASURED
380	0.167	0.179	0.001	-0.012
390	0.215	0.216	0.001	-0.001
400	0.260	0.261	0.007	-0.002
410	0.298	0.294	0.002	0.004
420	0.332	0.328	0.002	0.004
430	0.363	0.364	0.007	-0.001
440	0.388	0.384	0.001	0.004
450	0.409	0.405	0.002	0.004
460	0.424	0.421	0.002	0.003
470	0.430	0.421	0.002	0.008
480	0.424	0.419	0.003	0.005
490	0.406	0.402	0.001	0.004
500	0.375	0.370	0.001	0.005
510	0.333	0.327	0.002	0.006
520	0.286	0.281	0.001	0.004
530	0.241	0.237	0.002	0.004
540	0.201	0.198	0.001	0.003
550	0.167	0.164	0.001	0.002
560	0.139	0.139	0.001	0.000
570	0.118	0.119	0.001	0.000
580	0.102	0.103	0.000	-0.001
590	0.090	0.093	0.001	-0.003
600	0.081	0.081	0.000	0.000
610	0.074	0.074	0.001	0.000
620	0.070	0.070	0.000	0.000
630	0.068	0.067	0.001	0.000
640	0.067	0.067	0.001	0.000
650	0.067	0.068	0.001	0.000
660	0.070	0.070	0.001	0.000
670	0.074	0.074	0.000	0.000
680	0.080	0.080	0.001	0.000
690	0.088	0.089	0.001	-0.001
700	0.099	0.098	0.001	0.001
710	0.111	0.110	0.001	0.002
720	0.122	0.118	0.002	0.004
730	0.129	0.124	0.003	0.004
740	0.132	0.128	0.000	0.004
750	0.132	0.118	0.002	0.015
760	0.131	0.106	0.014	0.025

* 45/0 measurements by Dr. Jack Hsia of NIST

** Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

*** Standard deviation of the four measurements

Table 5. Spectral Reflectance Factors of a Dark Gray BCRA Tile

Wavelength (nm)	NIST* VALUES	MEASURED** (N=4)	SDEV*** (N=4)	NIST MINUS MEASURED
380	0.045	0.046	0.001	0.000
390	0.046	0.046	0.000	0.000
400	0.047	0.047	0.000	0.000
410	0.047	0.048	0.000	-0.001
420	0.047	0.047	0.000	0.000
430	0.047	0.047	0.000	0.000
440	0.047	0.046	0.001	0.001
450	0.047	0.046	0.001	0.001
460	0.047	0.047	0.001	0.000
470	0.047	0.046	0.001	0.001
480	0.047	0.046	0.001	0.001
490	0.047	0.046	0.001	0.001
500	0.047	0.046	0.001	0.001
510	0.047	0.047	0.000	0.000
520	0.048	0.048	0.001	0.000
530	0.049	0.049	0.001	0.001
540	0.050	0.049	0.001	0.001
550	0.051	0.050	0.000	0.001
560	0.050	0.050	0.001	0.000
570	0.049	0.049	0.000	0.000
580	0.048	0.048	0.001	0.000
590	0.048	0.048	0.001	0.000
600	0.047	0.047	0.001	0.000
610	0.047	0.047	0.000	0.000
620	0.048	0.048	0.000	0.000
630	0.049	0.049	0.001	0.000
640	0.050	0.050	0.001	0.000
650	0.050	0.050	0.000	0.000
660	0.051	0.051	0.000	0.000
670	0.054	0.054	0.000	0.000
680	0.060	0.059	0.000	0.001
690	0.069	0.068	0.001	0.001
700	0.081	0.079	0.001	0.002
710	0.095	0.092	0.001	0.003
720	0.110	0.105	0.003	0.005
730	0.125	0.120	0.004	0.005
740	0.140	0.136	0.006	0.005
750	0.155	0.132	0.012	0.023
760	0.166	0.117	0.020	0.049

* 45/0 measurements by Dr. Jack Hsia of NIST

** Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

*** Standard deviation of the four measurements

Table 6. Spectral Reflectance Factors of a Deep Pink BCRA Tile

Wavelength (nm)	NIST* VALUES	MEASURED** (N=4)	SDEV*** (N=4)	NIST MINUS MEASURED
380	0.146	0.140	0.002	0.006
390	0.145	0.141	0.003	0.005
400	0.142	0.138	0.003	0.004
410	0.136	0.134	0.003	0.002
420	0.127	0.126	0.002	0.001
430	0.117	0.115	0.001	0.002
440	0.106	0.105	0.001	0.002
450	0.097	0.095	0.001	0.002
460	0.088	0.087	0.001	0.001
470	0.080	0.079	0.001	0.001
480	0.074	0.073	0.001	0.001
490	0.069	0.068	0.001	0.001
500	0.065	0.064	0.001	0.001
510	0.064	0.063	0.001	0.001
520	0.064	0.063	0.001	0.001
530	0.066	0.065	0.001	0.001
540	0.071	0.070	0.002	0.001
550	0.079	0.077	0.001	0.002
560	0.090	0.088	0.001	0.002
570	0.104	0.101	0.002	0.003
580	0.122	0.118	0.002	0.004
590	0.146	0.139	0.002	0.007
600	0.175	0.169	0.004	0.007
610	0.209	0.202	0.003	0.007
620	0.247	0.238	0.002	0.009
630	0.288	0.277	0.003	0.010
640	0.329	0.315	0.004	0.014
650	0.368	0.357	0.002	0.011
660	0.405	0.393	0.004	0.012
670	0.440	0.429	0.002	0.012
680	0.469	0.457	0.003	0.013
690	0.492	0.483	0.007	0.009
700	0.511	0.495	0.001	0.016
710	0.527	0.510	0.006	0.017
720	0.538	0.514	0.002	0.024
730	0.544	0.524	0.005	0.020
740	0.547	0.521	0.007	0.026
750	0.547	0.487	0.008	0.061
760	0.545	0.401	0.011	0.144

* 45/0 measurements by Dr. Jack Hsia of NIST

** Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

*** Standard deviation of the four measurements

Table 7. Spectral Reflectance Factors of a Green BCRA Tile

Wavelength (nm)	NIST* VALUES	MEASURED** (N=4)	SDEV*** (N=4)	NIST MINUS MEASURED
380	0.055	0.065	0.000	-0.010
390	0.061	0.065	0.001	-0.004
400	0.067	0.069	0.000	-0.002
410	0.071	0.072	0.000	-0.001
420	0.076	0.076	0.001	0.000
430	0.082	0.085	0.001	-0.002
440	0.092	0.091	0.002	0.001
450	0.104	0.104	0.000	0.000
460	0.124	0.124	0.001	0.000
470	0.153	0.149	0.001	0.003
480	0.193	0.191	0.001	0.002
490	0.242	0.239	0.004	0.004
500	0.291	0.290	0.002	0.001
510	0.317	0.316	0.001	0.001
520	0.310	0.308	0.002	0.002
530	0.282	0.281	0.003	0.002
540	0.246	0.244	0.002	0.002
550	0.211	0.209	0.001	0.002
560	0.181	0.180	0.001	0.001
570	0.157	0.157	0.002	0.001
580	0.139	0.141	0.003	-0.003
590	0.124	0.124	0.001	-0.001
600	0.113	0.113	0.001	0.000
610	0.105	0.105	0.001	0.000
620	0.100	0.099	0.001	0.000
630	0.096	0.096	0.001	0.000
640	0.095	0.096	0.001	0.000
650	0.096	0.097	0.001	0.000
660	0.099	0.099	0.000	0.000
670	0.104	0.105	0.000	-0.001
680	0.111	0.112	0.001	0.000
690	0.121	0.133	0.018	-0.011
700	0.134	0.133	0.001	0.001
710	0.149	0.148	0.000	0.001
720	0.161	0.159	0.002	0.002
730	0.169	0.167	0.004	0.002
740	0.173	0.169	0.004	0.004
750	0.174	0.167	0.012	0.007
760	0.173	0.146	0.016	0.026

* 45/0 measurements by Dr. Jack Hsia of NIST

** Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

*** Standard deviation of the four measurements

Table 8. Spectral Reflectance Factors of a Orange BCRA Tile

Wavelength (nm)	NIST* VALUES	MEASURED** (N=4)	SDEV*** (N=4)	NIST MINUS MEASURED
380	0.048	0.051	0.000	-0.003
390	0.049	0.050	0.000	0.000
400	0.050	0.050	0.000	0.000
410	0.051	0.050	0.000	0.001
420	0.051	0.050	0.001	0.001
430	0.052	0.050	0.001	0.002
440	0.052	0.051	0.000	0.001
450	0.053	0.051	0.000	0.001
460	0.053	0.052	0.000	0.001
470	0.054	0.053	0.000	0.001
480	0.055	0.053	0.000	0.001
490	0.055	0.054	0.000	0.002
500	0.057	0.055	0.000	0.002
510	0.062	0.060	0.000	0.002
520	0.067	0.064	0.001	0.002
530	0.075	0.071	0.001	0.004
540	0.108	0.098	0.001	0.011
550	0.215	0.190	0.002	0.025
560	0.399	0.374	0.003	0.025
570	0.545	0.524	0.009	0.021
580	0.620	0.604	0.002	0.016
590	0.662	0.639	0.006	0.023
600	0.693	0.685	0.003	0.007
610	0.714	0.706	0.005	0.009
620	0.729	0.718	0.002	0.011
630	0.741	0.732	0.004	0.009
640	0.750	0.742	0.001	0.008
650	0.759	0.751	0.002	0.008
660	0.768	0.761	0.003	0.007
670	0.779	0.774	0.002	0.005
680	0.790	0.782	0.003	0.008
690	0.799	0.791	0.002	0.008
700	0.807	0.794	0.002	0.013
710	0.815	0.804	0.005	0.011
720	0.820	0.809	0.008	0.011
730	0.823	0.815	0.005	0.008
740	0.825	0.818	0.013	0.007
750	0.826	0.818	0.000	0.009
760	0.828	0.772	0.014	0.056

* 45/0 measurements by Dr. Jack Hsia of NIST

** Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

*** Standard deviation of the four measurements

Table 9. CIELAB Color Differences of BCRA Tile Measurements

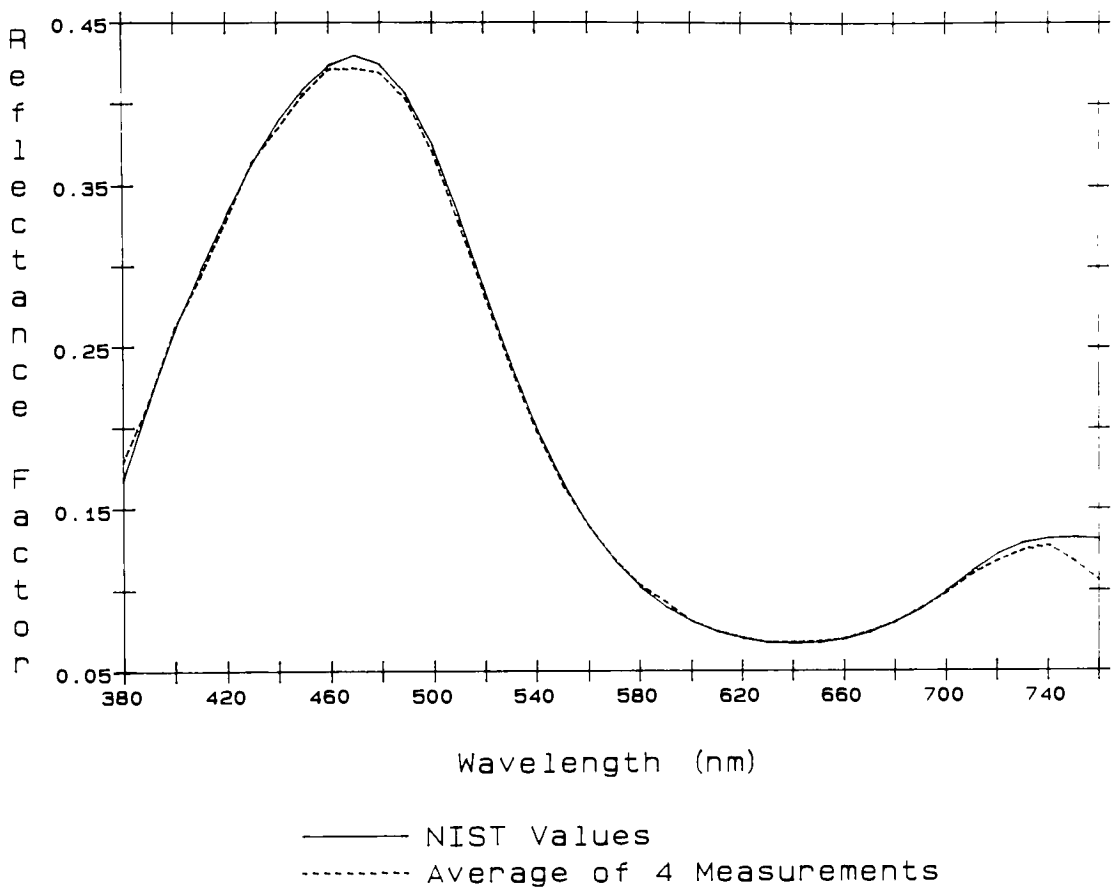
BCRA TILE	MEASURED BY	L*	a*	b*	dE
CYAN	NIST	49.74	-12.88	-33.47	
	RIT	49.54	-12.37	-33.35	0.57
DARK GRAY	NIST	26.38	-0.46	0.94	
	RIT	26.26	-0.25	0.96	0.23
DEEP PINK	NIST	40.43	30.66	6.26	
	RIT	39.88	30.03	5.75	0.98
GREEN	NIST	51.38	-35.48	14.56	
	RIT	51.28	-35.12	12.59	0.34
ORANGE	NIST	66.29	40.13	66.26	
	RIT	65.31	41.21	65.27	1.77

NOTES:

1) 45/0 measurements by Dr. Jack Hsia of NIST

2) RIT reflectance factors measured using 0/45 geometry and 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

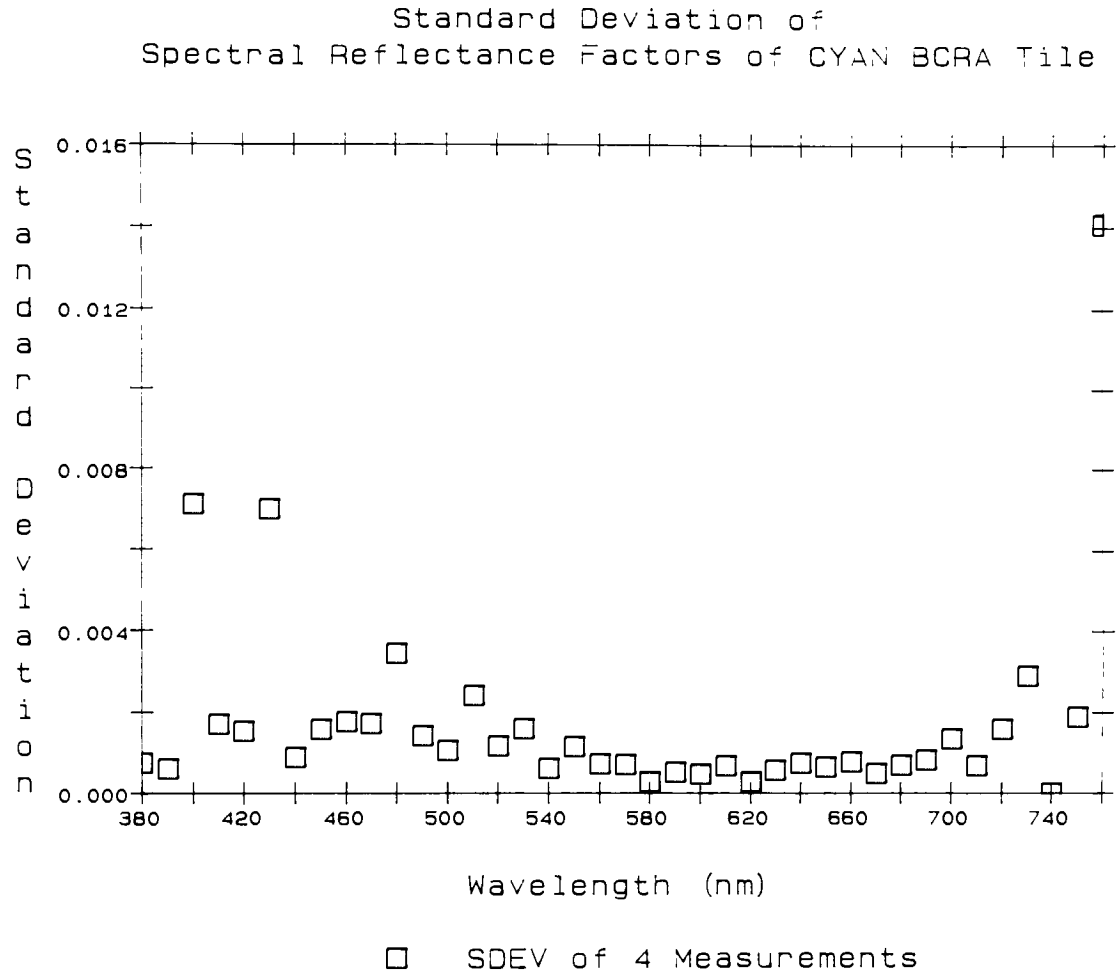
3) Color coordinates calculated using CIE Illuminant D65 and the 2 degree observer



NOTES:

1) NIST data measured by Dr. Jack Hsia
2) Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

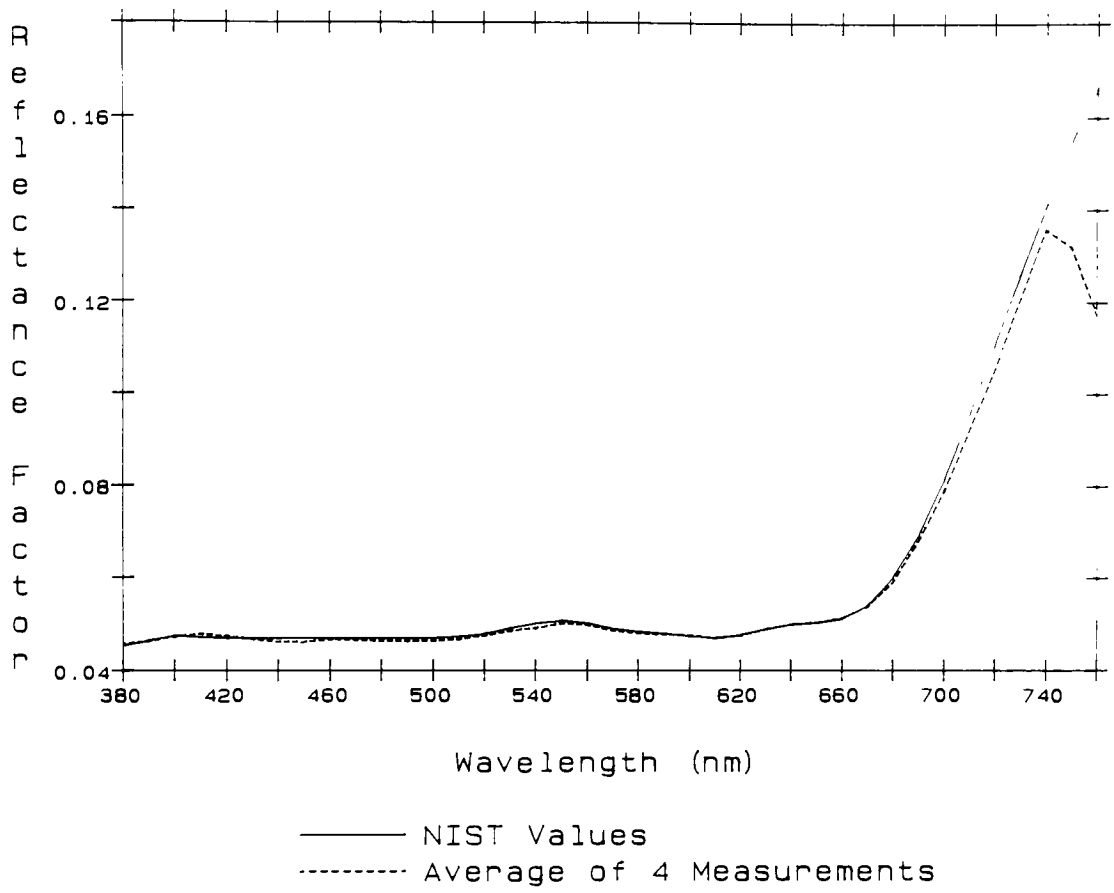
Figure 20. Spectral Reflectance Factors of a Cyan BCRA Tile



NOTES:

1) Standard deviations of 4 measurements

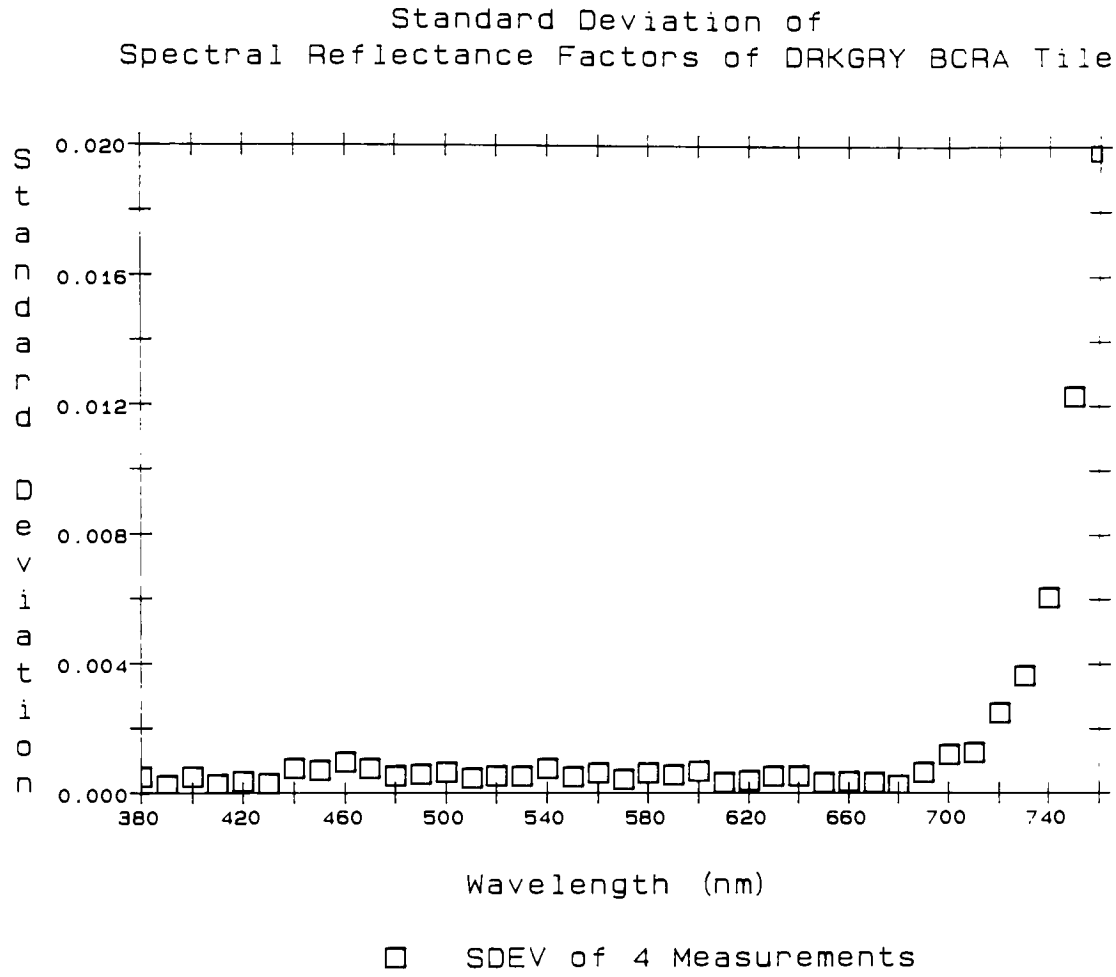
Figure 21. Standard Deviation of Spectral Reflectance Factors of a Cyan BCRA Tile



NOTES:

1) NIST data measured by Dr. Jack Hsia
2) Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

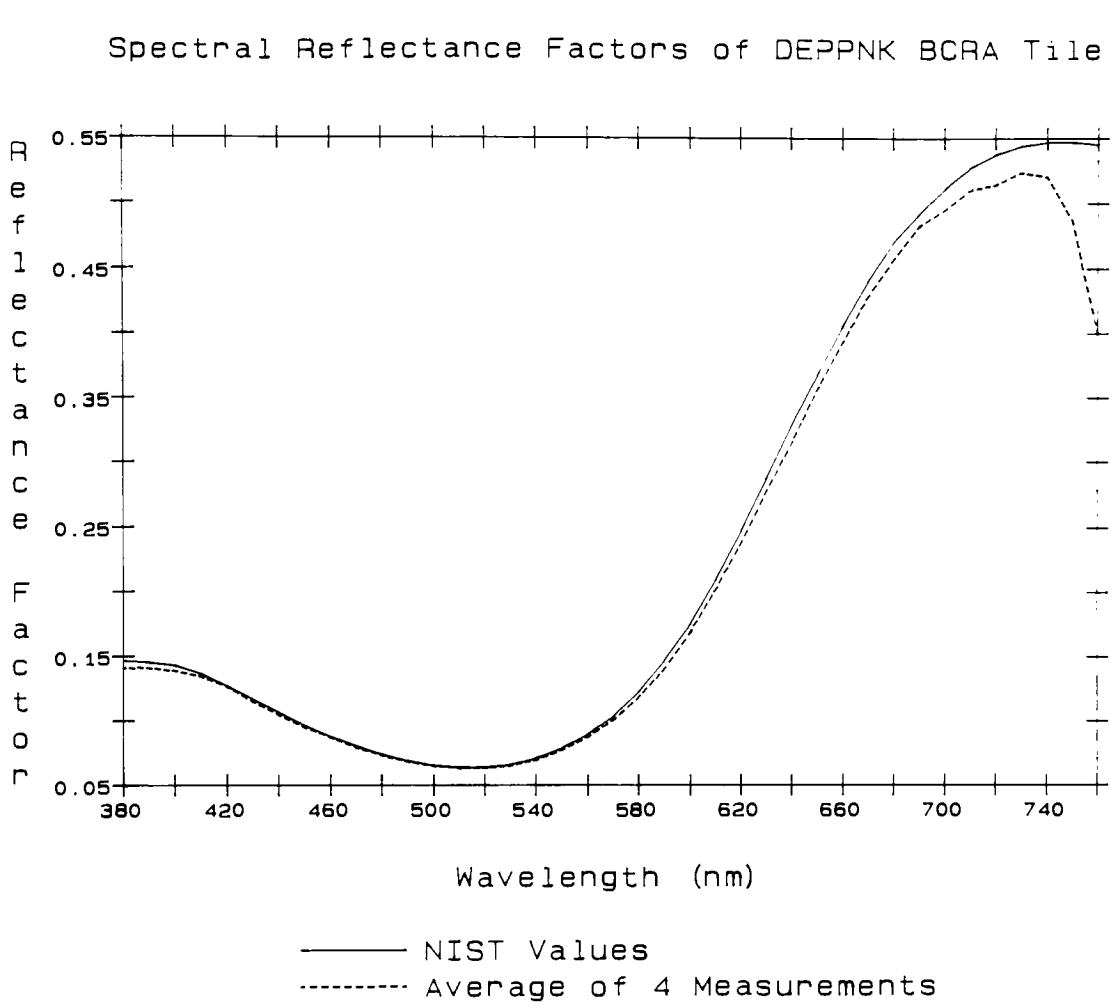
Figure 22. Spectral Reflectance Factors of a Dark Gray BCRA Tile



NOTES:

1) Standard deviations of 4 measurements

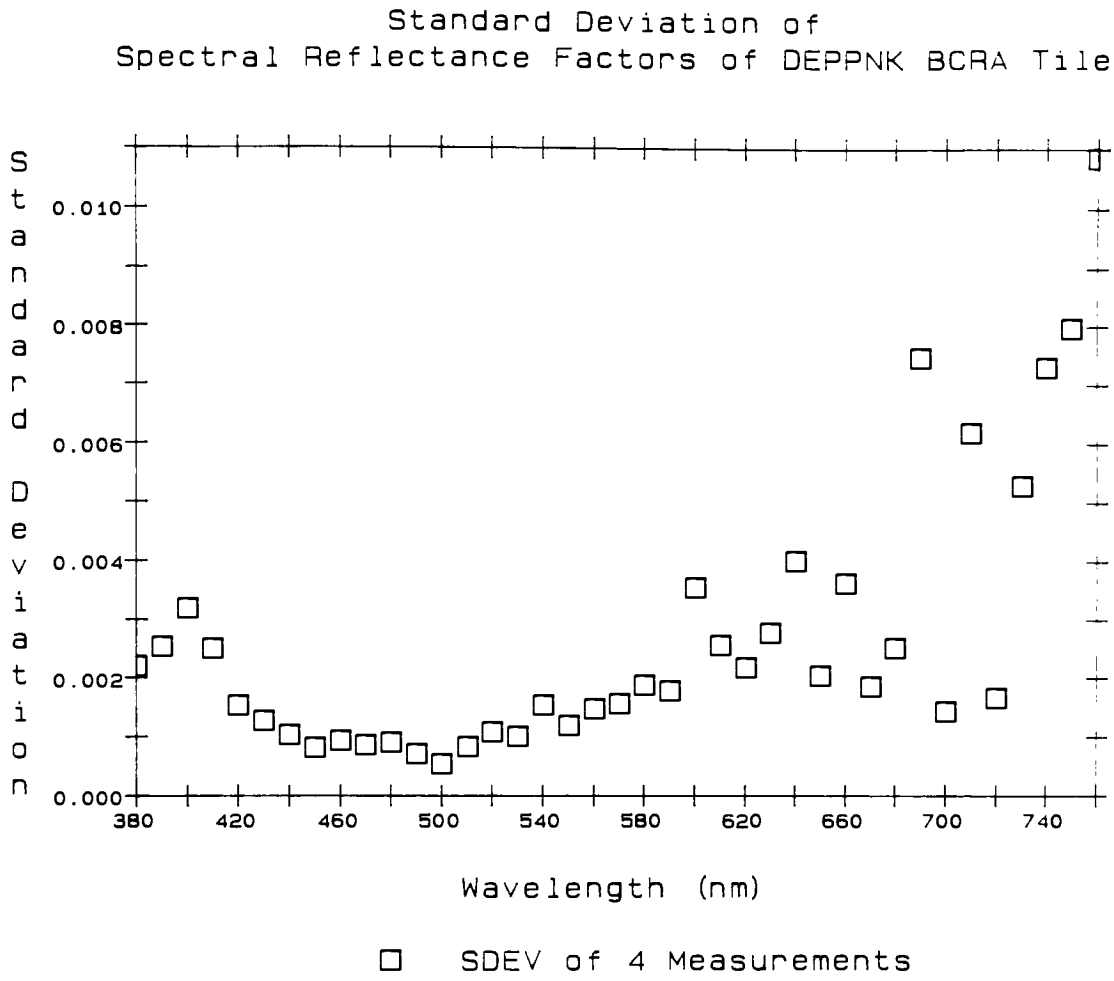
Figure 23. Standard Deviation of Spectral Reflectance Factors of a Dark Gray BCRA Tile



NOTES:

1) NIST data measured by Dr. Jack Hsia
2) Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

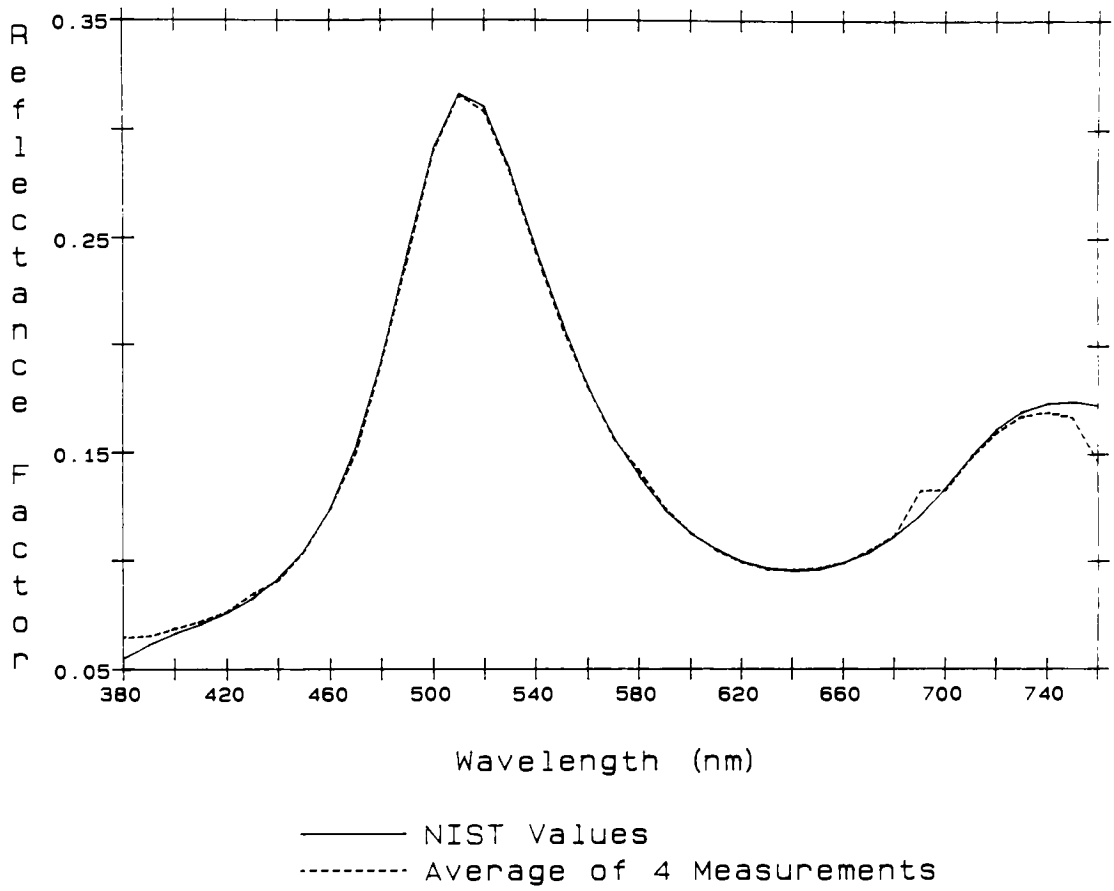
Figure 24. Spectral Reflectance Factors of a Deep Pink BCRA Tile



NOTES:

1) Standard deviations of 4 measurements

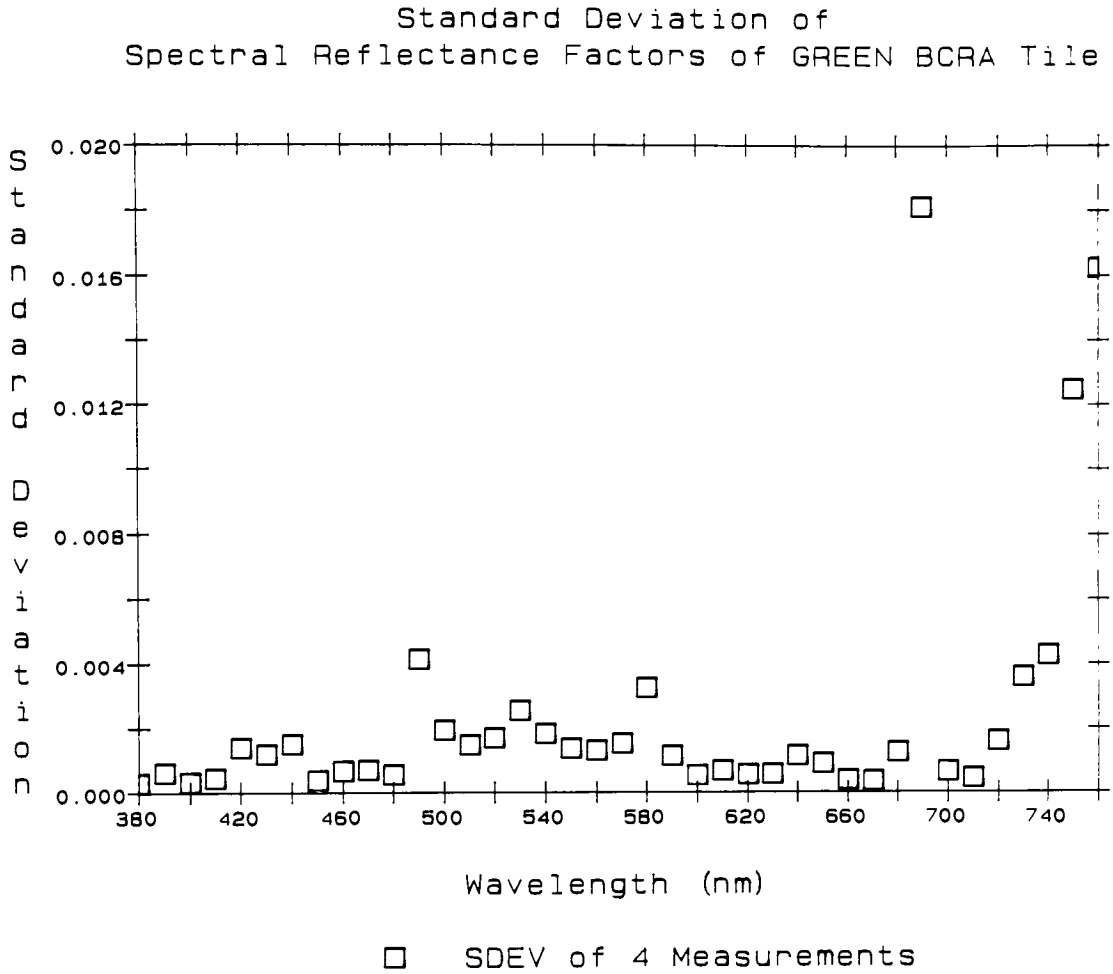
Figure 25. Standard Deviation of Spectral Reflectance Factors of a Deep Pink BCRA Tile



NOTES:

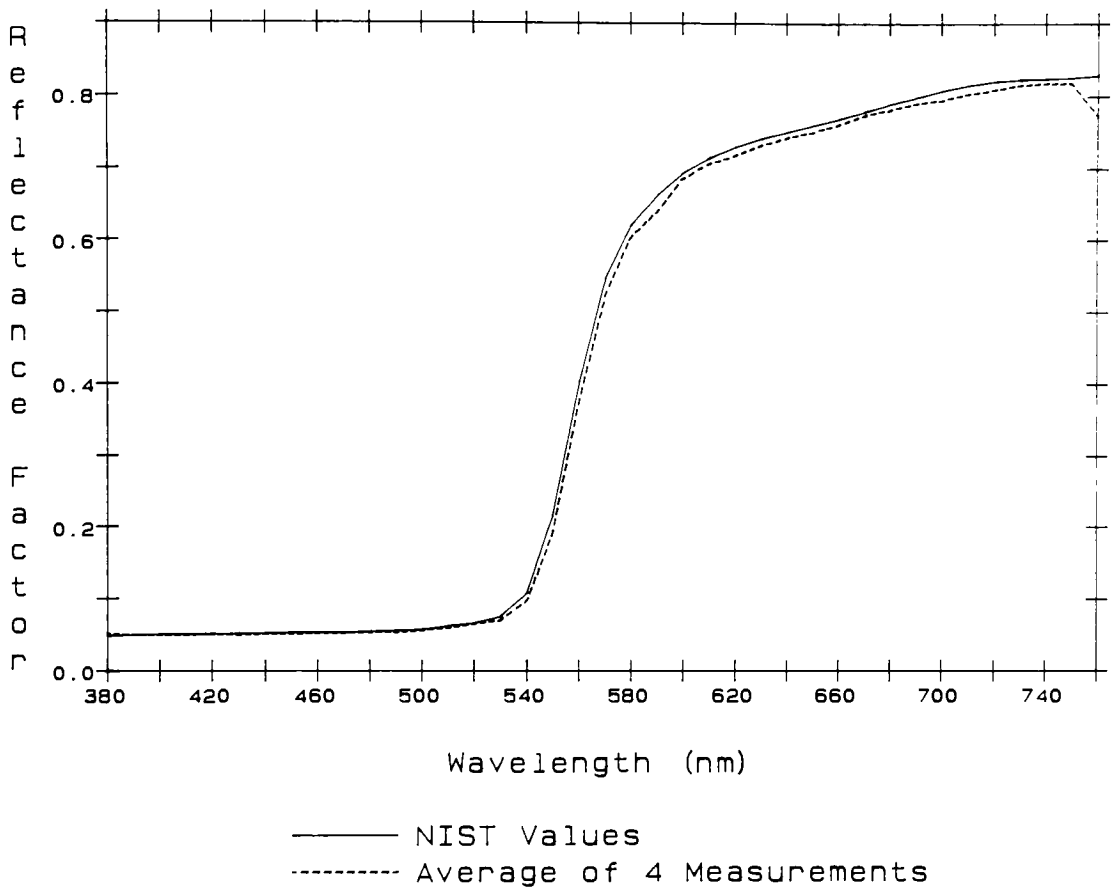
1) NIST data measured by Dr. Jack Hsia
2) Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

Figure 26. Spectral Reflectance Factors of a Green BCRA Tile



NOTES:
1) Standard deviations of 4 measurements

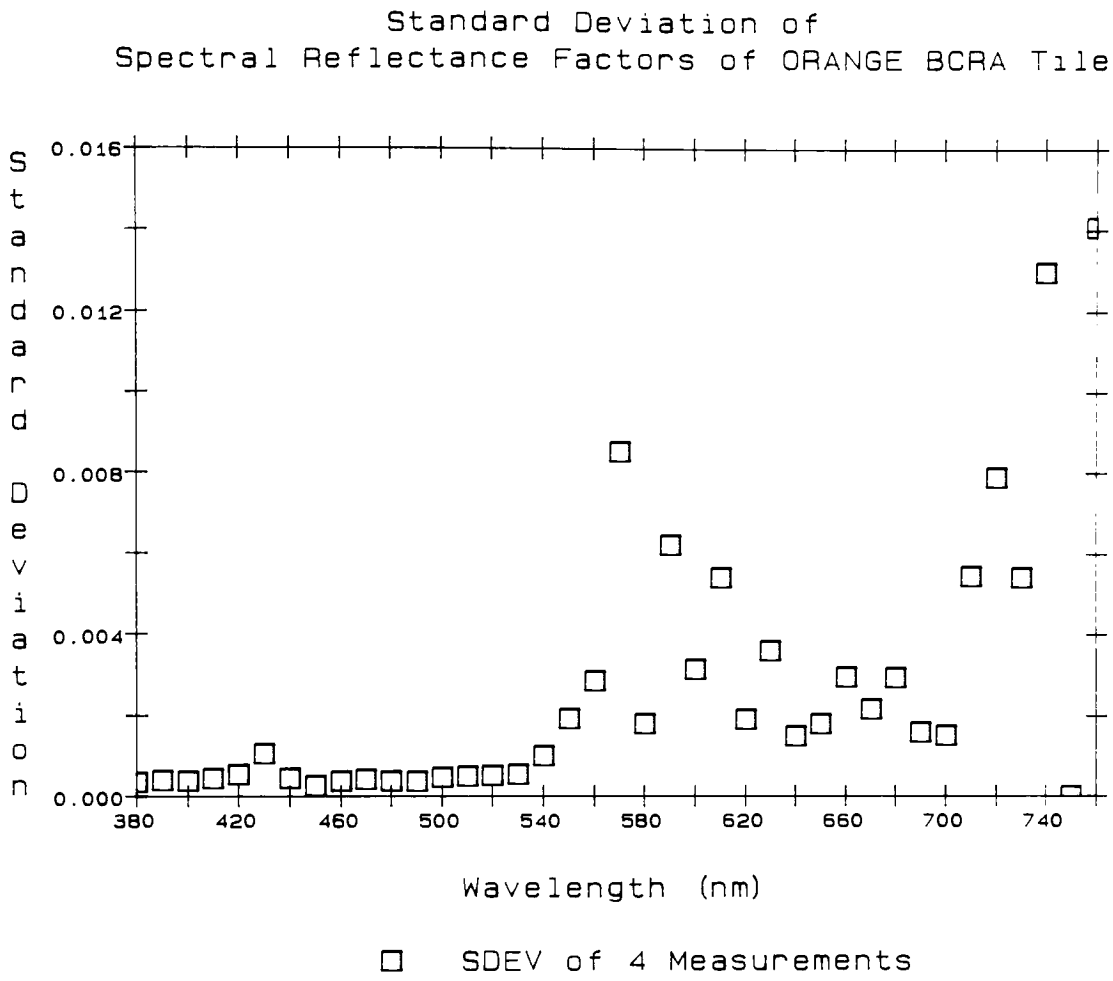
Figure 27. Standard Deviation of Spectral Reflectance Factors of a Green BCRA Tile



NOTES:

1) NIST data measured by Dr. Jack Hsia
2) Average of 4 spectral reflectance factor measurements using a 6.6 nm bandpass after photometric calibration to a pale gray BCRA tile calibrated by NIST

Figure 28. Spectral Reflectance Factors of a Orange BCRA Tile



NOTES:

1) Standard deviations of 4 measurements

Figure 29. Standard Deviation of Spectral Reflectance Factors of a Orange BCRA Tile

DISCUSSION

The cosine factors (shown in Table 1) correct measured radiance values for the increased area illuminated when illuminating angles are increased from zero degrees. Differences between corrected measurements of reciprocal geometries were minor, indicating the cosine factors accurately accounted for the decreased radiance values measured at non-zero illuminating angles.

Clarke's data[7] was used to confirm the radiometer operation. As shown in Figure 17, the measured luminous reflectance factors overlay Clarke's data over the range of 15 to 60 degrees. Beyond 60 degrees the measured reflectance factors decrease more rapidly than Clarke's values because the tele-radiometer measured beyond the illuminated sample area*. As the viewing angle is increased with the illuminating angle held constant at 0 degrees, the viewed sample area increases. The differences observed would have been greatly or completely reduced if the illuminated sample area was increased from one square centimeter to a much larger area. As the illuminated sample area increases the power per square centimeter decreases proportionately. Since viewing angles over 60 degrees were

* The surface characteristics of barium sulphate samples measured by Clarke could have been different than those measured during this thesis, however, one could observe the measuring aperture viewing more than illuminated sample area during measurement at large viewing angles.

not going to be used and increasing the illuminated sample area would decrease the radiance measured for all samples, the illuminated area was not increased from one square centimeter. Some differences between Clarke's data and the measured

Spectral reflectance factors of Halon decrease as the viewing angle is increased, as shown in Figure 18. Patterns observed in the data near 440 and 460 nanometers may have resulted from radiometer non-linearities. The rise in value at 600 nanometers coincides with the change from the cyan filter to orange-red filter. The radiometer range switch was changed from one scale setting to the next near these wavelengths. The above patterns were not observed when measuring the spectral reflectance of the sample on a Milton Roy Match Scan II spectrophotometer, and Grum's data[27] does not show the patterns. Reflectance factors calculated from these data show a similar relationship with geometry changes as reported by Grum[27] and Fairchild[6] though spectral reflectance factors do not show a similar relationship. Differences could be due to poor sample uniformity or uniformity of the irradiating beam, which were not checked.

The measurements of the BCRA tiles are shown in Figures 20 through 29. The greatest discrepancies are observed at longer wavelengths where the PMT radiometer had very low

sensitivity. The standard deviations seldom rose above 0.005 from 380 to 700 nanometers, but were as high as 0.020 from 700 to 760 nanometers. The largest standard deviations are observed throughout the wavelength range where the PMT radiometer has very low sensitivity. CIELAB color differences between measured values and NIST's measurements were typically less than 1.0, and the orange BCRA tile measurements had the largest color difference of 1.77. The orange tile had the steepest spectral reflectance factor curve of all samples measured and therefore was the most difficult to measure accurately.

The low sensitivity of the silicon detector tele-radiometer would prevent its use for non-white samples, however, it could be used for near-white and white samples if the non-linearities of the measuring system were characterized. The PMT tele-radiometer had sufficient sensitivity to measure colored samples and white samples from 380 to 700 nanometers. Beyond 700 nanometers the meter sensitivity was so low (as indicated by noisy data and large standard deviations) that measurements may be inaccurate.

CONCLUSION

The goniospectrophotometer is a versatile instrument that will surely be used for many applications in the pursuit of appearance research. This thesis work has improved the instrument, but modifications will continually be required as new technologies become available and new applications are pursued.

Since the time that data were collected for this thesis, the RIT Munsell Color Science Laboratory has replaced the goniospectrophotometer's detector with a Photoresearch diode array radiometer. The new detector provides monochromatic collection, allowing samples to be irradiated with white light, permitting the collection of spectral total radiance factors* of fluorescent samples. It is recommended for future research to utilize monochromatic illumination and collection to permit the measurement of spectral reflected** and spectral fluorescent*** radiance factors.

* "The total radiance factor is the sum of the reflected and fluorescent radiance factor."[22]

** "The reflected radiance factor is the ratio of the radiance due to reflection of the medium to that of a perfect reflecting diffuser identically irradiated."[22]

*** "The fluorescent radiance factor is the ratio of the radiance due to fluorescence of the medium to that of a perfect reflected diffuser identically irradiated."[22]

The data collection and storage capacity of today's computers have made the effective use of a goniospectrophotometer possible. Years ago, a goniophotometer was considered by researchers to collect massive amounts of data to characterize a sample. Now, spectral evaluations can produce twenty to forty times more data (20 or 10 nanometer data) per measurement. What seems impractical today, may not be so tomorrow.

REFERENCES

1. The Measurement of Appearance, R. Hunter and R. Harold Eds., John Wiley and Sons, New York, 1987, p v.

2. F. Nicodemus et al,"NBS Monograph 160: Geometrical Considerations and Nonmenclature for Reflectance", U.S. Government Printing Office, Washington, DC, 1977, p 8.

3. W. Erb, "Properties of Standard Reference Materials for Reflection", Color Research and Application, 4, 113-118 (1979).

4. F. Clarke, "Goniophotometry and the Use of Ceramic Colour Standards", Proceedings of the International Colour Association, Congress COLOUR 73, 346-350, 1973.

5. Colorimetry, CIE Publication 15, 1971, p 14.

6. M. Fairchild, "Goniospectrophotometric Characteristics of White Reflectance Standards with the CIE Recommended View Angle Limits for Normal/45 Reflectance Factor Measurements", research paper for RIT course PPHS-771, 1985.

7. F. Clarke et al, "Goniophotometric and Polarization Properties of White Reflection Standard Materials", Lighting Research and Technology, 15, 133-149 (1983).

8. F. Clarke et al, "Goniophotometric and Polarization Properties of the Common White Reflection Standards", National Physical Laboratory Report MOM 13, 1975.

9. V. Kartachevskaya et al, "International Comparison of Measurements of Luminance Factor and Reflectance of White Diffusing Samples", Applied Optics, 14, 2694-2702 (1975).

10. J. Hsia, "NBS 45/Normal Reflectometer for Absolute Reflectance Factors", Metrologia, 17, 97-102 (1981).

11. A. Robertson, "International Comparison of Working Standards for Colorimetry", J.O.S.A., 55, 694-706 (1965).

12. A. Robertson, "Effect of Polarization on Reflectance Factor Measurements", Applied Optics, 11, 1936-1941 (1972).

13. D. Carmer, "Some Polarization Characteristics of Magnesium Oxide and Other Diffuse Reflectors", Applied Optics, 8, 1597-1605 (1969).

14. H. Minato et al, "Errors in Spectrophotometry and Colorimetry of Fluorescent Samples Caused by Polarization of the Measuring System", Color Research and Applications, 8, 238-244 (1983).

15. R. Boyd, Radiometry and the Detection of Optical Radiation, John Wiley and Sons, New York, 1983, p 16.

16. C. de Boor, A Practical Guide to Splines, Springer-Verlag, New York, 1978, pp. 299-320.

17. A. Savitzky and M. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures", Analytical Chemistry, 36, 1627 (1964).

18. G. Wyszecki and W. Stiles, Color Science: Concepts and Methods, Quantitative Data and Formulae, John Wiley and Sons, New York, 1982, pp 145-146.

19. Optical Radiation Measurements Volume 2, F. Grum and C. Bartleson Eds., Academic Press, New York, 1980, p 72.

20. ibid., pp 80-92.

21. ibid., pp 125-130.

22. ibid., p 245.

23. A. Robertson, "Computation of Correlated Color Temperature and Distribution Temperature", J.O.S.A., 58, 1528-1535 (1968).

24. ASTM Standards on Color and Appearance Measurement, American Society for Testing and Materials, Philadelphia, PA, 1987, pp 166-213.

25. H.S. Fairmain, "The Calculation of Weight Factors for Tristimulus Integration", Color Research and Application, 10, 199-203 (1985).

26. Method of Measuring and Specifying Colour Rendering Properties of Light Sources, CIE Publication 13.2, 1974, pp 9-14.

27. F. Grum and M. Saltzman,"New White Standard of Reflectance", CIE Publication 36, Proceedings 18th Session CIE, London, England, 1975, 1976, pp. 91-97.

APPENDIX

This section contains compiler listing of software written to collect and reduce data from the goniospectrophotometer. The source code has been split into the following sections for easier access:

(A) GONIO: A goniospectrophotometer data collection program for the DEC RT-11 operating system.

(B) PLOT: A plotting program for the DEC RT-11 and S & H Computing TSX+ operating systems.

(C) CONVERT: A data file conversion program for the DEC RT-11 and S & H Computing TSX+ operating systems.

(D) COLR: A spectra handling and color calculation program for the DEC RT-11 and S & H Computing TSX+ operating systems.

(E) COLOR: A spectra handling and color calculation program for the DEC VAX/VMS operating system.

APPENDIX A

GONIO PROGRAM

The gonio program permits a user to collect, display, and save spectral reflectance factor data from the Munsell Color Science Laboratory Goniospectrophotometer. The program consists of the following modules:

A1. GONIO.PAS: a goniospectrophotometer data collection program.

A2. PTWAIT.MAC: a procedure to suspend program execution for a specified number of 1/60th second increments.

```
1       PROGRAM spectro_gonio_photometer;
2
3        { This code is for a main program
4          Main Program File Name :  GONIO.PAS
5          Other subprograms needed :  PTWAIT.MAC
6          Compiling Command : pas gonio/fis
7          Linking Command :  link gonio,obj:ptwait,sy:pas2.fis
8          Hardware Required :  DRV11J at default address
9
10      Software Modifications/ Date/ Programmer
11         Program Written/ 05-23-84/ R.M. Miller
12         Wavelength drive control added/ 05-29-84/ R.M. Miller
13         Change parameter option added/ 05-30-84/ R.M. Miller
14         Collection of angular data at one wavelength (or white light illumination)
15           added/ 06-14-84/ R.M. Miller
16         Reading & writing of data files (angular and spectral) added/ 06-20-84/
17           R.M. Miller
18         Collection of spectral data at a particular angle added/ 06-20-84/ R.M.
19           Miller
20         Default parameters procedure changed to read values from a disk file/
21           06-21-84
22         Stepping motor procedures changed to work properly with the new stepping
23           motor translator/ 10-27-84/ R.M. Miller
24         Stepping motor procedures changed to ramp the stepping speed from off to
25           full speed, and from full speed to off/ 10-28-84/ R.M. Miller
26         Enhanced testing of stepping motors by adding absolute stepping in addition
27           to relative stepping/ 11-08-84/ R.M. Miller
28         Function  GETVAL changed to wait for the meter display to settle before
29           reporting a value/ 11-09-84/ R.M. Miller
30         Procedures changed to irradiate the sample with 550 nm light
31           to permit mounting and adjusting the sample/ 11-10-84/ R.M. Miller
32         Gonio data collection procedure changed to write reflectance factors in
33           decimal form instead of scientific notation unless values are very
34           large or small/ 11-21-84/ R.M. Miller
35         Spectral collection routine enhanced to enable reference data to be
36           collected at wavelength increments other than 5 nm/ 11-27-84/ R.M. Miller
37         Option added to display a period for each attempt to read the photometer
38           display/ 01-06-85/ R.M. Miller
39         Option added to prevent wavelength drive from rewinding wavelength until
40           radiometer lens is capped, default device changed to GON:, code added to
41           prevent a user from trying to perform a measurement without first
42           calibrating, option added for changing starting and ending wavelengths,
43           wavelengths and photometric value now displayed on terminal as data is
44           collected/ 04-21-85/ R.M. Miller
45         Option added for automatic flipping of AMBER and CYAN filters on mono/
46           5-86/ R.M. Miller
47
48                              BRIEF PROGRAM DESCRIPTION
49
50         This program runs the goniospectrophotometer wavelength drive,
51         angular drive, and reads the photometer port for each photometric
52         value needed.
53
```

```
 54          end of informational heading }
 55
 56        CONST
 57          spectralangles = false;
 58          max_n = 100; { maximum number of reads displayed when in DEBUG mode }
 59          csroutval = 400B;
 60          defwldelay = 350;
 61          defwlinc = 5;
 62          defarmdelay = 200;
 63          defdelayafterstepping = 5000;
 64          def_between_reads_delay = 100;
 65          defnumberofreads = 10;
 66          defdelaynumberofsteps = 5;
 67          defdelaymult = 3;
 68          defradiometertolerance = 0.001;
 69          version = 4.0;
 70          def_wl_flip = 600;
 71          def_flip_fraction = 1.1;
 72          maxangles = 15;
 73          defstartingwavelength = 380;
 74          defendingwavelength = 780;
 75          defstarting_scale = 0;
 76          defrewind = true;
 77          mininc = 5;
 78          numberofwavelengths = 81;
 79          noofwavelengths = 77;
 80          stringmax = 40;
 81          defdev = 'GON';
 82          defext = 'GON';
 83          defwhiteref = 'DK0:HALONA.REF';
 84          top = 10;
 85
 86        TYPE
 87          longint = 0..65535;
 88          adata =
 89            RECORD
 90              val: ARRAY [1..15] OF real;
 91              descrip: PACKED ARRAY [1..stringmax] OF char;
 92            END;
 93          string14 = PACKED ARRAY [1..14] OF char;
 94          string = PACKED ARRAY [1..stringmax] OF char;
 95          string6 = PACKED ARRAY [1..6] OF char;
 96          string3 = PACKED ARRAY [1..3] OF char;
 97          string2 = PACKED ARRAY [1..2] OF char;
 98          word = 0..65535;
 99          data =
100            RECORD
101              descrip: string;
102              inc: integer;
103              val: ARRAY [1..numberofwavelengths] OF real;
104            END;
105
```

```
106         VAR
107           currentwave, value, anglefactor, flip_fraction, radiometertolerance: real;
108           debug, showreads, wavecalibrated, anglecalibrated, r_calibrated, rewind, debug_wait: boolean;
109           csra ORIGIN 164160B, csrb ORIGIN 164164B, csrc ORIGIN 164170B, csrd ORIGIN 164174B,
110            dbra ORIGIN 164162B, dbrb ORIGIN 164166B, dbrc ORIGIN 164172B, dbrd ORIGIN 164176B: integer;
111           q: char;
112           starting_scale, scale, i, j, k, wldelay, armdelay, numberofreads, wl_flip, startingwavelength,
113            endingwavelength, delayafterstepping, between_reads_delay, inc, wlinc, delaynumberofsteps,
114            delaymult, len: integer;
115           asample: adata;
116           whiteref, fn: string14;
117           sample, reference: data;
118           f: FILE OF data;
119           out: text;
120           stepdelaymult: ARRAY [ - 5..5] OF real;
121
122         LABEL
123           1;
124
125
126         PROCEDURE twait(n: word);
127           EXTERNAL;
128
129
130         PROCEDURE writedb(s: PACKED ARRAY [lower..upper: integer] OF char);
131
132           VAR
133             i: integer;
134             q: char;
135
136
137           BEGIN
138             IF debug THEN
139               BEGIN
140               FOR i := lower TO upper DO write(s[i]);
141               writeln;
142               IF debug_wait THEN
143                 BEGIN
144                 write('select  1: disable debug,  2: disable waits, or RETURN to continue');
145                 q := ' ';
146                 IF NOT eoln THEN readln(q)
147                 ELSE readln;
148                 CASE q OF
149                   '1': debug := false;
150                   '2': debug_wait := false;
151                   OTHERWISE;
152                   END;
153                 END;
154               END;
155           END;
156
157
```

```
158         PROCEDURE wait(n: integer);
159
160           VAR
161             j: integer;
162             q: char;
163
164
165           BEGIN
166             FOR j := 1 TO n DO q := 'X';
167           END;
168
169
170         PROCEDURE readch(VAR q: char);
171
172
173           BEGIN
174             q := ' ';
175             IF NOT eoln THEN readln(q)
176             ELSE readln;
177             IF q = '?' THEN GOTO 1;
178             IF q IN ['a'..'z'] THEN q := chr(ord(q) - 32);
179           END;
180
181
182         PROCEDURE return(x2: integer);
183
184           VAR
185             i3: integer;
186             qr: char;
187
188
189           BEGIN
190             FOR i3 := 0 TO x2 DO
191               BEGIN
192               writeln;
193               END;
194             write(' ': 25, 'press RETURN to continue');
195             readch(qr);
196           END;
197
198
199         PROCEDURE jump(x2: integer);
200
201           VAR
202             i3: integer;
203
204
205           BEGIN
206             FOR i3 := 1 TO x2 DO
207               BEGIN
208               writeln;
209               END;
```

```
210           END;
211
212
213         PROCEDURE prompt(x1: integer);
214
215           VAR
216             x2: integer;
217
218
219           BEGIN
220             writeln(chr(27), 'H', chr(27),
221                     'JMunsell Color Science Laboratory Spectrogoniophotometer Version', version: 4: 1);
222             jump(x1);
223           END;
224
225
226         PROCEDURE instruct(direction: PACKED ARRAY [lower..upper: integer] OF char);
227
228           VAR
229             q: char;
230             i: integer;
231
232
233           BEGIN
234             prompt(10);
235             FOR i := lower TO upper DO write(direction[i]);
236             writeln;
237             writeln;
238             writeln;
239             write('     Press RETURN when this has been done     ');
240             readch(q);
241           END;
242
243
244         PROCEDURE concat(VAR filename: string14;
245                          dev: string3;
246                          filnam: string6;
247                          ext: string3);
248
249           VAR
250             i: integer;
251
252
253           BEGIN
254             FOR i := 1 TO 3 DO
255               BEGIN
256               filename[i] := dev[i];
257               filename[i + 11] := ext[i]
258               END;
259             FOR i := 1 TO 6 DO filename[i + 4] := filnam[i];
260             filename[4] := ':';
261             filename[11] := '.';
```

```
262           END;
263
264
265         PROCEDURE getfilename(VAR temp: string14);
266
267           VAR
268             nc, np, i, j, colon, period, endpos: integer;
269             dev, ext: string3;
270             fn: string6;
271             error: boolean;
272             ch: char;
273
274
275           BEGIN
276             REPEAT
277               BEGIN
278               FOR i := 1 TO 14 DO temp[i] := chr(0);
279               readln(temp);
280               IF temp[1] = '?' THEN GOTO 1;
281               FOR i := 1 TO 14 DO IF temp[i] = ' ' THEN temp[i] := chr(0);
282               FOR i := 1 TO 14 DO
283                 BEGIN
284                 IF temp[i] = chr(0) THEN
285                   BEGIN
286                   FOR j := i TO 13 DO temp[j] := temp[j + 1];
287                   temp[14] := chr(0);
288                   END;
289                 END;
290               nc := 0;
291               np := 0;
292               endpos := 15;
293               colon := 0;
294               FOR i := 1 TO 14 DO
295                 BEGIN
296                 IF temp[i] = ':' THEN
297                   BEGIN
298                   nc := nc + 1;
299                   colon := i;
300                   END;
301                 IF temp[i] = '.' THEN
302                   BEGIN
303                   np := np + 1;
304                   period := i;
305                   END;
306                 IF temp[15 - i] = chr(0) THEN endpos := 15 - i;
307                 END;
308               IF np = 0 THEN period := endpos;
309               error := false;
310               FOR i := 1 TO (endpos - 1) DO
311                 BEGIN
312                 ch := temp[i];
313                 IF NOT (ch IN ['a'..'z', 'A'..'Z', '0'..'9', ':', '.']) THEN error := true;
```

```
314               END;
315             IF (period < colon) OR (colon = 1) OR (period = 1) OR (colon > 4) OR
316                ((period - colon) > 7) OR ((endpos - period) > 4) OR (period = colon + 1) THEN
317               error := true;
318             IF error THEN
319               BEGIN
320               prompt(1);
321               writeln('          Filename Entered by User :  ', temp);
322               writeln('WARNING!!! This file name is not in the proper format');
323               writeln('          The filename should be six characters long');
324               writeln('          with no intermediate spaces. The only legal');
325               writeln('          characters are letters and numbers. If you');
326               writeln('          wish a file can also be specified as :');
327               writeln('                                DEV:FILNAM.EXT');
328               writeln('Enter ? to RETURN to menu');
329               write('  -or-  enter filename again (?,filename) :  ');
330               END
331             ELSE
332               BEGIN
333               FOR i := 1 TO 3 DO
334                 BEGIN
335                 ext[i] := chr(0);
336                 END;
337               dev := ext;
338               FOR i := 1 TO 6 DO fn[i] := chr(0);
339               IF colon > 1 THEN FOR i := 1 TO (colon - 1) DO dev[i] := temp[i]
340               ELSE dev := defdev;
341               FOR i := (colon + 1) TO (period - 1) DO fn[i - colon] := temp[i];
342               IF endpos > period THEN
343                 FOR i := (period + 1) TO (endpos - 1) DO ext[i - period] := temp[i]
344               ELSE ext := defext;
345               END;
346             END;
347           UNTIL NOT error;
348           concat(temp, dev, fn, ext);
349         END;
350
351
352       PROCEDURE retrievefile(VAR name: string14;
353                              VAR s1: data);
354
355         VAR
356           i: integer;
357
358
359         BEGIN
360           REPEAT
361             BEGIN
362             reset(f, name, , len);
363             IF len = - 1 THEN
364               BEGIN
365               prompt(1);
```

```
366                writeln(' ': 10, name, ' is not a valid file name');
367                jump(1);
368                write(' ': 10, 'enter ? to return to menu -or-');
369                writeln;
370                write(' ': 15, 'enter valid file name (xxxxxx) :  ');
371                getfilename(name);
372                END;
373              END;
374            UNTIL len <> - 1;
375            get(f);
376            s1 := f^;
377            FOR i := (noofwavelengths + 1) TO numberofwavelengths DO
378              s1.val[i] := s1.val[noofwavelengths];
379            close(f);
380          END;
381
382
383        PROCEDURE getfile(VAR s: data);
384
385          VAR
386            i: integer;
387
388
389          BEGIN
390            prompt(top);
391            write('enter file name to retrieve (XXXXXX) :  ');
392            getfilename(fn);
393            retrievefile(fn, s);
394          END;
395
396
397        PROCEDURE savefile(enter: data);
398
399          VAR
400            i, len: integer;
401            f: FILE OF data;
402            fn: string14;
403            q: char;
404
405
406          BEGIN
407            prompt(top);
408            write('enter filename for data (xxxxxx) :  ');
409            getfilename(fn);
410            REPEAT
411              BEGIN
412              reset(f, fn, , len);
413              IF len <> - 1 THEN
414                BEGIN
415                close(f);
416                prompt(1);
417                writeln('WARNING!!!! file name : ', fn, '  already exists');
```

```
418                jump(2);
419                write(' O(elete this file and use same name -OR- E(nter new name [E]:  ');
420                readch(q);
421                IF q <> 'O' THEN
422                  BEGIN
423                  jump(1);
424                  write('enter new file name (xxxxxx) :  ');
425                  getfilename(fn);
426                  END;
427                END;
428              END;
429            UNTIL (q = 'O') OR (len = - 1);
430            rewrite(f, fn);
431            f^ := enter;
432            put(f);
433            close(f);
434          END;
435
436
437        PROCEDURE bell(i: integer);
438
439          VAR
440            q: char;
441            j: integer;
442
443
444          BEGIN
445            q := chr(7);
446            IF i = 1 THEN write(q, q, q, q, q)
447            ELSE
448              BEGIN
449              FOR j := 1 TO 6 DO
450                BEGIN
451                write(q, q, q, q, q);
452                wait(delayafterstepping);
453                END;
454              END;
455          END;
456
457
458        PROCEDURE getscale(sc: integer);
459
460          VAR
461            i: integer;
462            q: char;
463
464
465          BEGIN
466            writeln;
467            writeln;
468            bell(1);
469            q := 'C';
```

```
470             IF (sc >= 0) AND (sc <= 4) THEN
471               BEGIN
472               writeln('set the radiometer multiplier to   X 10 ^ ', sc: 1);
473               writeln;
474               write('press RETURN when scale is set to above value -or- C(hange to another value :  ');
475               readch(q);
476               END;
477             IF q = 'C' THEN
478               BEGIN
479               prompt(top);
480               writeln;
481               writeln('Set the radiometer multiplier to one of the following appropriate values');
482               writeln;
483               FOR i := 0 TO 4 DO writeln(i: 1, ':    Scale  X  10 ^ ', i: 1);
484               writeln;
485               writeln;
486               REPEAT
487                 BEGIN
488                 writeln(chr(27), 'A                                                    ');
489                 write(chr(27), 'Aand enter the option number :  ');
490                 readch(q);
491                 END;
492               UNTIL (q IN ['0'..'4']);
493               scale := ord(q) - 48;
494               END
495             ELSE scale := sc;
496             prompt(1);
497             writeln('Collecting data !!!');
498           END;
499
500
501         FUNCTION getval: real;
502
503           VAR
504             bit, val: longint;
505             i, j, k, attempt, defdbrd: integer;
506             indvalue, oldvalue, value, mult: real;
507             temp: ARRAY [1..max_n] OF real;
508             error: boolean;
509
510
511           BEGIN
512             defdbrd := 31 OR (40B AND dbrd);
513             dbrd := defdbrd;
514             twait(10 * between_reads_delay);
515             REPEAT
516               dbrd := defdbrd;
517               value := - 1.0E10;
518               error := false;
519               attempt := 0;
520               REPEAT
521                 dbrd := defdbrd;
```

```
522                 twait(3 * between_reads_delay);
523                 oldvalue := value;
524                 attempt := attempt + 1;
525                 IF showreads THEN write('.');
526                 value := 0.0;
527                 k := 0;
528                 REPEAT
529                   dbrd := defdbrd;
530                   indvalue := 0;
531                   WHILE csrc > 0 DO write('!');
532                   dbrd := dbrd AND 177757B;
533                   wait(40);
534                   val := dbrc;
535                   IF csrc > 0 THEN writeln('******** ERROR... Conversion in progress');
536                   dbrd := defdbrd;
537                   twait(between_reads_delay);
538                   dbrd := defdbrd;
539                   IF (val AND 40000B) > 0 THEN
540                     BEGIN
541                     prompt(top);
542                     writeln('OVERLOAD !!!!!', chr(7), chr(7), chr(7));
543                     writeln;
544                     IF scale < 4 THEN
545                       BEGIN
546                       getscale(scale + 1);
547                       error := true;
548                       END
549                     ELSE
550                       BEGIN
551                       FOR i := 1 TO 10 DO write(chr(7));
552                       writeln('UNRECOVERABLE ERROR!!! Too Much Energy... Press RETURN');
553                       GOTO 1;
554                       END;
555                     END;
556                   IF NOT error THEN
557                     BEGIN
558                     mult := 0.01;
559                     bit := 1;
560                     FOR i := 1 TO 3 DO
561                       BEGIN
562                       FOR j := 1 TO 4 DO
563                         BEGIN
564                         IF (val AND bit) > 0 THEN indvalue := indvalue + mult;
565                         mult := mult * 2;
566                         bit := bit * 2;
567                         END;
568                       mult := mult / 1.6;
569                       END;
570                     IF (val AND 10000B) > 0 THEN indvalue := indvalue + 10.00;
571                     IF (val AND 20000B) = 0 THEN indvalue := - 1 * indvalue;
572                     k := k + 1;
573                     temp[k] := indvalue;
```

```
574                     value := value + indvalue;
575                     END;
576                 UNTIL (k = numberofreads) OR error;
577                 IF NOT error THEN
578                   BEGIN
579                   value := value / numberofreads;
580                   IF debug THEN
581                     BEGIN
582                     writeln('number of points = ', k);
583                     writeln('value = ', value: 7: 4);
584                     indvalue := 0;
585                     FOR i := 1 TO k DO
586                       writeln(i, ' ', temp[i]: 7: 4, '  ', ((temp[i] - value) / value * 100): 7: 3);
587                     END;
588                   indvalue := 0;
589                   j := 0;
590                   FOR i := 1 TO k DO
591                     BEGIN
592                     IF value <> 0 THEN
593                       BEGIN
594                       IF abs((value - temp[i]) / value) < 0.25 THEN
595                         BEGIN
596                         indvalue := indvalue + temp[i];
597                         j := j + 1;
598                         END
599                       ELSE IF showreads THEN write(' * ');
600                       END
601                     ELSE
602                       BEGIN
603                       indvalue := indvalue + temp[i];
604                       j := j + 1;
605                       END;
606                     END;
607                   IF j <> 0 THEN indvalue := indvalue / j
608                   ELSE
609                     BEGIN
610                     error := true;
611                     writeln('All values out of tolerance. Data set will be re-read');
612                     indvalue := value;
613                     END;
614                   IF debug THEN
615                     BEGIN
616                     IF j <> numberofreads THEN writeln('BAD DATA POINTS!!! ', value, ' ==> ', indvalue);
617                     END;
618                   IF j < numberofreads THEN value := indvalue;
619                   IF (value < 1.0) AND (scale > 0) THEN getscale(scale - 1);
620                   IF (attempt > 100) THEN
621                     BEGIN
622                     writeln;
623                     writeln(chr(7),
624                          'WARNING !!! 100 attempts to read stable data from the radiometer have been made'
625                             );
```

```
626                     write('C(ontinue reading, U(se last values, -or- ? to abort (C,U,?) [U] :  ');
627                     readch(q);
628                     IF q = 'C' THEN attempt := 2
629                     ELSE
630                       BEGIN
631                       value := (value + oldvalue) / 2.0;
632                       oldvalue := value;
633                       ENO;
634                     END;
635                 ENO;
636               UNTIL error OR (abs(oldvalue - value) <= abs(radiometertolerance * value));
637             UNTIL NOT ((value < 1.0) ANO (scale > 0)) ANO NOT error;
638             FOR i := 1 TO scale OO value := value * 10.0;
639             getval := value;
640           ENO;
641
642
643         PROCEOURE displayval;
644
645           VAR
646             i, j, k, temp: integer;
647             q: char;
648
649
650           BEGIN
651             showreads := false;
652             temp := numberofreads;
653             numberofreads := 1;
654             write('enter number of reads to be displayed :  ');
655             readln(k);
656             j := 0;
657             REPEAT
658               BEGIN
659               prompt(10);
660               write('  Photometer Value :  ', ' ': 5);
661               REPEAT
662                 BEGIN
663                 FOR i := 1 TO 5 OO write(chr(8));
664                 write(getval: 5: 2);
665                 j := j + 1;
666                 ENO;
667               UNTIL j = k;
668               writeln;
669               writeln;
670               write('Q(uit  -or- RETURN to continue :  ');
671               readch(q);
672               ENO;
673             UNTIL q = 'Q';
674             showreads := true;
675             numberofreads := temp;
676           ENO;
677
```

```
678
679         PROCEDURE step(motor, number: integer);
680
681           VAR
682             i, defaultvalue, delay, stepvalue, stopvalue, currentdelay: integer;
683
684
685           FUNCTION ramp: integer;
686
687
688             BEGIN
689               IF (i > delaynumberofsteps) AND (i < (stopvalue - delaynumberofsteps)) THEN ramp := delay
690               ELSE IF (i < (stopvalue - delaynumberofsteps)) THEN
691                 ramp := round(delay * (delaymult - (delaymult - 1) * i / delaynumberofsteps))
692               ELSE
693                 ramp := round(delay * (1.D + (delaymult - 1) * (i - stopvalue + delaynumberofsteps) /
694                               delaynumberofsteps));
695             END;
696
697
698           BEGIN
699             defaultvalue := 31 OR (dbrd AND 40B);
700             IF (motor < 1) OR (motor > 2) THEN writeln('Illegal Motor #')
701             ELSE
702               BEGIN
703               IF motor = 2 THEN
704                 BEGIN
705                 stepvalue := 4;
706                 delay := armdelay;
707                 END
708               ELSE
709                 BEGIN
710                 stepvalue := 1;
711                 delay := wldelay;
712                 END;
713               IF number < 0 THEN stepvalue := stepvalue * 2;
714               stepvalue := defaultvalue - stepvalue;
715               stopvalue := abs(number);
716               FOR i := 1 TO stopvalue DO
717                 BEGIN
718                 dbrd := stepvalue;
719                 wait(delay);
720                 dbrd := defaultvalue;
721                 wait(ramp);
722                 END;
723               END;
724           END;
725
726
727         PROCEDURE wlstep(number: integer);
728
729           VAR
```

```
730             j, k: integer;
731
732
733           BEGIN
734             step(1, number);
735             wait(delayafterstepping);
736           END;
737
738
739         PROCEDURE armstep(number: integer);
740
741           VAR
742             j, k: integer;
743
744
745           BEGIN
746             step(2, - 1 * number);
747             wait(delayafterstepping);
748           END;
749
750
751         PROCEDURE movewave(wlnew: real);
752
753           VAR
754             q: char;
755
756
757           PROCEDURE setwl;
758
759             VAR
760               offset: integer;
761
762
763             BEGIN
764               offset := round((wlnew - currentwave) * 4.0);
765               currentwave := wlnew;
766               IF offset < 0 THEN
767                 BEGIN
768                 wlstep(offset - 40);
769                 wait(delayafterstepping * 2);
770                 wlstep(40);
771                 END
772               ELSE IF offset > 0 THEN wlstep(offset);
773             END;
774
775
776           BEGIN
777             IF wavecalibrated THEN setwl
778             ELSE
779               BEGIN
780               REPEAT
781                 prompt(10);
```

```
782                write('Is the wavelength set to ', wlnew: 5: 1, ' nanometers ? (Y,N) [N] :  ');
783                readch(q);
784                IF q <> 'Y' THEN
785                  BEGIN
786                  writeln;
787                  write('Enter current wavelength :  ');
788                  readln(currentwave);
789                  setwl;
790                  END
791                ELSE currentwave := wlnew;
792              UNTIL q = 'Y';
793              wavecalibrated := true;
794              END;
795          END;
796
797
798        PROCEDURE setarm(value: real);
799
800          VAR
801            currentvalue: real;
802            offset: integer;
803            q: char;
804
805
806          BEGIN
807            writedb('in procedure setarm');
808            REPEAT
809              prompt(10);
810              writeln('NOTE: Angles are measured to the irradiating beam');
811              writeln;
812              write('Is the gonio arm set to ', value: 5: 1, ' degrees  ? (Y,N) [N] :  ');
813              readch(q);
814              IF q <> 'Y' THEN
815                BEGIN
816                writeln;
817                write('Enter current position :  ');
818                readln(currentvalue);
819                offset := round((value - currentvalue) * 10.0);
820                IF offset < 0 THEN
821                  BEGIN
822                  armstep(offset - 100);
823                  wait(delayafterstepping * 2);
824                  armstep(100);
825                  END
826                ELSE IF offset > 0 THEN armstep(offset);
827                END;
828            UNTIL q = 'Y';
829          END;
830
831
832        PROCEDURE wlstepr;
833
```

```
834         VAR
835           stepdelay, value: integer;
836           desiredwl: real;
837           q: char;
838
839
840         BEGIN
841           writedb('in procedure wlstepr');
842           wavecalibrated := false;
843           stepdelay := wldelay;
844           prompt(10);
845           writeln('Current Delay Value :  ', stepdelay);
846           writeln;
847           write('Use current delay value (Y,N) [Y] :  ');
848           readch(q);
849           IF q = 'N' THEN
850             BEGIN
851             REPEAT
852               write('enter delay value (1-10000) :  ');
853               readln(wldelay);
854             UNTIL wldelay > 0;
855             END;
856           writeln;
857           write('M(anually test -or- A(utomatically test control ? (M,A) [A] :  ');
858           readch(q);
859           IF q = 'M' THEN
860             BEGIN
861             REPEAT
862               write('enter number of steps (positive for forward) :  ');
863               readln(value);
864               wlstep(value);
865             UNTIL value = 0;
866             END
867           ELSE
868             BEGIN
869             wavecalibrated := false;
870             REPEAT
871               write('enter desired wavelength in nanometers :  ');
872               readln(desiredwl);
873               movewave(desiredwl);
874               write('T(ry again -or- Q(uit (T,Q) [T] :  ');
875               readch(q);
876             UNTIL q = 'Q';
877             END;
878           wldelay := stepdelay;
879         END;
880
881
882       PROCEDURE testarm;
883
884         VAR
885           stepdelay, value: integer;
```

```
886             desiredangle: real;
887             q: char;
888
889
890           BEGIN
891             writedb('in procedure testarm');
892             stepdelay := armdelay;
893             prompt(10);
894             writeln('Current Oelay Value :  ', armdelay);
895             writeln;
896             write('Use current delay value (Y,N) [Y] :  ');
897             readch(q);
898             IF q = 'N' THEN
899               BEGIN
900               REPEAT
901                 write('enter delay value (1-10000) :  ');
902                 readln(armdelay);
903               UNTIL armdelay > 0;
904               ENO;
905             writeln;
906             write('M(anually test -or- A(utomatically test control ? (M,A) [A] :  ');
907             readch(q);
908             IF q = 'M' THEN
909               BEGIN
910               REPEAT
911                 write('enter number of steps (positive for forward) :  ');
912                 readln(value);
913                 armstep(value);
914               UNTIL value = 0;
915               END
916             ELSE
917               BEGIN
918               REPEAT
919                 write('enter desired angle in degrees :  ');
920                 readln(desiredangle);
921                 setarm(desiredangle);
922                 write('T(ry again -or- Q(uit (T,Q) [T] :  ');
923                 readch(q);
924               UNTIL q = 'Q';
925               END;
926             armdelay := stepdelay;
927           END;
928
929
930         PROCEDURE saveangular(VAR s: adata);
931
932           VAR
933             i, len: integer;
934             q: char;
935             f: text;
936
937           BEGIN
```

```
938           writedb('in procedure saveangular');
939           prompt(10);
940           write('enter filename for data (xxxxxx) :  ');
941           readln(fn);
942           REPEAT
943             BEGIN
944             reset(f, fn, , len);
945             IF len <> - 1 THEN
946               BEGIN
947               close(f);
948               prompt(10);
949               writeln('WARNING!!!! file name : ', fn, '  already exists');
950               writeln;
951               writeln;
952               write(' D(elete this file and use same name -OR- E(nter new name [E]:  ');
953               readch(q);
954               IF q <> 'D' THEN
955                 BEGIN
956                 jump(1);
957                 write('enter new file name (xxxxxx) :  ');
958                 readln(fn);
959                 END;
960               END;
961             END;
962           UNTIL (q = 'D') OR (len = - 1);
963           rewrite(f, fn);
964           FOR i := 1 TO maxangles DO
965             BEGIN
966             write(f, (10 + 5 * i): 2, ' , ');
967             IF (s.val[i] > 0.000001) AND (s.val[i] < 100.0) THEN writeln(f, s.val[i]: 10: 6)
968             ELSE writeln(f, s.val[i]);
969             END;
970           writeln(f, s.descrip);
971           close(f);
972         END;
973
974
975       PROCEDURE collectangular(VAR s: adata);
976
977         VAR
978           i: integer;
979           wave: real;
980           q: char;
981
982         BEGIN
983           writedb('in procedure collectangular');
984           prompt(10);
985           write('enter wavelength you wish to measure at :  ');
986           readln(wave);
987           movewave(550.0);
988           instruct('Please install and align the sample to be measured');
989           movewave(wave);
```

```
            setarm(15.0);
            write('press RETURN to start the scan');
            readch(q);
            FOR i := 1 TO 14 DO
              BEGIN
              s.val[i] := getval;
              armstep(5 * 10);
              END;
            s.val[15] := getval;
            prompt(10);
            write('enter a ', stringmax: 2, ' character description :  ');
            readln(s.descrip);
            armstep( - 10 * (85 - 15 + 10));
            wait(delayafterstepping * 2);
            armstep(10 * 10);
          END;


        PROCEDURE readbl(VAR out: text;
                         VAR b: boolean);

          VAR
            q: char;


          BEGIN
            readln(out, q);
            b := (q = 'T') OR (q = 't');
          END;


        PROCEDURE setdefaults;

          VAR
            len: integer;


          BEGIN
            concat(fn, defdev, 'PARAMS', 'DEF');
            reset(out, fn, , len);
            IF len <> - 1 THEN
              BEGIN
              readbl(out, debug);
              writedb('The DEBUG mode has been selected');
              readln(out, startingwavelength);
              readln(out, endingwavelength);
              readln(out, wlinc);
              readbl(out, rewind);
              readln(out, starting_scale);
              readln(out, wldelay);
              readln(out, armdelay);
              readln(out, numberofreads);
```

```
1042              readln(out, delayafterstepping);
1043              readln(out, delaynumberofsteps);
1044              readln(out, delaymult);
1045              readln(out, radiometertolerance);
1046              readln(out, between_reads_delay);
1047              readln(out, wl_flip);
1048              readln(out, flip_fraction);
1049              close(out);
1050              END
1051            ELSE
1052              BEGIN
1053              starting_scale := defstarting_scale;
1054              startingwavelength := defstartingwavelength;
1055              endingwavelength := defendingwavelength;
1056              rewind := defrewind;
1057              wldelay := defwldelay;
1058              armdelay := defarmdelay;
1059              numberofreads := defnumberofreads;
1060              delayafterstepping := defdelayafterstepping;
1061              delaynumberofsteps := defdelaynumberofsteps;
1062              delaymult := defdelaymult;
1063              radiometertolerance := defradiometertolerance;
1064              wlinc := defwlinc;
1065              between_reads_delay := def_between_reads_delay;
1066              wl_flip := def_wl_flip;
1067              flip_fraction := def_flip_fraction;
1068              END;
1069            writedb('past reading from PARAMS.DEF');
1070            concat(fn, defdev, 'STDREF', 'DEF');
1071            reset(out, fn, , len);
1072            IF len <> - 1 THEN
1073              BEGIN
1074              readln(out, whiteref);
1075              close(out);
1076              END
1077            ELSE whiteref := defwhiteref;
1078            writedb('past reading STDREF.DEF');
1079          END;
1080
1081
1082        PROCEDURE chgpar;
1083
1084          VAR
1085            q: char;
1086            names: ARRAY [1..5] OF string14;
1087            i, j: integer;
1088
1089
1090          BEGIN
1091            writedb('in procedure chgpar');
1092            REPEAT
1093              BEGIN
```

```
1094              prompt(0);
1095              writeln('Program Parameters     ': 35, 'Current Value': 20, 'Default Value': 20);
1096              writeln;
1097              writeln('W:', 'Wavelength Delay': 33, wldelay: 20, defwldelay: 20);
1098              writeln('A:', 'Gonio Arm Delay': 33, armdelay: 20, defarmdelay: 20);
1099              writeln('N:', 'Number of Reads Averaged': 33, numberofreads: 20, defnumberofreads: 20);
1100              writeln('S:', 'Delay after steps': 33, delayafterstepping: 20, defdelayafterstepping: 20);
1101              writeln('R:', 'Ramped Number of Steps': 33, delaynumberofsteps: 20, defdelaynumberofsteps:
1102                      20);
1103              writeln('M:', 'Delay Ramp Multiplier': 33, delaymult: 20, defdelaymult: 20);
1104              writeln('T:', 'Radiometer Tolerance': 33, radiometertolerance: 20: 5,
1105                      defradiometertolerance: 20: 5);
1106              writeln('C:', 'White calibration reference': 33, whiteref: 20, defwhiteref: 20);
1107              writeln('I:', 'Wavelength Increment': 33, wlinc: 20, defwlinc: 20);
1108              writeln('1:', 'Starting wavelength': 33, startingwavelength: 20, defstartingwavelength: 20);
1109              writeln('2:', 'Ending wavelength': 33, endingwavelength: 20, defendingwavelength: 20);
1110              writeln('3:', 'Starting scale': 33, starting_scale: 20, defstarting_scale: 20);
1111              writeln('4:', 'Rewind wavelength at EOR': 33, rewind: 20, defrewind: 20);
1112              writeln('5:', 'Delay between radiometer reads': 33, between_reads_delay: 20,
1113                      def_between_reads_delay: 20);
1114              writeln('6:', 'Debug Mode Enabled': 33, debug: 20, 'FALSE': 20);
1115              writeln;
1116              writeln('D:  change all parameters to their default values');
1117              writeln;
1118              writeln;
1119              REPEAT
1120                BEGIN
1121                write(chr(27), 'A', chr(27), 'JSelect W,A,N,S,R,M,T,C,I,1..6,D or Q(uit :  ');
1122                readch(q);
1123                END;
1124              UNTIL (q IN ['W', 'A', 'N', 'S', 'R', 'M', 'Q', 'T', 'C', 'I', '1'..'6', 'D']);
1125              writeln;
1126              CASE q OF
1127                'W':
1128                  BEGIN
1129                  REPEAT
1130                    BEGIN
1131                    write('enter new wavelength delay value (1 - 30000) : ');
1132                    readln(wldelay);
1133                    END;
1134                  UNTIL ((wldelay > 0) AND (wldelay <= 30000));
1135                  END;
1136                'A':
1137                  BEGIN
1138                  REPEAT
1139                    BEGIN
1140                    write('enter new gonio arm delay value (1 - 30000) : ');
1141                    readln(armdelay);
1142                    END;
1143                  UNTIL ((armdelay > 0) AND (armdelay <= 30000));
1144                  END;
1145                'N':
```

```
1146                  BEGIN
1147                  REPEAT
1148                    BEGIN
1149                    write('enter new number of reads value (1 - 1000) : ');
1150                    readln(numberofreads);
1151                    END;
1152                  UNTIL ((numberofreads > 0) AND (numberofreads <= 1000));
1153                  END;
1154                'S':
1155                  BEGIN
1156                  REPEAT
1157                    BEGIN
1158                    write('enter new delay after stepping value (1 - 30000) : ');
1159                    readln(delayafterstepping);
1160                    END;
1161                  UNTIL ((delayafterstepping > 0) AND (delayafterstepping <= 30000));
1162                  END;
1163                'R':
1164                  BEGIN
1165                  REPEAT
1166                    BEGIN
1167                    write('enter new ramped number of steps (1 - 1000) : ');
1168                    readln(delaynumberofsteps);
1169                    END;
1170                  UNTIL ((delaynumberofsteps > 0) ANO (delaynumberofsteps <= 1000));
1171                  END;
1172                'M':
1173                  BEGIN
1174                  REPEAT
1175                    BEGIN
1176                    write('enter new delay ramp multiplier (1 - 1000) : ');
1177                    readln(delaymult);
1178                    END;
1179                  UNTIL ((delaymult > 0) AND (delaymult <= 1000));
1180                  END;
1181                'T':
1182                  BEGIN
1183                  REPEAT
1184                    BEGIN
1185                    write('enter new radiometer tolerance (0.000001 - 0.50) : ');
1186                    readln(radiometertolerance);
1187                    END;
1188                  UNTIL ((radiometertolerance >= 1.0E-7) AND (radiometertolerance <= 0.5));
1189                  END;
1190                'C':
1191                  BEGIN
1192                  writeln('White reference data currently available :');
1193                  concat(fn, defdev, 'STDREF', 'DEF');
1194                  reset(out, fn, , len);
1195                  i := 1;
1196                  IF len <> - 1 THEN
1197                    BEGIN
```

```
1198                    WHILE NOT eof(out) AND (i < 6) DO
1199                      BEGIN
1200                      readln(out, names[i]);
1201                      writeln(i: 10, ':  ', names[i]);
1202                      i := i + 1;
1203                      END;
1204                    close(out);
1205                    i := i - 1;
1206                    writeln;
1207                    writeln;
1208                    REPEAT
1209                      write(chr(27), 'A', chr(27), 'KSelect white reference for calibration :  ');
1210                      readln(j);
1211                    UNTIL (j > 0) AND0 (j <= i);
1212                    whiteref := names[j];
1213                    END
1214                  ELSE
1215                    BEGIN
1216                    writeln;
1217                    writeln('WARNING !!! Unable to locate file :  STDREF.DEF');
1218                    write('enter filename of white reference data :  ');
1219                    readln(whiteref);
1220                    END;
1221                  END;
1222                '1':
1223                  BEGIN
1224                  REPEAT
1225                    write('enter wavelength increment (5,10,15...,50) :  ');
1226                    readln(wlinc);
1227                  UNTIL (wlinc > 4) AND (wlinc < 51) AND ((wlinc MOD 5) = 0);
1228                  END;
1229                '1':
1230                  BEGIN
1231                  REPEAT
1232                    write('enter starting wavelength :  ');
1233                    readln(startingwavelength);
1234                  UNTIL (startingwavelength >= 380) AND (startingwavelength <= 780) AND
1235                       ((startingwavelength MOD 5) = 0);
1236                  END;
1237                '2':
1238                  BEGIN
1239                  REPEAT
1240                    write('enter ending wavelength :  ');
1241                    readln(endingwavelength);
1242                  UNTIL (endingwavelength >= 380) AND (endingwavelength <= 780) AND
1243                       ((endingwavelength MOD 5) = 0);
1244                  END;
1245                '3':
1246                  BEGIN
1247                  REPEAT
1248                    write('enter starting scale number (0-3) :  ');
1249                    readln(starting_scale);
```

```
1250                UNTIL (starting_scale >= 0) AND (starting_scale <= 3);
1251                END;
1252              '4':
1253                BEGIN
1254                REPEAT
1255                  write('automatically rewind wavelength at EOR ? (Y,N) :  ');
1256                  readch(q);
1257                UNTIL (q = 'Y') OR (q = 'N');
1258                rewind := q = 'Y';
1259                END;
1260              '5':
1261                BEGIN
1262                REPEAT
1263                  BEGIN
1264                  write('enter new delay between reads (1 - 30000) : ');
1265                  readln(between_reads_delay);
1266                  END;
1267                UNTIL ((between_reads_delay > 0) AND (between_reads_delay <= 30000));
1268                END;
1269              '6':
1270                BEGIN
1271                REPEAT
1272                  write('Enable the debug mode of operation ? (Y,N) :  ');
1273                  readch(q);
1274                UNTIL (q = 'Y') OR (q = 'N');
1275                debug := q = 'Y';
1276                END;
1277              'D':
1278                BEGIN
1279                setdefaults;
1280                q := 'Q';
1281                END;
1282              'Q': ;
1283              END;
1284            END;
1285          UNTIL q = 'Q';
1286        END;
1287
1288
1289      PROCEDURE getwlinc;
1290
1291        VAR
1292          value: PACKED ARRAY [1..2] OF char;
1293
1294
1295        BEGIN
1296          writedb('in procedure getwlinc');
1297          REPEAT
1298            prompt(10);
1299            writeln('valid wavelength increments :  5,10,15,20,25,30,35,40,45,50');
1300            write('enter increment [', wlinc: 2, '] :  ');
1301            readln(value);
```

```
1302              IF (value[1] IN ['0'..'9', ' ']) AND (value[2] IN ['0'..'9', ' ']) THEN
1303                BEGIN
1304                IF value = '  ' THEN inc := wlinc
1305                ELSE IF value[2] = ' ' THEN inc := ord(value[1]) - 48
1306                ELSE
1307                  BEGIN
1308                  inc := 10 * (ord(value[1]) - 48);
1309                  inc := inc + ord(value[2]) - 48;
1310                  END
1311                END
1312              ELSE inc := 0;
1313            UNTIL (inc >= 5) AND (inc <= 50) AND ((inc MOD 5) = 0);
1314          END;
1315
1316
1317        PROCEDURE collectdata;
1318
1319          VAR
1320            i, wave: integer;
1321            doreference, filterchanged: boolean;
1322            temp, temp1, angle: real;
1323            q: char;
1324
1325
1326          BEGIN
1327            writedb('in procedure collectdata');
1328            movewave(550.0);
1329            filterchanged := false;
1330            REPEAT
1331              prompt(top);
1332              IF NOT r_calibrated THEN
1333                BEGIN
1334                writeln('You MUST calibrate to a white reference before performing measurements');
1335                writeln(' -or- you can press ? and retrieve a calibration file from disk');
1336                writeln;
1337                END;
1338              write('Are you collecting data for a S(ample or white R(eference (S,R) [S] :  ');
1339              readch(q);
1340            UNTIL (q = 'R') OR r_calibrated;
1341            IF q = 'R' THEN
1342              BEGIN
1343              doreference := true;
1344              getwlinc;
1345              END
1346            ELSE doreference := false;
1347
1348            wave := startingwavelength;
1349            dbrd := dbrd AND 177737B; { set bit5 low, ... set for cyan filter in beam}
1350            instruct('prepare the photometer and mount the sample for this run');
1351            movewave(wave);
1352            prompt(top);
1353            IF spectralangles THEN
```

```
1354              BEGIN
1355              write('Enter the angle you would like to measure at :  ');
1356              readln(angle);
1357              setarm(angle);
1358              END;
1359            writeln('enter a 40 character description   [include geometry of measurement]');
1360            writeln('-->', ' ': 40, '<--');
1361            write('-->');
1362            readln(sample.descrip);
1363            FOR i := 1 TO numberofwavelengths DO sample.val[i] := 0;
1364            sample.inc := inc;
1365            prompt(top);
1366            getscale(starting_scale);
1367            REPEAT
1368              movewave(wave);
1369              IF ((wave >= wl_flip) AND NOT filterchanged) THEN
1370                BEGIN
1371                temp := getval;
1372                REPEAT
1373                  writeln('attempting to change filter');
1374                  dbrd := dbrd OR 40B;
1375                  twait(20 * between_reads_delay);
1376                  temp1 := getval;
1377                UNTIL abs(temp * flip_fraction - temp1) / temp1 < abs(1 - flip_fraction) / 3;
1378                writeln('continuing...');
1379                filterchanged := true;
1380                END;
1381              temp := getval;
1382              sample.val[(wave - 375) DIV 5] := temp;
1383              writeln(wave: 3, ' nm ==> ', temp);
1384              wave := wave + inc;
1385            UNTIL wave > endingwavelength;
1386            IF NOT rewind THEN instruct('cover the detector lens to prevent saturation');
1387            movewave(550.0);
1388            IF doreference THEN
1389              BEGIN
1390              prompt(top);
1391              writeln('Reading reference white reflectance values from file :  ', whiteref);
1392              retrievefile(whiteref, reference);
1393              prompt(top);
1394              reference.descrip := sample.descrip;
1395              reference.inc := sample.inc;
1396              FOR i := 1 TO numberofwavelengths DO
1397                BEGIN
1398                IF sample.val[i] = 0.0 THEN reference.val[i] := 0.0
1399                ELSE reference.val[i] := reference.val[i] / sample.val[i];
1400                END;
1401              prompt(top);
1402              write('do you wish to save the data for the white reference ? (Y,N) [Y] :  ');
1403              readch(q);
1404              r_calibrated := true;
1405              IF q <> 'N' THEN savefile(reference);
```

```
1406              END
1407            ELSE
1408              BEGIN
1409              FOR i := 1 TO numberofwavelengths DO sample.val[i] := sample.val[i] * reference.val[i];
1410              prompt(top);
1411              write('do you wish to save the data for this sample ? (Y,N) [Y] :  ');
1412              readch(q);
1413              IF q <> 'N' THEN savefile(sample);
1414              END;
1415          END;
1416
1417
1418        PROCEDURE menu;
1419
1420          VAR
1421            i: integer;
1422            wave, angle: real;
1423            q0, q1: char;
1424
1425
1426          BEGIN
1427            writedb('in procedure menu');
1428            i := 10;
1429            prompt(7);
1430            writeln(' C': i, ': Spectrogoniophotometer Mode Data Collection');
1431            writeln(' G': i, ': Goniophotometer Mode (White Light) Data Collection');
1432            writeln(' R': i, ': Retrieve Data from Disk');
1433            writeln(' D': i, ': DISPLAY values from Photometer');
1434            writeln(' W': i, ': test WAVELENGTH stepping motor');
1435            writeln(' A': i, ': test GONIO ARM stepping motor');
1436            writeln(' P': i, ': change program Parameters');
1437            writeln(' X': i, ': EXIT program');
1438            writeln;
1439            writeln;
1440            writeln;
1441            REPEAT
1442              BEGIN
1443              writeln(chr(27), 'A                                                    ');
1444              write(chr(27), 'ASelect C,G,R,D,W,A,P or X :  ');
1445              readch(q);
1446              END;
1447            UNTIL (q IN ['C', 'G', 'R', 'D', 'W', 'A', 'P', 'X']);
1448            CASE q OF
1449              'R':
1450                BEGIN
1451                prompt(top);
1452                write('do you wish to retrieve a S(ample or R(eference ? (S,R) [R] :  ');
1453                readch(q0);
1454                IF q0 = 'S' THEN getfile(sample)
1455                ELSE
1456                  BEGIN
1457                  getfile(reference);
```

```
1458                  r_calibrated := true;
1459                  inc := reference.inc;
1460                  END;
1461                END;
1462              'C': collectdata;
1463              'G':
1464                BEGIN
1465
1466                prompt(10);
1467                IF anglecalibrated THEN
1468                  BEGIN
1469                  write('do you wish to R(ecalibrate -or- U(se the current calibration (R,U) [U] :  ');
1470                  readch(q0);
1471                  IF q0 = 'R' THEN anglecalibrated := false;
1472                  END;
1473                IF NOT anglecalibrated THEN
1474                  BEGIN
1475                  writeln('please install your reference standard');
1476                  writeln;
1477                  write('enter the angle do you wish to calibrate at :  ');
1478                  readln(angle);
1479                  setarm(angle);
1480                  prompt(10);
1481                  write('enter wavelength you wish to calibrate at :  ');
1482                  readln(wave);
1483                  movewave(550.0);
1484                  instruct('align the standard to prepare for the measurement');
1485                  movewave(wave);
1486                  prompt(10);
1487                  write('enter the absolute reflectance of the standard at this angle :  ');
1488                  readln(anglefactor);
1489                  getscale( - 1);
1490                  anglefactor := anglefactor / getval;
1491                  anglecalibrated := true;
1492                  END;
1493                collectangular(asample);
1494                FOR i := 1 TO maxangles DO asample.val[i] := asample.val[i] * anglefactor;
1495                prompt(10);
1496                write('save this data to a disk file ? (Y,N) [Y] :  ');
1497                readch(q1);
1498                saveangular(asample);
1499
1500                END;
1501              'D': displayval;
1502              'W': wlstepr;
1503              'A': testarm;
1504              'P': chgpar;
1505              'X': ;
1506              END;
1507          END;
1508
1509
```

```
1510        BEGIN
1511          debug_wait := true;
1512          showreads := true;
1513          anglecalibrated := false;
1514          wavecalibrated := false;
1515          stepdelaymult[0] := 0.0;
1516          r_calibrated := false;
1517          FOR i := 1 TO 5 DO
1518            BEGIN
1519            stepdelaymult[i] := i / 5.0;
1520            stepdelaymult[ - 1 * i] := i / 5.0;
1521            END;
1522          csrd := csroutval;
1523          dbrd := 31;
1524          setdefaults;
1525          writedb('Past set defaults');
1526        1:
1527          BEGIN
1528          REPEAT
1529            BEGIN
1530            menu;
1531            END;
1532          UNTIL q = 'X';
1533          END;
1534          prompt(10);
1535          writeln('bye');
1536        END.

*** No lines with errors detected ***
```

```
                                ;  This code is for a sub program
                                ;   Sub Program File Name :  PTWAIT.MAC
                                ;   Other programs needed :  GONIO.PAS
                                ;   Compiling Command : r macro  PTWAIT=PASMAC,PTWAIT
                                ;   Linking Command :
                                ;   Hardware Required :  system clock
                                ;
                                ;Software Modifications/ Date/ Programmer
                                ;   Program Written/ 09-04-84/ R.M. Miller
                                ;   Procedure changed to time in 1/60 sec units instead of 1 sec units
                                ;    /04-28-85/ R.M. Miller
                                ;
                                ;                       BRIEF PROGRAM DESCRIPTION
                                ;
                                ;  This routine causes the a timed wait... (suspension of the users'
                                ;  job) for the requested number of seconds and is called by:
                                ;
                                ;     Procedure TWait(n: word); external;
                                ;
                                ;   where n  =  number of seconds to wait
                                ;
                                ;      word  =  0..65535;  {Pascal TYPE statement}
                                ;
                                ;   end of informational heading
                                ;
                                ;
                                        .title  twait
                                        .enabl  lc
                                        .mcall  .twait

000000                                  proc    twait
000000                                  param   n,integer
000000                                  save    <r0,r2,r3>
000000                                  begin

000012  016602  000010          start:  mov     n(sp),r2
                                ;       mul     #60.,r2   ; convert seconds to number of 1/60 sec ticks
000016  005003                          clr     r3
000020  010267  000050                  mov     r2,time+2
000024  010367  000042                  mov     r3,time
000030  012700  000066'                 mov     #area,r0
000034                                  .twait  #area,#time
000054                                  endpr

000066  000000  000000          area:   .word   0,0
000072  000000  000000          time:   .word   0,0
        000012'                         .end    start
```

```
Twait   MACRO V05.04  Sunday 09-Jul-89 15:53  Page 20-1
Symbol table

ADDRES= 000002          POINTE= 000002          Q$CUR  = 000006          Q$VRLN= 000000          REAL   = 000004
AREA    000066R         PP.PRO= 000000          Q$REGS= 000065          Q$$C1 = 177777          SCALAR= 000001
BOOLEA= 000001          PP.STA= 000002          Q$RGLN= 000006          Q$$C2 = 000200          START   000012R
CHAR   = 000001         PROCPA= 000004          Q$STAT= 000000          Q$$C3 = 000003          TIME    000072R
DOUBLE= 000010          P$127 = ****** G        Q$VIA = 177777          Q$$I0 = 000020          TWAIT   000000RG
INTEGE= 000002          Q$ARCT= 000001          Q$VRCT= 000000          Q$$R1 = 000004          ...V1 = 000003
N      = 000010         Q$ARLN= 000002

. ABS.  000000    000   (RW,I,GBL,ABS,OVR)
        000076    001   (RW,I,LCL,REL,CON)
Errors detected:  0

*** Assembler statistics


Work  file  reads: 0
Work  file writes: 0
Size of work file: 11749 Words  ( 46 Pages)
Size of core pool: 18432 Words  ( 72 Pages)
Operating  system: RT-11

Elapsed time: 00:00:21.53
,PTWAIT.LST=PASMAC,PTWAIT
```

APPENDIX B

PLOT PROGRAM

The plot program permits a user to plot data collected from the Munsell Color Science Laboratory Goniospectrophotometer and other other sources. The program consists of the following modules:

B1. P4N.PAS: the main program.

B2. P4NB.PAS: external user interface routines.

B3. P4NSPL.PAS: taut cubic spline and interpolation routines.

```
    1       PROGRAM plot_main;
    2
    3        { This code is for a main program
    4          Main Program File Name :  P4N.PAS
    5          Other subprograms needed :  P4NB,P4NSPL, MILPK2, MILPT2
    6          Compiling Command : pas P4N
    7          Linking Command :  @P4N.LNK
    8          Hardware Required :  HP7475 plotter
    9
   10       Software Modifications/ Date/ Programmer
   11          Program Written/ 1984/ R.M. Miller
   12
   13
   14                              BRIEF PROGRAM DESCRIPTION
   15
   16         PLOT can read data from COLOR program 77 point binary data files,
   17       XY data files, and OPTRONICS format data files to produce presentation
   18       quality plots on an HP7475 plotter.
   19
   20          end of informational heading }
   21
   22         CONST
   23           max_data = 300;
   24           max_pens = 8;
   25           number_of_labels = 8;
   26           max_overlays = 6;
   27           m = 'RIT Munsell Color Science Laboratory Plotting Package';
   28           singlechar = true;
   29           numberofwavelengths = 81;
   30           noofwavelengths = 77;
   31
   32           %INCLUDE 'string.typ';
2  1       TYPE
2  2         string3 = PACKED ARRAY [1..3] OF char;
2  3         string6 = PACKED ARRAY [1..6] OF char;
2  4         string14 = PACKED ARRAY [1..14] OF char;
   33
   34         TYPE
   35           string = PACKED ARRAY [1..40] OF char;
   36           string2 = PACKED ARRAY [1..2] OF char;
   37           pen_numb = 0..max_pens;
   38           labeldata = ARRAY [0..number_of_labels] OF string;
   39           file_types = (asciixy, color, optronics);
   40           values = ARRAY [1..max_data] OF real;
   41           breakdata = ARRAY [1..max_data] OF real;
   42           sdata = ARRAY [1..max_data, 1..6] OF real;
   43           coefdata = ARRAY [1..4, 1..max_data] OF real;
   44           pen_color_type = ARRAY [pen_numb] OF string14;
   45
   46           xydata =
   47             RECORD
   48               xdata, ydata: values;
   49               gamma, xmin, xmax, ymin, ymax: real;
```

```
50              filename: string14;
51              descrip: string;
52              deriv, order, number_of_points, number_of_knots, line_type: integer;
53              pen: pen_numb;
54              cubic_spline: boolean;
55              plotting_character: char;
56              use_standard_character_set: boolean;
57            END;
58
59          data =
60            RECORD
61              descrip: string;
62              inc: integer;
63              val: ARRAY [1..numberofwavelengths] OF real;
64            END;
65
66          plot_data = ARRAY [1..max_overlays] OF xydata;
67
68        VAR
69          number_of_copies, index, file_number, number_of_files, i, j, k, l, standard_set,
70          alternate_set: integer;
71          startwave, stopwave, waveinc, scalex, scaley, xinc, xplot, yplot, plotxmin, plotxmax, plotymin,
72           plotymax: real;
73          plot: plot_data;
74          copies: string2;
75          defdev, defext: string3;
76          f: FILE OF data;
77          pen_color: pen_color_type;
78          data_type: file_types;
79          inp, out: text;
80          title, xlabel, ylabel: labeldata;
81          rotate_plot, print_logo: boolean;
82          axes_color, title_color, x_axis_label_color, y_axis_label_color, logo_color,
83          numbering_color: pen_numb;
84          q: char;
85
86        LABEL
87          1;
88
89
90        PROCEDURE Spline(VAR x: xydata;
91                         VAR break: breakdata;
92                         VAR coef: coefdata;
93                         VAR error: boolean);
94          EXTERNAL;
95
96
97        FUNCTION Interp(x: real;
98                        VAR break: breakdata;
99                        VAR coef: coefdata;
100                       VAR m, k, index: integer;
101                       VAR error: boolean;
```

```
  102                         order: integer): real;
  103          EXTERNAL;
  104
  105
  106        PROCEDURE get_options(VAR data_type: file_types;
  107                              VAR defdev, defext: string3;
  108                              VAR plot: plot_data;
  109                              VAR pen_color: pen_color_type;
  110                              VAR title, xlabel, ylabel: labeldata;
  111                              VAR axes_color, title_color, x_axis_label_color, y_axis_label_color,
  112                               logo_color, numbering_color: pen_numb;
  113                              VAR print_logo, rotate_plot: boolean;
  114                              VAR standard_set, alternate_set: integer);
  115          EXTERNAL;
  116
  117
  118        PROCEDURE get_data(VAR data_type: file_types;
  119                           VAR defdev, defext: string3;
  120                           VAR plot: plot_data;
  121                           VAR pen_color: pen_color_type;
  122                           VAR title, xlabel, ylabel: labeldata;
  123                           VAR axes_color, title_color, x_axis_label_color, y_axis_label_color,
  124                            logo_color, numbering_color: pen_numb;
  125                           VAR print_logo, rotate_plot: boolean;
  126                           VAR plotxmin, plotxmax, plotymin, plotymax: real;
  127                           VAR number_of_files, file_number: integer);
  128          EXTERNAL;
  129
  130        %INCLUDE 'milpt2.pas';
2   1       { This code is for a sub program
2   2         Main Program File Name :  MILPT2.PAS
2   3         Other subprograms needed :  any that use these MIL-PLOT facilities
2   4         Compiling Command : N/A
2   5         Linking Command :  N/A
2   6         Hardware Required : an HP-7475 compatible digital X-Y plotter
2   7
2   8      Software Modifications/ Date/ Programmer
2   9         Procedures Written/ 1984/ R.M. Miller
2  10         Procedure added to draw the RIT Munsell Color Science Lab logo/ 1984/ Chris Pearson
2  11
2  12                           BRIEF PROGRAM DESCRIPTION
2  13
2  14
2  15                          MIL-PLOT Plotting Routines
2  16
2  17      These procedures enable a user to easily plot data, shapes, labels, etc.,
2  18      by calling various Pascal procedures. To use these procedures in a program
2  19      the following directive must appear after the variable declaration area for
2  20      the main program:
2  21
2  22          include 'milpt2.pas';
2  23
```

```
2  24      The following procedures are included in the package:
2  25
2  26
2  27      procedure HP_init;
2  28      procedure HP_default;
2  29      procedure HP_page_feed;
2  30      procedure HP_reset_coordinate;
2  31      procedure HP_set_papersize(x: papersize);
2  32      procedure HP_get_pen(x: pen_number);
2  33      procedure HP_put_label(x: packed array[lower..upper: integer] of char);
2  34      procedure HP_set_velocity(x: real);
2  35      procedure HP_pen(x: pen_controls);
2  36      procedure HP_move_pen(x,y: integer; z: pen_controls);
2  37      procedure HP_move_relative(x,y: integer; z: pen_controls);
2  38      procedure HP_draw_circle(x:integer);
2  39      procedure HP_turn_plotter(x: control);
2  40      procedure HP_set_symbol_mode(x: control; ch: char);
2  41      procedure HP_set_line_type(x:control; c: integer);
2  42      procedure HP_set_char_size(w,h: real);
2  43      procedure HP_scale_char_size(w: real);
2  44      procedure HP_turn_scaling(x:control; xmin,max,ymin,ymax: real);
2  45      procedure HP_set_plot_size(x:control; p1x,p1y,p2x,p2y: integer);
2  46      procedure HP_rotate_coordinates(x: control);
2  47      procedure HP_set_tick_length(x: control; tp,tn: real);
2  48      procedure HP_draw_x_tick;
2  49      procedure HP_draw_y_tick;
2  50
2  51
2  52
2  53
2  54
2  55                      ******  MAXIMUM PLOTTING RANGES ******
2  56
2  57                                  A4                      A3
2  58
2  59          X-axis units          0 - 10,365              0 - 16,640
2  60          X-axis dimension       10.15 in                16.3 in
2  61
2  62
2  63          Y-axis units          0 - 7,962               0 - 10,365
2  64          Y-axis dimension        7.80 in                10.15 in
2  65
2  66
2  67         end of informational heading )
2  68
2  69      TYPE
2  70        papersize = (large, small);
2  71        pen_number = 0..8;
2  72        pen_controls = (up, down);
2  73        control = (on, off);
2  74        character_set = (standard, alternate);
2  75
```

```
2  76
2  77      PROCEDURE HP_designate_character_set(cs: character_set;
2  78                                           n: integer);
2  79
2  80        BEGIN
2  81          IF cs = standard THEN write(out, 'CS')
2  82          ELSE write(out, 'CA');
2  83          writeln(out, n: 2, ';')
2  84        END;
2  85
2  86
2  87      PROCEDURE HP_set_character_set(cs: character_set);
2  88
2  89        BEGIN
2  90          IF cs = standard THEN writeln(out, 'SS;')
2  91          ELSE writeln(out, 'SA;');
2  92        END;
2  93
2  94
2  95      PROCEDURE HP_page_feed;
2  96
2  97        BEGIN
2  98          writeln(out, 'PG;');
2  99        END;
2 100
2 101
2 102      PROCEDURE HP_reset_coordinates;
2 103
2 104        BEGIN
2 105          writeln(out, 'IP;');
2 106        END;
2 107
2 108
2 109      PROCEDURE HP_init;
2 110
2 111        BEGIN
2 112          writeln(out, 'IN;');
2 113        END;
2 114
2 115
2 116      PROCEDURE HP_default;
2 117
2 118        BEGIN
2 119          writeln(out, 'DF;');
2 120        END;
2 121
2 122
2 123      PROCEDURE HP_set_papersize(x: papersize);
2 124
2 125        BEGIN
2 126
2 127      { This routine commented out because the plotter is not being used in a mode
```

```
2 128         that requires this command }
2 129
2 130           {  if x=large then writeln(out,'PS0;') else writeln(out,'PS4;');}
2 131         END;
2 132
2 133
2 134       PROCEDURE HP_get_pen(x: pen_number);
2 135
2 136         BEGIN
2 137           writeln(out, 'SP', x: 1, ';');
2 138         END;
2 139
2 140
2 141       PROCEDURE HP_put_label(x: PACKED ARRAY [lower..upper: integer] OF char);
2 142
2 143         VAR
2 144           i, iend: integer;
2 145
2 146         BEGIN
2 147           write(out, 'LB');
2 148           iend := upper;
2 149           IF x[upper] = chr(0) THEN WHILE (iend > lower - 1) AND (x[iend] = chr(0)) DO iend := iend - 1
2 150           ELSE WHILE (iend > lower - 1) AND (x[iend] = ' ') DO iend := iend - 1;
2 151           FOR i := lower TO iend DO write(out, x[i]);
2 152           writeln(out, chr(3));
2 153         END;
2 154
2 155
2 156       PROCEDURE HP_write_number(r: real);
2 157
2 158         BEGIN
2 159           write(out, 'LB');
2 160           IF r = 0.0 THEN write(out, '    0.000')
2 161           ELSE IF (abs(r) > 9999.9995) OR (abs(r) < 0.010) THEN write(out, r: 9: - 2)
2 162           ELSE write(out, r: 9: 3);
2 163           writeln(out, chr(3));
2 164         END;
2 165
2 166
2 167       PROCEDURE HP_set_velocity(x: real);
2 168
2 169         BEGIN
2 170           IF (x >= 0) AND (x < 128) THEN writeln(out, 'VS', x: 8: 4, ';')
2 171           ELSE writeln(out, 'VS;');
2 172         END;
2 173
2 174
2 175       PROCEDURE HP_pen(x: pen_controls);
2 176
2 177         BEGIN
2 178           IF x = up THEN writeln(out, 'PU;')
2 179           ELSE writeln(out, 'PD;');
```

```
2 180         END;
2 181
2 182
2 183       PROCEDURE HP_move_pen(x, y: real;
2 184                             z: pen_controls);
2 185
2 186         BEGIN
2 187           write(out, 'PAP');
2 188           IF z = up THEN write(out, 'U')
2 189           ELSE write(out, '0');
2 190           writeln(out, x: 9: 3, ',', y: 9: 3, ';');
2 191         END;
2 192
2 193
2 194       PROCEDURE HP_move_relative(x, y: real;
2 195                                  z: pen_controls);
2 196
2 197         BEGIN
2 198           write(out, 'PRP');
2 199           IF z = up THEN write(out, 'U')
2 200           ELSE write(out, '0');
2 201           writeln(out, x: 9: 3, ',', y: 9: 3, ';');
2 202         END;
2 203
2 204
2 205       PROCEDURE HP_draw_circle(x: real);
2 206
2 207         BEGIN
2 208           writeln(out, 'CI', x: 9: 3, ';');
2 209         END;
2 210
2 211
2 212       PROCEDURE HP_turn_plotter(x: control);
2 213
2 214         BEGIN
2 215
2 216       { This routine commented out because the plotter is not being used in a mode
2 217         that requires this command }
2 218
2 219       { write(out,chr(27),'.');
2 220         if x=on then write(out,'(') else writeln(out,')'); }
2 221         END;
2 222
2 223
2 224       PROCEDURE HP_set_symbol_mode(x: control;
2 225                                    ch: char);
2 226
2 227         BEGIN
2 228           IF x = off THEN writeln(out, 'SM;')
2 229           ELSE writeln(out, 'SM', ch, ';');
2 230         END;
2 231
```

```
2 232
2 233       PROCEDURE HP_set_line_type(x: control;
2 234                                  c: integer);
2 235
2 236         BEGIN
2 237           IF (x = off) OR (c > 6) OR (c < 0) THEN writeln(out, 'LT;')
2 238           ELSE writeln(out, 'LT', c: 1, ';');
2 239         END;
2 240
2 241
2 242       PROCEDURE HP_set_char_size(w, h: real);
2 243
2 244         BEGIN
2 245           IF (abs(w) <= 1E-6) OR (abs(h) <= 1E-6) OR (abs(w) > 128) OR (abs(h) > 128) THEN
2 246             writeln(out, 'SI;')
2 247           ELSE writeln(out, 'SI', w: 8: 4, ',', h: 8: 4, ';');
2 248         END;
2 249
2 250
2 251       PROCEDURE HP_scale_char_size(w: real);
2 252
2 253         BEGIN
2 254           IF (abs(w) <= 1E-6) OR (abs(w) > 128) THEN writeln(out, 'SR;')
2 255           ELSE writeln(out, 'SR', w: 8: 4, ',', (2 * w): 8: 4, ';');
2 256         END;
2 257
2 258
2 259       PROCEDURE HP_turn_scaling(x: control;
2 260                                 xmin, xmax, ymin, ymax: real);
2 261
2 262         BEGIN
2 263           IF x = off THEN writeln(out, 'SC;')
2 264           ELSE writeln(out, 'SC', xmin: 7: 0, ',', xmax: 7: 0, ',', ymin: 7: 0, ',', ymax: 7: 0, ';');
2 265         END;
2 266
2 267
2 268       PROCEDURE HP_set_plot_size(x: control;
2 269                                  p1x, p1y, p2x, p2y: integer);
2 270
2 271         BEGIN
2 272           IF x = off THEN writeln(out, 'IP;')
2 273           ELSE writeln(out, 'IP', p1x, ',', p1y, ',', p2x, ',', p2y, ';');
2 274         END;
2 275
2 276
2 277       PROCEDURE HP_rotate_coordinates(x: control);
2 278
2 279         BEGIN
2 280           IF x = off THEN writeln(out, 'RO;')
2 281           ELSE writeln(out, 'RO90;');
2 282         END;
2 283
```

```
2 284
2 285       PROCEDURE HP_set_tick_length(x: control;
2 286                                    tp, tn: real);
2 287
2 288         BEGIN
2 289           IF x = off THEN writeln(out, 'TL;')
2 290           ELSE writeln(out, 'TL', tp: 10: 4, ',', tn: 10: 4);
2 291         END;
2 292
2 293
2 294       PROCEDURE HP_draw_x_tick;
2 295
2 296         BEGIN
2 297           writeln(out, 'XT;');
2 298         END;
2 299
2 300
2 301       PROCEDURE HP_draw_y_tick;
2 302
2 303         BEGIN
2 304           writeln(out, 'YT;');
2 305         END;
2 306
2 307
2 308       PROCEDURE HP_set_handshake;
2 309
2 310         BEGIN
2 311
2 312       { This routine commented out because the plotter is not being used in a mode
2 313         that requires this command }
2 314
2 315           {  writeln(out,chr(27),'.I81;;17:',chr(27),'.N;19:');}
2 316         END;
2 317
2 318
2 319       PROCEDURE HP_set_labeling_direction(x: control;
2 320                                           angle: real);
2 321
2 322         VAR
2 323           rise, run: real;
2 324
2 325         BEGIN
2 326           rise := sin(angle / 180 * 3.1415927);
2 327           run := cos(angle / 180 * 3.1415927);
2 328           IF x = off THEN writeln(out, 'DI;')
2 329           ELSE writeln(out, 'DI', run: 10: 4, ',', rise: 10: 4, ';');
2 330         END;
2 331
2 332
2 333       PROCEDURE HP_set_plotting_window(x: control;
2 334                                        xll, yll, xur, yur: integer);
2 335
```

```
2 336         BEGIN
2 337           IF x = off THEN writeln(out, 'IW;')
2 338           ELSE writeln(out, 'IW', xll, ',', yll, ',', xur, ',', yur, ';');
2 339         END;
2 340
2 341
2 342       PROCEDURE HP_munsell_logo;
2 343
2 344         BEGIN
2 345
2 346         { The Munsell Logo procedure was written by Chris Pearson of the Munsell
2 347           Color Science Lab }
2 348
2 349           write(out, 'UC9,-1,100,-9,0,0,10,22,0,-100,3,0,100,20,0,0,3,8,-8,-8,-8,');
2 350           write(out, '0,3,-5,0,-100,0,10,100,0,-10,-2,-4,-1,-1,-4,-1,-5,0,0,3,-100,');
2 351           write(out, '0,4,100,0,3,5,0,4,1,1,1,2,4,0,1,-1,2,-1,1,-2,1,-4,1,-5,0,');
2 352           write(out, '-100,-5,0,100,-5,0,-4,-1,-2,-1,-1,-1,-1,-2,0,-1,2,-4,1,-1,');
2 353           write(out, '5,-1,5,0,0,14,-2,-1,5,8,3,-4,-1,-1,0,-10,-2,0,0,-6,-100,-3,3,');
2 354           write(out, '100,8,-8,-8,-8,-100,5,5,100,0,-6,-5,-5,0,9,-5,0,-5,1,-1,1,');
2 355           write(out, '-2,4,0,10');
2 356           writeln(out, ';');
2 357         END;
2 358
2 359
2 360       PROCEDURE HP_space_pen(s, l: real);
2 361
2 362         BEGIN
2 363           writeln(out, 'CP', s: 10: 4, ',', l: 10: 4);
2 364         END;
  131         %INCLUDE 'milpk2.pas';
2   1       { This code represents the MILPAK UTILITIES
2   2
2   3        { This code is for a sub program
2   4          Sub-Program File Name :  MILPK2.PAS
2   5          Other subprograms needed :
2   6          Compiling Command :
2   7          Linking Command :
2   8          Hardware Required :  none
2   9
2  10       Software Modifications/ Date/ Programmer
2  11          Program Written/ 10-20-84/ R.M. Miller
2  12          Procedure prompt changed to use a local procedure to avoid passing
2  13           the prompt label with each call to procedure PROMPT/ 10-27-84/ R.M. Miller
2  14
2  15
2  16                              BRIEF PROGRAM DESCRIPTION
2  17
2  18         Following are procedures for input io error trapping/checking for reading
2  19       reals, characters, strings, and integers, procedures for clearing the screen,
2  20       locating the curser, and prompting the user to continue.
2  21
2  22       SPECIAL NOTES:  a program using these utilities must adhere to the following
```

```
2  23                      conventions :
2  24
2  25          1. A label must be declared and assigned in the program as an abort
2  26             location for any io error trapping, for example:
2  27
2  28                          LABEL  1;  < to declare the label     >
2  29
2  30                          1: begin   < later in the program
2  31                             end;      to label a block of code >
2  32
2  33          2. Each call to Procedure Prompt must pass an integer parameter for the
2  34             line number the curser should be located. A string parameter having
2  35             variable name "m" that names the program and version number MUST be
2  36             declared as follows:
2  37
2  38                   constant m='MILPAK Utilities Version 2.0';
2  39
2  40              Call procedure prompt as follows:
2  41
2  42                   Prompt(10);
2  43
2  44          3. The program must call Procedure Startup before any program input
2  45             or input errors will not be trapped.
2  46
2  47
2  48          4. To include these utilities in a program, use the following command:
2  49
2  50                    %include 'milpak.pas';
2  51
2  52
2  53          5. A constant must be declared to indicate whether single character
2  54             activation will be used with the program. To use single character
2  55             activation make the following declaration:
2  56
2  57                    const  singlechar=true;
2  58
2  59             to disable single character activation make the following declaration:
2  60
2  61                    const  singlechar=false;
2  62
2  63             Single character activation enables the readch procedure to read a
2  64             character from the keyboard without the necessity of a carriage
2  65             return being pressed to terminate the response.
2  66
2  67           6. You must declare the following data types in the TYPE area using
2  68              the following command before the VAR section:
2  69
2  70              %include 'string.typ';
2  71
2  72
2  73
2  74        ***** LIST of procedures included in MILPAK:  *****
```

```
2  75
2  76
2  77         NOTE: This list is contained in MILPK2.EXT if needed  }
2  78
2  79       {START OF MILPK2.EXT}
2  80
2  81
2  82       PROCEDURE abort;
2  83         EXTERNAL;
2  84
2  85
2  86       PROCEDURE vt52;
2  87         EXTERNAL;
2  88
2  89
2  90       PROCEDURE gotoxy(x2, x1: integer);
2  91         EXTERNAL;
2  92
2  93
2  94       PROCEDURE readch(VAR q: char);
2  95         EXTERNAL;
2  96
2  97
2  98       PROCEDURE readre(s: PACKED ARRAY [lower..upper: integer] OF char;
2  99                        VAR r: real);
2 100         EXTERNAL;
2 101
2 102
2 103       PROCEDURE readin(s: PACKED ARRAY [lower..upper: integer] OF char;
2 104                        VAR r: integer);
2 105         EXTERNAL;
2 106
2 107
2 108       PROCEDURE readst(VAR s: PACKED ARRAY [lower..upper: integer] OF char);
2 109         EXTERNAL;
2 110
2 111
2 112       PROCEDURE return(x2: integer);
2 113         EXTERNAL;
2 114
2 115
2 116       PROCEDURE jump(x2: integer);
2 117         EXTERNAL;
2 118
2 119
2 120       PROCEDURE prompt(x1: integer);
2 121         EXTERNAL;
2 122
2 123
2 124       PROCEDURE writec(s: PACKED ARRAY [lower..upper: integer] OF char);
2 125         EXTERNAL;
2 126
```

```
2 127
2 128       PROCEDURE instruct(direction: PACKED ARRAY [lower..upper: integer] OF char);
2 129         EXTERNAL;
2 130
2 131
2 132       PROCEDURE startup;
2 133         EXTERNAL;
2 134
2 135
2 136       PROCEDURE getfn(VAR temp: string14;
2 137                       defdev, defext: string3);
2 138         EXTERNAL;
2 139
2 140
2 141       PROCEDURE readfn(VAR temp: string14;
2 142                        defdev, defext: string3);
2 143         EXTERNAL;
2 144
2 145       {END OF MILPK2.EXT}
2 146
2 147       {    ***** END of LIST *****
2 148
2 149
2 150
2 151          end of informational heading }
2 152
2 153
2 154       FUNCTION readsn: char;
2 155         EXTERNAL;
2 156
2 157
2 158       PROCEDURE abort;
2 159
2 160         BEGIN
2 161           writeln;
2 162           writec('press RETURN to ABORT');
2 163           readln;
2 164           GOTO 1;
2 165         END;
2 166
2 167
2 168       PROCEDURE vt52;
2 169
2 170         BEGIN
2 171           write(chr(27), '[?2l');
2 172         END;
2 173
2 174
2 175       PROCEDURE gotoxy;
2 176
2 177         BEGIN
2 178           write(chr(27), 'Y', chr(x1 + 32), chr(x2 + 32));
```

```
2 179         END;
2 180
2 181
2 182       PROCEDURE readch;
2 183
2 184         VAR
2 185           i: integer;
2 186           error: boolean;
2 187
2 188         BEGIN
2 189           q := ' ';
2 190           error := false;
2 191
2 192           IF singlechar THEN
2 193             BEGIN
2 194             q := readsn;
2 195             writeln(q);
2 196             END
2 197           ELSE
2 198             BEGIN
2 199             IF NOT eoln THEN read(q);
2 200             error := ioerror(input);
2 201             readln;
2 202             error := ioerror(input) OR error;
2 203             END;
2 204           IF ioerror(input) OR error THEN
2 205             BEGIN
2 206             IF iostatus(input) = 17 THEN
2 207               BEGIN
2 208               reset(input);
2 209               noioerror(input);
2 210               END;
2 211             q := ' ';
2 212             END;
2 213           i := ord(q);
2 214           IF i < 0 THEN i := i + 128;
2 215           IF (i = 13) AND (singlechar) THEN q := readsn;
2 216           IF (i = 10) OR (i = 27) OR (i = 13) THEN q := ' ';
2 217           IF q IN ['a'..'z'] THEN q := chr(ord(q) - 32);
2 218           IF q = '?' THEN GOTO 1;
2 219         END;
2 220
2 221
2 222       PROCEDURE readre;
2 223
2 224         VAR
2 225           error: boolean;
2 226           i: integer;
2 227           q, esc: char;
2 228
2 229         BEGIN
2 230           esc := chr(27);
```

```
2 231           writeln;
2 232           REPEAT
2 233             write(esc, 'A', esc, 'K');
2 234             FOR i := lower TO upper DO write(s[i]);
2 235             error := false;
2 236             IF NOT eoln THEN read(r)
2 237             ELSE error := true;
2 238             error := error OR ioerror(input);
2 239             readln;
2 240             error := error OR ioerror(input);
2 241             IF error THEN
2 242               BEGIN
2 243               write(esc, 'A', esc, 'K');
2 244               write('Real Input Error !!!  Press ? to abort -or- any other key to try again :  ');
2 245               readch(q);
2 246               END;
2 247           UNTIL NOT error;
2 248         END;
2 249
2 250
2 251       PROCEDURE readin;
2 252
2 253         VAR
2 254           error: boolean;
2 255           esc: char;
2 256           q: char;
2 257           i: integer;
2 258
2 259         BEGIN
2 260           esc := chr(27);
2 261           writeln;
2 262           REPEAT
2 263             write(esc, 'A', esc, 'K');
2 264             FOR i := lower TO upper DO write(s[i]);
2 265             error := false;
2 266             IF NOT eoln THEN read(r)
2 267             ELSE error := true;
2 268             error := error OR ioerror(input);
2 269             readln;
2 270             error := error OR ioerror(input);
2 271             IF error THEN
2 272               BEGIN
2 273               write(esc, 'A', esc, 'K');
2 274               write('Integer Input Error !!!  Press ? to abort -or- any other key to try again :  ');
2 275               readch(q);
2 276               END;
2 277           UNTIL NOT error;
2 278         END;
2 279
2 280
2 281       PROCEDURE readst;
2 282
```

```
2 283         VAR
2 284           error: boolean;
2 285           i: integer;
2 286           temp: PACKED ARRAY [1..100] OF char;
2 287
2 288         BEGIN
2 289           error := false;
2 290           read(temp);
2 291           error := error OR ioerror(input);
2 292           readln;
2 293           error := error OR ioerror(input);
2 294           IF error THEN FOR i := lower TO upper DO s[i] := ' '
2 295           ELSE
2 296             BEGIN
2 297             IF (upper - lower + 1) > 100 THEN
2 298               BEGIN
2 299               FOR i := lower TO (lower + 100 - 1) DO s[i] := temp[i - lower + 1];
2 300               FOR i := (lower + 100) TO upper DO s[i] := ' ';
2 301               END
2 302             ELSE FOR i := lower TO upper DO s[i] := temp[i - lower + 1]
2 303             END;
2 304           IF s[lower] = '?' THEN GOTO 1;
2 305         END;
2 306
2 307
2 308       PROCEDURE return;
2 309
2 310         VAR
2 311           i3: integer;
2 312           qr: char;
2 313
2 314         BEGIN
2 315           FOR i3 := 0 TO x2 DO writeln;
2 316           write(' ': 25, 'press any key to continue');
2 317           readch(qr);
2 318         END;
2 319
2 320
2 321       PROCEDURE jump;
2 322
2 323         VAR
2 324           i3: integer;
2 325
2 326         BEGIN
2 327           FOR i3 := 1 TO x2 DO
2 328             BEGIN
2 329             writeln;
2 330             END;
2 331         END;
2 332
2 333
2 334       PROCEDURE writec;
```

```
2 335
2 336         VAR
2 337           i: integer;
2 338
2 339         BEGIN
2 340           write(' ': ((80 - (upper - lower + 1)) DIV 2));
2 341           FOR i := lower TO upper DO write(s[i]);
2 342           writeln;
2 343         END;
2 344
2 345
2 346       PROCEDURE prompt;
2 347
2 348         VAR
2 349           x2: integer;
2 350
2 351         BEGIN
2 352           write(chr(27), 'H', chr(27), 'J');
2 353           writec(m);
2 354           gotoxy(0, x1 + 1);
2 355         END;
2 356
2 357
2 358       PROCEDURE instruct;
2 359
2 360         VAR
2 361           q: char;
2 362           i: integer;
2 363
2 364         BEGIN
2 365           prompt(10);
2 366           FOR i := lower TO upper DO write(direction[i]);
2 367           jump(4);
2 368           write('           press any key when this has been done');
2 369           readch(q);
2 370         END;
2 371
2 372
2 373       PROCEDURE startup;
2 374
2 375         BEGIN
2 376           noioerror(input);
2 377         END;
2 378
2 379
2 380       PROCEDURE getfn;
2 381
2 382         VAR
2 383           dev, ext: string3;
2 384           fn: string6;
2 385           nc, np, i, j, colon, period, endpos: integer;
2 386           error: boolean;
```

```
2 387           ch: char;
2 388
2 389         LABEL
2 390           2;
2 391
2 392         BEGIN
2 393           error := false;
2 394         2:
2 395           BEGIN
2 396           IF error THEN
2 397             BEGIN
2 398             writeln;
2 399             writeln('          Filename Entered by User :  ', temp);
2 400             writeln('WARNING!!! This file name is not in the proper format');
2 401             writeln('          The filename should be six characters long');
2 402             writeln('          with no intermediate spaces. The only legal');
2 403             writeln('          characters are letters and numbers. If you');
2 404             writeln('          wish a file can also be specified as :');
2 405             writeln('                              DEV:FILNAM.EXT');
2 406             error := false;
2 407             jump(2);
2 408             write('enter filename :  ');
2 409             readln(temp);
2 410             END;
2 411           END;
2 412           IF temp[1] = '?' THEN GOTO 1;
2 413           FOR i := 1 TO 14 DO IF temp[i] = ' ' THEN temp[i] := chr(0);
2 414           FOR i := 1 TO 14 DO
2 415             BEGIN
2 416             IF temp[i] = chr(0) THEN
2 417               BEGIN
2 418               FOR j := i TO 13 DO temp[j] := temp[j + 1];
2 419               temp[14] := chr(0);
2 420               END;
2 421             END;
2 422           nc := 0;
2 423           np := 0;
2 424           endpos := 15;
2 425           colon := 0;
2 426           FOR i := 1 TO 14 DO
2 427             BEGIN
2 428             IF temp[i] = ':' THEN
2 429               BEGIN
2 430               nc := nc + 1;
2 431               colon := i;
2 432               END;
2 433             IF temp[i] = '.' THEN
2 434               BEGIN
2 435               np := np + 1;
2 436               period := i;
2 437               END;
2 438             IF temp[15 - i] = chr(0) THEN endpos := 15 - i;
```

```
2 439           END;
2 440         IF np = 0 THEN period := endpos;
2 441         FOR i := 1 TO (endpos - 1) DO
2 442           BEGIN
2 443           ch := temp[i];
2 444           IF NOT (ch IN ['a'..'z', 'A'..'Z', '0'..'9', ':', '.']) THEN error := true;
2 445           END;
2 446         IF (period < colon) OR (colon = 1) OR (period = 1) OR (colon > 4) OR ((period - colon) > 7) OR
2 447            ((endpos - period) > 4) OR (period = colon + 1) THEN
2 448           error := true;
2 449         IF error THEN GOTO 2;
2 450         FOR i := 1 TO 3 DO
2 451           BEGIN
2 452           ext[i] := chr(0);
2 453           dev[i] := ext[i];
2 454           END;
2 455         FOR i := 1 TO 6 DO fn[i] := chr(0);
2 456         IF colon > 1 THEN
2 457           FOR i := 1 TO (colon - 1) DO dev[i] := temp[i]
2 458         ELSE dev := defdev;
2 459         FOR i := (colon + 1) TO (period - 1) DO fn[i - colon] := temp[i];
2 460         IF endpos > period THEN FOR i := (period + 1) TO (endpos - 1) DO ext[i - period] := temp[i]
2 461         ELSE ext := defext;
2 462         FOR i := 1 TO 3 DO temp[i] := dev[i];
2 463         FOR i := 1 TO 6 DO temp[i + 4] := fn[i];
2 464         FOR i := 1 TO 3 DO temp[i + 11] := ext[i];
2 465         temp[4] := ':';
2 466         temp[11] := '.';
2 467       END;
2 468
2 469
2 470     PROCEDURE readfn;
2 471
2 472       BEGIN
2 473         write('enter filename :  ');
2 474         readln(temp);
2 475         getfn(temp, defdev, defext);
2 476       END;
  132
  133
  134       FUNCTION xval(x: real): real;
  135
  136
  137         BEGIN
  138           xval := (x - plotxmin) * scalex;
  139         END;
  140
  141
  142       FUNCTION yval(y: real): real;
  143
  144
  145         BEGIN
```

```
146           yval := (y - plotymin) * scaley;
147         END;
148
149
150       PROCEDURE Do_Spline(VAR x: xydata);
151
152         VAR
153           break: breakdata;
154           coef: coefdata;
155           error: boolean;
156           xv, mx: real;
157
158
159         BEGIN
160           HP_turn_plotter(off);
161           Spline(x, break, coef, error);
162           HP_turn_plotter(on);
163           IF NOT error THEN
164             BEGIN
165             xinc := abs(plotxmax - plotxmin) / 500;
166             IF plotxmin < plotxmax THEN
167               BEGIN
168               IF plotxmin < x.xmin THEN xv := x.xmin
169               ELSE xv := plotxmin;
170               IF plotxmax > x.xmax THEN mx := x.xmax
171               ELSE mx := plotxmax;
172               END
173             ELSE
174               BEGIN
175               IF plotxmin > x.xmax THEN mx := x.xmax
176               ELSE mx := plotxmin;
177               IF plotxmax < x.xmin THEN xv := x.xmin
178               ELSE xv := plotxmax;
179               END;
180             index := 1;
181             REPEAT
182               HP_move_pen(xval(xv), yval(Interp(xv, break, coef, x.number_of_knots, x.order, index,
183                                                 error, x.deriv)), down);
184               xv := xv + xinc;
185             UNTIL error OR (xv > mx);
186             END;
187           IF error THEN
188             BEGIN
189             HP_turn_plotter(off);
190             writeln;
191             writeln('WARNING !!! Error encountered in INTERP');
192             return(2);
193             HP_turn_plotter(on);
194             END;
195         END;
196
197
```

```
198         BEGIN
199           startup;
200           get_options(data_type, defdev, defext, plot, pen_color, title, xlabel, ylabel, axes_color,
201                       title_color, x_axis_label_color, y_axis_label_color, logo_color, numbering_color,
202                       print_logo, rotate_plot, standard_set, alternate_set);
203           REPEAT
204             get_data(data_type, defdev, defext, plot, pen_color, title, xlabel, ylabel, axes_color,
205                      title_color, x_axis_label_color, y_axis_label_color, logo_color, numbering_color,
206                      print_logo, rotate_plot, plotxmin, plotxmax, plotymin, plotymax, number_of_files,
207                      file_number);
208
209             IF number_of_files > 0 THEN
210               BEGIN
211               REPEAT
212                 prompt(10);
213                 readln('Enter number of copies desired (0 - 99) :  ', number_of_copies);
214                 IF number_of_copies > 0 THEN
215                   BEGIN
216                   prompt(10);
217                   writeln('Please wait while the data is plotted');
218                   END;
219                 FOR j := 1 TO number_of_copies DO
220                   BEGIN
221                   rewrite(out, 'PLT:CURRNT.PLT');
222
223                   scalex := (100 / (plotxmax - plotxmin));
224                   scaley := (100 / (plotymax - plotymin));
225
226                   HP_turn_plotter(on);
227                   HP_set_handshake;
228                   HP_init;
229
230                   HP_designate_character_set(standard, standard_set);
231                   HP_designate_character_set(alternate, alternate_set);
232                   IF rotate_plot THEN
233                     BEGIN
234                     HP_rotate_coordinates(on);
235                     HP_set_plot_size(on, 600, 5000, 7200, 10300);
236                     END;
237
238                   HP_turn_scaling(on, - 16, 100, - 10, 115);
239                   HP_set_plotting_window(off, 0, 0, 0, 0);
240
241                   IF (plotymin <= 0.0) AND (plotymax >= 0.0) THEN yplot := 0.0
242                   ELSE yplot := plotymin;
243
244                   HP_get_pen(axes_color);
245                   HP_move_pen(100, yval(yplot), up);
246                   FOR i := 10 DOWNTO 0 DO
247                     BEGIN
248                     HP_move_pen(i * 10, yval(yplot), down);
249                     HP_draw_x_tick;
```

```
250                   END;
251                 IF (plotxmin <= 0.0) AND (plotxmax >= 0.0) THEN xplot := 0.0
252                 ELSE xplot := plotxmin;
253                 HP_move_pen(xval(xplot), 0, up);
254                 FOR i := 0 TO 10 DO
255                   BEGIN
256                   HP_move_pen(xval(xplot), i * 10, down);
257                   HP_draw_y_tick;
258                   END;
259                 HP_move_pen(5, 105, up);
260                 HP_scale_char_size(1.3);
261                 HP_get_pen(title_color);
262                 HP_put_label(title[0]);
263                 HP_scale_char_size(0.5);
264
265                 FOR file_number := 1 TO number_of_files DO
266                   BEGIN
267                   HP_get_pen(plot[file_number].pen);
268                   HP_move_pen(xval(plot[file_number].xdata[1]), yval(plot[file_number].ydata[1]), up);
269                   HP_set_line_type(on, plot[file_number].line_type);
270                   HP_set_symbol_mode(off, '+');
271                   IF plot[file_number].cubic_spline THEN Do_Spline(plot[file_number])
272                   ELSE
273                     BEGIN
274                     IF plot[file_number].plotting_character <> ' ' THEN
275                       BEGIN
276                       IF NOT plot[file_number].use_standard_character_set THEN
277                         HP_set_character_set(alternate);
278                       HP_set_symbol_mode(on, plot[file_number].plotting_character);
279                       HP_set_character_set(standard);
280                       END;
281                     IF plot[file_number].line_type = - 1 THEN
282                       FOR i := 1 TO plot[file_number].number_of_points DO
283                         HP_move_pen(xval(plot[file_number].xdata[i]), yval(plot[file_number].ydata[i]),
284                                     up)
285                     ELSE
286                       FOR i := 1 TO plot[file_number].number_of_points DO
287                         HP_move_pen(xval(plot[file_number].xdata[i]), yval(plot[file_number].ydata[i]),
288                                     down);
289                     END;
290                   END;
291                 HP_set_line_type(off, 0);
292                 HP_set_symbol_mode(off, '+');
293                 HP_scale_char_size(0.8);
294                 HP_get_pen(numbering_color);
295                 IF (plotymin <= 0.0) AND (plotymax >= 0.0) THEN yplot := 0.0
296                 ELSE yplot := plotymin;
297                 { write x axis numbers }
298                 FOR i := 1 TO 5 DO
299                   BEGIN
300                   HP_move_pen((i - 1) * 20 + 3, yval(yplot) - 5, up);
301                   xplot := (plotxmax - plotxmin) / 5 * (i - 1) + plotxmin + (plotxmax - plotxmin) / 10;
```

```
302                     IF (abs(xplot * 1000) < abs(plotxmax - plotxmin)) AND
303                        (abs(plotxmax - plotxmin) > 1) THEN
304                       xplot := 0.0;
305                     HP_write_number(xplot);
306                     END;
307
308                   IF (plotxmin <= 0.0) AND (plotxmax >= 0.0) THEN xplot := 0.0
309                   ELSE xplot := plotxmin;
310                   { write y axis numbers }
311                   FOR i := 1 TO 6 DO
312                     BEGIN
313                     HP_move_pen(xval(xplot) - 14, (i - 1) * 20 - 1, up);
314                     yplot := (plotymax - plotymin) / 5 * (i - 1) + plotymin;
315                     IF (abs(yplot * 1000) < abs(plotymax - plotymin)) AND
316                        (abs(plotymax - plotymin) > 1) THEN
317                       yplot := 0.0;
318                     HP_write_number(yplot);
319                     END;
320
321                   HP_scale_char_size(1);
322                   HP_move_pen(35, - 10, up);
323                   HP_get_pen(x_axis_label_color);
324                   HP_put_label(xlabel[0]);
325
326                   HP_scale_char_size(0.8);
327                   HP_move_pen(5, 101, up);
328                   HP_put_label(plot[1].descrip);
329
330                   HP_set_labeling_direction(on, 90);
331                   HP_move_pen( - 14, 10, up);
332                   HP_get_pen(y_axis_label_color);
333                   HP_put_label(ylabel[0]);
334
335                   HP_set_labeling_direction(off, 0);
336
337                   HP_scale_char_size(0.3);
338                   HP_move_pen( - 15, 105, up);
339
340                   IF print_logo THEN
341                     BEGIN
342                     HP_get_pen(logo_color);
343                     HP_put_label('Munsell');
344                     HP_munsell_logo;
345                     HP_space_pen(8, 0);
346                     HP_put_label('Laboratory');
347                     HP_space_pen( - 17, 1.5625);
348                     HP_put_label('Color');
349                     HP_space_pen( - 6, - 3.125);
350                     HP_put_label('Science');
351                     END;
352                   HP_get_pen(0);
353                   HP_turn_plotter(off);
```

```
354                   HP_page_feed;
355                   close(out);
356                   END;
357                 prompt(10);
358                 writec('Plotting Output Options');
359                 jump(2);
360                 writeln(' ': 20, 'C:  Make more copies of current plot');
361                 writeln(' ': 20, 'S:  Select new files to be plotted');
362                 writeln(' ': 20, 'Q:  QUIT this program');
363                 jump(2);
364                 write(' ': 16, 'Select one of the above (C,S,Q) [C] :  ');
365                 readch(q);
366               UNTIL (number_of_files = 0) OR (q = 'Q') OR (q = 'S');
367               END;
368           UNTIL (number_of_files = 0) OR (q = 'Q');
369           prompt(10);
370         1:
371           BEGIN
372           writeln('Bye');
373           END;
374         END.
```

*** No lines with errors detected ***

```
    1       PROGRAM plot_external;
    2
    3        {$nomain}
    4
    5        { This code is for a sub program
    6          Sub-Program File Name :  P4NB.PAS
    7          Other subprograms needed :  P4N, MILPK2
    8          Compiling Command : pas P4NB
    9          Linking Command :  @p4n.lnk
   10          Hardware Required :
   11
   12       Software Modifications/ Date/ Programmer
   13          Program Written/ 1984/ R.M. Miller
   14
   15
   16                              BRIEF PROGRAM DESCRIPTION
   17
   18       P4NB contains external routines called by PLOT
   19
   20          end of informational heading }
   21
   22         CONST
   23           max_data = 3D0;
   24           max_pens = 8;
   25           number_of_labels = 8;
   26           max_overlays = 6;
   27           m = 'RIT Munsell Color Science Laboratory Plotting Package';
   28           singlechar = true;
   29           numberofwavelengths = 81;
   30           noofwavelengths = 77;
   31
   32           %INCLUDE 'string.typ';
2  1       TYPE
2  2         string3 = PACKED ARRAY [1..3] OF char;
2  3         string6 = PACKED ARRAY [1..6] OF char;
2  4         string14 = PACKED ARRAY [1..14] OF char;
   33
   34         TYPE
   35           string = PACKED ARRAY [1..40] OF char;
   36           pen_numb = 0..max_pens;
   37           labeldata = ARRAY [0..number_of_labels] OF string;
   38           file_types = (asciixy, color, optronics);
   39           values = ARRAY [1..max_data] OF real;
   40           breakdata = ARRAY [1..max_data] OF real;
   41           sdata = ARRAY [1..max_data, 1..6] OF real;
   42           coefdata = ARRAY [1..4, 1..max_data] OF real;
   43           pen_color_type = ARRAY [pen_numb] OF string14;
   44
   45           xydata =
   46             RECORD
   47               xdata, ydata: values;
   48               gamma, xmin, xmax, ymin, ymax: real;
   49               filename: string14;
```

```
   50              descrip: string;
   51              deriv, order, number_of_points, number_of_knots, line_type: integer;
   52              pen: pen_numb;
   53              cubic_spline: boolean;
   54              plotting_character: char;
   55              use_standard_character_set: boolean;
   56            END;
   57
   58          data =
   59            RECORD
   60              descrip: string;
   61              inc: integer;
   62              val: ARRAY [1..numberofwavelengths] OF real;
   63            END;
   64
   65          plot_data = ARRAY [1..max_overlays] OF xydata;
   66
   67          { procedures contained in this module which are external to the main program }
   68
   69
   70        PROCEDURE get_options(VAR data_type: file_types;
   71                              VAR defdev, defext: string3;
   72                              VAR plot: plot_data;
   73                              VAR pen_color: pen_color_type;
   74                              VAR title, xlabel, ylabel: labeldata;
   75                              VAR axes_color, title_color, x_axis_label_color, y_axis_label_color,
   76                               logo_color, numbering_color: pen_numb;
   77                              VAR print_logo, rotate_plot: boolean;
   78                              VAR standard_set, alternate_set: integer);
   79          EXTERNAL;
   80
   81
   82        PROCEDURE get_data(VAR data_type: file_types;
   83                           VAR defdev, defext: string3;
   84                           VAR plot: plot_data;
   85                           VAR pen_color: pen_color_type;
   86                           VAR title, xlabel, ylabel: labeldata;
   87                           VAR axes_color, title_color, x_axis_label_color, y_axis_label_color,
   88                            logo_color, numbering_color: pen_numb;
   89                           VAR print_logo, rotate_plot: boolean;
   90                           VAR plotxmin, plotxmax, plotymin, plotymax: real;
   91                           VAR number_of_files, file_number: integer);
   92          EXTERNAL;
   93
   94          { procedures contained in the main program which are external to this module }
   95
   96          %INCLUDE 'milpk2.ext';
2   1       {START OF MILPK2.EXT}
2   2
2   3       {$nolist}
2  68       {$list}
2  69
```

```
2  70      {END OF MILPK2.EXT}
   97
   98
   99        PROCEDURE readbl(VAR inp: text;
  100                         VAR b: boolean);
  101
  102          VAR
  103            q: char;
  104
  105          BEGIN
  106            q := ' ';
  107            IF NOT eoln(inp) THEN readln(inp, q)
  108            ELSE readln(inp);
  109            b := (q = 'T') OR (q = 't');
  110          END;
  111
  112
  113        PROCEDURE get_labels(VAR x: labeldata;
  114                             m: PACKED ARRAY [lower..upper: integer] OF char);
  115
  116          VAR
  117            i, j: integer;
  118            q: char;
  119
  120          BEGIN
  121            j := number_of_labels + 1;
  122            REPEAT
  123              REPEAT
  124                prompt(8);
  125                write('standard plotting ');
  126                FOR i := lower TO upper DO write(m[i]);
  127                writeln('s');
  128                writeln;
  129                FOR i := 1 TO number_of_labels DO writeln(' ': 10, i: 1, ':  ', x[i]);
  130                write(' ': 10, j: 1, ':  < enter a non-standard ');
  131                FOR i := lower TO upper DO write(m[i]);
  132                writeln(' >');
  133                writeln;
  134                readin('Select one of the above :  ', i);
  135              UNTIL i IN [1..j];
  136              IF i = j THEN
  137                BEGIN
  138                writeln;
  139                write('enter ');
  140                FOR i := lower TO upper DO write(m[i]);
  141                write(':  ');
  142                readln(x[0]);
  143                END
  144              ELSE x[0] := x[i];
  145              prompt(8);
  146              FOR i := lower TO upper DO write(m[i]);
  147              writeln(' selected :  ', x[0]);
```

```
148             writeln;
149             write('Is this correct ? (Y,N) [Y] :  ');
150             readch(q);
151           UNTIL q <> 'N';
152         END;
153
154
155       PROCEDURE minmax(VAR x: xydata);
156
157         VAR
158           i: integer;
159
160         BEGIN
161           WITH x DO
162             BEGIN
163             ymin := ydata[1];
164             ymax := ydata[1];
165             xmin := xdata[1];
166             xmax := xdata[1];
167             FOR i := 2 TO number_of_points DO
168               BEGIN
169               IF ydata[i] > ymax THEN ymax := ydata[i];
170               IF ydata[i] < ymin THEN ymin := ydata[i];
171               IF xdata[i] > xmax THEN xmax := xdata[i];
172               IF xdata[i] < xmin THEN xmin := xdata[i];
173               END;
174             END;
175         END;
176
177
178       PROCEDURE wonval(v1: real);
179
180         BEGIN
181           IF (((v1 < 1000.0) AND (v1 > 0.001)) OR (v1 = 0.0)) THEN write(v1: 10: 4)
182           ELSE write(v1: 10: - 3);
183         END;
184
185
186       PROCEDURE select_min_max(VAR xmin, xmax, ymin, ymax: real;
187                                VAR plot: plot_data;
188                                number_of_files: integer);
189
190         VAR
191           i: integer;
192
193         BEGIN
194           prompt(2);
195           writeln('File            X                              Y            Description');
196           writeln(' No.  Minimum      Maximum      Minimum      Maximum');
197           FOR i := 1 TO number_of_files DO
198             BEGIN
199             write(i: 3, ' ': 2);
```

```
200             wonval(plot[i].xmin);
201             write(' ': 3);
202             wonval(plot[i].xmax);
203             write(' ': 3);
204             wonval(plot[i].ymin);
205             write(' ': 3);
206             wonval(plot[i].ymax);
207             writeln(' ': 2, plot[i].descrip);
208             END;
209           ymin := plot[1].ymin;
210           ymax := plot[1].ymax;
211           xmin := plot[1].xmin;
212           xmax := plot[1].xmax;
213           FOR i := 2 TO number_of_files DO
214             BEGIN
215             IF plot[i].ymax > ymax THEN ymax := plot[i].ymax;
216             IF plot[i].ymin < ymin THEN ymin := plot[i].ymin;
217             IF plot[i].xmax > xmax THEN xmax := plot[i].xmax;
218             IF plot[i].xmin < xmin THEN xmin := plot[i].xmin;
219             END;
220           write((20): 3, ' ': 2);
221           wonval(xmin);
222           write(' ': 3);
223           wonval(xmax);
224           write(' ': 3);
225           wonval(ymin);
226           write(' ': 3);
227           wonval(ymax);
228           writeln(' ': 2, 'OVERALL Min/Max of above');
229           writeln((30): 3, '  Enter your own Min/Max values for X and Y');
230           writeln((40): 3, '  Enter your own Min/Max values for X (use OVERALL for Y)');
231           writeln((50): 3, '  Enter your own Min/Max values for Y (use OVERALL for X)');
232           writeln((60): 3, '  Use  0..1   for Y (OVERALL for X)');
233           writeln((70): 3, '  Use  0..100 for Y (OVERALL for X)');
234           writeln;
235           writeln;
236           REPEAT
237             write(chr(27), 'A', chr(27), 'K');
238             readin('Select one of the above :  ', i);
239           UNTIL (i IN [1..number_of_files, 20, 30, 40, 50, 60, 70]);
240           CASE i OF
241             20: ;
242             30:
243               BEGIN
244               readre('enter xmin :  ', xmin);
245               readre('enter xmax :  ', xmax);
246               writeln;
247               readre('enter ymin :  ', ymin);
248               readre('enter ymax :  ', ymax);
249               END;
250             40:
251               BEGIN
```

```
252                readre('enter xmin :  ', xmin);
253                readre('enter xmax :  ', xmax);
254                END;
255              50:
256                BEGIN
257                readre('enter ymin :  ', ymin);
258                readre('enter ymax :  ', ymax);
259                END;
260              60:
261                BEGIN
262                ymin := 0;
263                ymax := 1;
264                END;
265              70:
266                BEGIN
267                ymin := 0;
268                ymax := 100;
269                END;
270              OTHERWISE
271                BEGIN
272                ymax := plot[i].ymax;
273                ymin := plot[i].ymin;
274                xmax := plot[i].xmax;
275                xmin := plot[i].xmin;
276                END;
277              END;
278          END;
279
280
281        PROCEDURE set_data_type(q: char;
282                                VAR data_type: file_types;
283                                VAR defdev, defext: string3);
284
285          BEGIN
286            CASE q OF
287              'O': data_type := optronics;
288              'X': data_type := asciixy;
289              OTHERWISE data_type := color;
290              END;
291            CASE data_type OF
292              optronics:
293                BEGIN
294                defdev := 'DK0';
295                defext := 'ORM';
296                END;
297              asciixy:
298                BEGIN
299                defdev := 'DK0';
300                defext := 'DAT';
301                END;
302              color:
3D3                BEGIN
```

```
304                defdev := 'STU';
305                defext := 'DAT';
306                END;
307              END;
308          END;
309
310
311        PROCEDURE find_data_type(VAR data_type: file_types;
312                                 VAR defdev, defext: string3);
313
314          VAR
315            q: char;
316            i: integer;
317
318          BEGIN
319            i := 10;
320            gotoxy(0, max_overlays + 5);
321            write(chr(27), 'J');
322            writeln('Data types that can be plotted with this program');
323            jump(2);
324            writeln(' ': i, 'C:  Color data file format');
325            writeln(' ': i, 'O:  Optronics data file format');
326            writeln(' ': i, 'X:  X,Y data (with description on first line)');
327            jump(2);
328            write('enter type of data files you will plot (C,O,X) [C] :  ');
329            readch(q);
330            set_data_type(q, data_type, defdev, defext);
331          END;
332
333
334        PROCEDURE get_options;
335
336          VAR
337            i: integer;
338            inp: text;
339            q: char;
340            fn: PACKED ARRAY [1..14] OF char;
341
342          BEGIN
343            prompt(10);
344            writec('Welcome to the MCSL plotting package');
345            jump(4);
346            writeln('Use C(urrent default options -or- S(pecify a special default');
347            write('  option file ? (C,S) [C] :  ');
348            readch(q);
349            i := - 1;
350            jump(2);
351            IF q = 'S' THEN
352              BEGIN
353              write('enter filename for plotting options (DEV:FILNAM.EXT)  :  ');
354              readst(fn);
355              reset(inp, fn, , i);
```

```
356             END;
357           IF i = - 1 THEN reset(inp, 'dk:milplt.def', , i);
358           IF i = - 1 THEN reset(inp, 'sy:milplt.def', , i);
359           IF i = - 1 THEN
360             BEGIN
361             writeln('WARNING!!! Files: SY:MILPLT.DEF and DK:MILPLT.DEF  NOT FOUND');
362             abort;
363             END
364           ELSE
365             BEGIN
366             readln(inp, q);
367             set_data_type(q, data_type, defdev, defext);
368             readln(inp, standard_set);
369             readln(inp, alternate_set);
370             FOR i := 1 TO max_pens DO readln(inp, pen_color[i]);
371             FOR i := 1 TO number_of_labels DO readln(inp, title[i]);
372             FOR i := 1 TO number_of_labels DO readln(inp, xlabel[i]);
373             FOR i := 1 TO number_of_labels DO readln(inp, ylabel[i]);
374             readln(inp, axes_color);
375             readln(inp, title_color);
376             readln(inp, x_axis_label_color);
377             readln(inp, y_axis_label_color);
378             readln(inp, numbering_color);
379             readln(inp, logo_color);
380             readbl(inp, print_logo);
381             readbl(inp, rotate_plot);
382             FOR i := 1 TO max_overlays DO
383               BEGIN
384               readln(inp, plot[i].pen);
385               readln(inp, plot[i].line_type);
386               readbl(inp, plot[i].use_standard_character_set);
387               readln(inp, plot[i].plotting_character);
388               readbl(inp, plot[i].cubic_spline);
389               readln(inp, plot[i].deriv);
390               readln(inp, plot[i].gamma);
391               END;
392             close(inp);
393             END;
394         END;
395
396
397       PROCEDURE get_data;
398
399         VAR
400           i, j, k, l, m: integer;
401           startwave, stopwave, waveinc: real;
402           f: FILE OF data;
403           q: char;
404           inp: text;
405
406         BEGIN
407           number_of_files := 0;
```

```
408             REPEAT
409               REPEAT
410                 l := - 1;
411                 prompt(0);
412                 gotoxy(2, max_overlays + 3);
413                 write('Current File Format :  ');
414                 CASE data_type OF
415                   color: writeln('COLOR PROGRAM Data');
416                   optronics: writeln('OPTRONICS Data');
417                   asciixy: writeln('STANDARD X,Y data');
418                   END;
419                 gotoxy(0, 2);
420                 FOR i := 1 TO number_of_files DO
421                   writeln('File ', i: 2, '  "', plot[i].filename, '" :   ', plot[i].descrip);
422                 gotoxy(0, max_overlays + 5);
423                 writeln('enter  >  to change current file format, $ to stop reading files,');
424                 write('  -or-   enter name of data file you wish to plot :  ');
425                 WITH plot[number_of_files + 1] DO
426                   BEGIN
427                   WHILE eoln DO
428                     BEGIN
429                     readln;
430                     writeln;
431                     write(chr(27), 'A', chr(27),
432                           'K  -or-   enter name of data file you wish to plot :  ');
433                     END;
434                   readln(filename);
435                   CASE filename[1] OF
436                     '>':
437                       BEGIN
438                       gotoxy(10, max_overlays + 5);
439                       write(chr(27), 'J');
440                       find_data_type(data_type, defdev, defext);
441                       END;
442                     '$': l := 9999;
443                     OTHERWISE
444                       BEGIN
445                       getfn(filename, defdev, defext);
446                       CASE data_type OF
447                         color: reset(f, filename, , l);
448                         optronics: reset(inp, filename, , l);
449                         asciixy: reset(inp, filename, , l);
450                         END;
451                       IF l = - 1 THEN
452                         BEGIN
453                         writeln;
454                         writeln('WARNING !!!! File does not exist');
455                         writeln;
456                         write('press RETURN to try again ');
457                         readch(q);
458                         END
459                       ELSE
```

```
460                     BEGIN
461                     CASE data_type OF
462                       optronics:
463                         BEGIN
464                         readln(inp, descrip);
465                         readln(inp, startwave, stopwave, waveinc);
466                         i := 1;
467                         WHILE NOT (eof(inp)) AND (i <= max_data) DO
468                           BEGIN
469                           readln(inp, ydata[i]);
470                           xdata[i] := startwave;
471                           startwave := startwave + waveinc;
472                           i := i + 1;
473                           END;
474                         number_of_points := i - 1;
475                         END;
476                       asciixy:
477                         BEGIN
478                         readln(inp, descrip);
479                         i := 1;
480                         WHILE NOT (eof(inp)) AND (i <= max_data) DO
481                           BEGIN
482                           readln(inp, xdata[i], ydata[i]);
483                           i := i + 1;
484                           END;
485                         number_of_points := i - 1;
486                         END;
487                       color:
488                         BEGIN
489                         get(f);
490                         descrip := f^.descrip;
491                         j := (f^.inc DIV 5);
492                         i := 1;
493                         k := 1;
494                         WHILE (f^.val[k] = 0) AND (k < numberofwavelengths) DO k := k + 1;
495                         REPEAT
496                           xdata[i] := 375 + k * 5;
497                           ydata[i] := f^.val[k];
498                           i := i + 1;
499                           k := k + j;
500                         UNTIL k > noofwavelengths;
501                         number_of_points := i - 1;
502                         END;
503
504                       END; { case data_type of }
505                     FOR i := i TO max_data DO
506                       BEGIN
507                       xdata[i] := 0;
508                       ydata[i] := 0;
509                       END;
510                     IF data_type = color THEN close(f)
511                     ELSE close(inp);
```

```
512                       END;
513                     END;
514                   ENO;
515                 END;
516             UNTIL (l <> - 1) OR (l = 9999);
517             IF (l <> 9999) THEN number_of_files := number_of_files + 1;
518           UNTIL (l = 9999) OR (number_of_files = max_overlays);
519           IF number_of_files > 0 THEN
520             BEGIN
521             get_labels(title, 'title');
522             get_labels(xlabel, 'x-axis label');
523             get_labels(ylabel, 'y-axis label');
524             FOR m := 1 TO number_of_files DO minmax(plot[m]);
525             select_min_max(plotxmin, plotxmax, plotymin, plotymax, plot, number_of_files);
526             END;
527         END;
```

*** No lines with errors detected ***

```
1       PROGRAM Spline_Interp_for_Plot;
2
3        {$nomain}
4
5        { This code is for a sub program
6          Sub-Program File Name :  P4NSPL.PAS
7          Other subprograms needed :  P4N.PAS
8          Compiling Command : pas P4NSPL
9          Linking Command :  @P4N.LNK
10         Hardware Required :  none
11
12      Software Modifications/ Date/ Programmer
13         Program Written/ 1984/ R.M. Miller
14
15
16                            BRIEF PROGRAM DESCRIPTION
17
18      P4NSPL contains the procedures to calculate a taut cubic spline of
19      supplied data and to interpolate the derivative of interest.
20
21         end of informational heading }
22
23        CONST
24          max_data = 300;
25
26        TYPE
27          string = PACKED ARRAY [1..40] OF char;
28          string14 = PACKED ARRAY [1..14] OF char;
29          pen_numb = 0..6;
30          values = ARRAY [1..max_data] OF real;
31          breakdata = ARRAY [1..max_data] OF real;
32          sdata = ARRAY [1..max_data, 1..6] OF real;
33          coefdata = ARRAY [1..4, 1..max_data] OF real;
34
35          xydata =
36            RECORD
37              xdata, ydata: values;
38              gamma, xmin, xmax, ymin, ymax: real;
39              filename: string14;
40              descrip: string;
41              deriv, order, number_of_points, number_of_knots, line_type: integer;
42              pen: pen_numb;
43              cubic_spline: boolean;
44              plotting_character: char;
45            END;
46
47
48        PROCEDURE Spline(VAR xx: xydata;
49                         VAR break: breakdata;
50                         VAR coef: coefdata;
51                         VAR error: boolean);
52          EXTERNAL;
53
```

```
54
55          FUNCTION Interp(x: real;
56                          VAR break: breakdata;
57                          VAR coef: coefdata;
58                          VAR m, k, index: integer;
59                          VAR error: boolean;
60                          order: integer): real;
61            EXTERNAL;
62
63
64          PROCEDURE Spline;
65
66            VAR
67              i, j, m, method, ntaum1: integer;
68              alpha, c, d, del, denom, divdif, entry, entry3, factor, factr2, gam, onemg3, onemzt, ratio,
69               sixth, temp, x, z, zeta, zt2: real;
70              s: sdata;
71
72            LABEL
73              9;
74
75              { Function ALPH local to procedure SPLINE }
76
77
78            FUNCTION alph(x: real): real;
79
80              VAR
81                x1: real;
82
83              BEGIN
84                x1 := onemg3 / x;
85                IF 1 < x1 THEN alph := 1
86                ELSE alph := x1;
87              END;
88
89
90            PROCEOURE docalc1;
91
92              BEGIN
93                coef[2, m] := divdif - s[i, 1] * (2 * s[i, 4] + s[i + 1, 4]) / 6;
94                coef[4, m] := (s[i + 1, 4] - s[i, 4]) / s[i, 1];
95              ENO;
96
97
98            PROCEDURE docalc2;
99
100             BEGIN
101               onemzt := gam * (1 - z);
102               IF onemzt = 0 THEN
103                 BEGIN
104                 coef[2, m] := divdif;
105                 coef[3, m] := 0;
```

```
106                 coef[4, m] := 0;
107                 ENO
108               ELSE
109                 BEGIN
110                 zeta := 1 - onemzt;
111                 alpha := alph(zeta);
112                 c := s[i + 1, 4] * s[i, 6];
113                 d := s[i, 4] / 6;
114                 del := zeta * s[i, 1];
115                 break[m + 1] := xx.xdata[i] + del;
116                 coef[2, m] := divdif - s[i, 1] * (2 * d + c);
117                 coef[4, m] := 6 * (c * alpha - d) / s[i, 1];
118                 m := m + 1;
119                 coef[4, m] := coef[4, m - 1] + 6 * (1 - alpha) * c / (s[i, 1] * sqr(onemzt) * onemzt);
120                 coef[3, m] := coef[3, m - 1] + del * coef[4, m - 1];
121                 coef[2, m] := coef[2, m - 1] + del * (coef[3, m - 1] + (del / 2) * coef[4, m - 1]);
122                 coef[1, m] := coef[1, m - 1] + del * (coef[2, m - 1] + (del / 2) * (coef[3, m - 1] +
123                               (del / 3) * coef[4, m - 1]));
124                 ENO;
125             END;
126
127
128           PROCEOURE docalc3;
129
130             BEGIN
131               IF z = 0 THEN
132                 BEGIN
133                 coef[2, m] := divdif;
134                 coef[3, m] := 0;
135                 coef[4, m] := 0;
136                 ENO
137               ELSE
138                 BEGIN
139                 zeta := gam * z;
140                 onemzt := 1 - zeta;
141                 c := s[i + 1, 4] / 6;
142                 d := s[i, 4] * s[i, 6];
143                 m := m + 1;
144                 del := zeta * s[i, 1];
145                 break[m] := xx.xdata[i] + del;
146                 zt2 := sqr(zeta);
147                 alpha := alph(onemzt);
148                 factor := sqr(onemzt) * alpha;
149                 coef[1, m] := xx.ydata[i] + divdif * del + sqr(s[i, 1]) * (d * onemzt * (factor - 1) + c *
150                               zeta * (zt2 - 1));
151                 coef[2, m] := divdif + s[i, 1] * (d * (1 - 3 * factor) + c * (3 * zt2 - 1));
152                 coef[3, m] := 6 * (d * alpha * onemzt + c * zeta);
153                 coef[4, m] := 6 * (c - d * alpha) / s[i, 1];
154                 coef[4, m - 1] := coef[4, m] - 6 * d * (1 - alpha) / (del * zt2);
155                 coef[2, m - 1] := coef[2, m] - del * (coef[3, m] - (del / 2) * coef[4, m - 1]);
156                 ENO;
157             END;
```

```
158
159
160         BEGIN { Procedure SPLINE }
161
162           FOR i := 1 TO max_data DO
163             BEGIN
164             FOR j := 1 TO 6 DO s[i, j] := 0;
165             END;
166
167           FOR i := 1 TO max_data DO
168             BEGIN
169             FOR j := 1 TO 4 DO coef[j, i] := 0;
170             break[i] := 0;
171             END;
172
173           error := false;
174           IF xx.number_of_points < 4 THEN
175             BEGIN
176             writeln('Number of data points = ', xx.number_of_points: 1,
177                     '.  This must be greater than 4');
178             writeln('   to use this SPLINE routine !!!');
179             error := true;
180             GOTO 9;
181             END;
182           ntaum1 := xx.number_of_points - 1;
183           FOR i := 1 TO ntaum1 DO
184             BEGIN
185             s[i, 1] := xx.xdata[i + 1] - xx.xdata[i];
186             IF s[i, 1] <= 0 THEN
187               BEGIN
188               writeln('Data points are disordered !!!');
189               writeln;
190               writeln('Data [', i, '] = ', xx.xdata[i], '     ', 'Data [', (i + 1), '] = ',
191                       xx.xdata[i + 1]);
192               GOTO 9;
193               END;
194             s[i + 1, 4] := (xx.ydata[i + 1] - xx.ydata[i]) / s[i, 1];
195             END;
196           FOR i := 2 TO ntaum1 DO s[i, 4] := s[i + 1, 4] - s[i, 4];
197
198           i := 2;
199           s[2, 2] := s[1, 1] / 3;
200           sixth := 1 / 6;
201           method := 2;
202           gam := xx.gamma;
203           IF gam <= 0 THEN method := 1
204           ELSE IF gam > 3 THEN
205             BEGIN
206             gam := gam - 3;
207             method := 3;
208             END;
209           onemg3 := 1 - gam / 3;
```

```
210             FOR i := 2 TO (ntaum1 - 1) DO
211               BEGIN
212               z := 0.5;
213               IF method <> 1 THEN
214                 BEGIN
215                 IF ((method = 2) AND ((s[i, 4] * s[i + 1, 4]) >= 0)) OR (method = 3) THEN
216                   BEGIN
217                   temp := abs(s[i + 1, 4]);
218                   denom := abs(s[i, 4]) + temp;
219                   IF denom <> 0 THEN
220                     BEGIN
221                     z := temp / denom;
222                     IF abs(z - 0.5) <= sixth THEN z := 0.5;
223                     END;
224                   END;
225                 END;
226               s[i, 5] := z;
227
228               IF z = 0.5 THEN
229                 BEGIN
230                 s[i, 2] := s[i, 2] + s[i, 1] / 3;
231                 s[i, 3] := s[i, 1] / 6;
232                 END
233               ELSE IF z < 0.5 THEN
234                 BEGIN
235                 zeta := gam * z;
236                 onemzt := 1 - zeta;
237                 zt2 := sqr(zeta);
238                 alpha := alph(onemzt);
239                 factor := zeta / (alpha * (zt2 - 1) + 1);
240                 s[i, 6] := zeta * factor / 6;
241                 s[i, 2] := s[i, 2] + s[i, 1] * ((1 - alpha * onemzt) * factor / 2 - s[i, 6]);
242                 IF s[i, 2] <= 0 THEN s[i, 2] := 1;
243                 s[i, 3] := s[i, 1] / 6;
244                 END
245               ELSE
246                 BEGIN
247                 onemzt := gam * (1 - z);
248                 zeta := 1 - onemzt;
249                 alpha := alph(zeta);
250                 factor := onemzt / (1 - alpha * zeta * (1 + onemzt));
251                 s[i, 6] := onemzt * factor / 6;
252                 s[i, 2] := s[i, 2] + s[i, 1] / 3;
253                 s[i, 3] := s[i, 6] * s[i, 1];
254                 END;
255
256               IF i = 2 THEN
257                 BEGIN
258                 s[1, 5] := 0.5;
259                 s[1, 2] := s[1, 1] / 6;
260                 s[1, 3] := s[2, 2];
261                 entry3 := s[2, 3];
```

```
262                 IF z = 0.5 THEN
263                   BEGIN
264                   ratio := s[2, 1] / s[1, 2];
265                   s[2, 2] := s[2, 1] + s[1, 1];
266                   s[2, 3] := - 1 * s[1, 1];
267                   END
268                 ELSE IF z > 0.5 THEN
269                   BEGIN
270                   ratio := s[2, 1] / s[1, 2];
271                   s[2, 2] := s[2, 1] + s[1, 1];
272                   s[2, 3] := - 1 * s[1, 1] * 6 * alpha * s[2, 6];
273                   END
274                 ELSE
275                   BEGIN
276                   factr2 := zeta * (alpha * (zt2 - 1) + 1) / (alpha * (zeta * zt2 - 1) + 1);
277                   ratio := factr2 * s[2, 1] / s[1, 2];
278                   s[2, 2] := factr2 * s[2, 1] + s[1, 1];
279                   s[2, 3] := - 1 * s[1, 1] * factr2;
280                   END;
281                 s[2, 2] := ratio * s[1, 3] + s[2, 2];
282                 s[2, 3] := ratio * entry3 + s[2, 3];
283                 s[1, 4] := s[2, 4];
284                 s[2, 4] := ratio * s[1, 4];
285                 END
286               ELSE
287                 BEGIN
288                 s[i, 2] := ratio * s[i - 1, 3] + s[i, 2];
289                 s[i, 4] := ratio * s[i - 1, 4] + s[i, 4];
290                 END;
291
292               IF z = 0.5 THEN
293                 BEGIN
294                 ratio := - 1 * (s[i, 1] / 6) / s[i, 2];
295                 s[i + 1, 2] := s[i, 1] / 3;
296                 END
297               ELSE IF z > 0.5 THEN
298                 BEGIN
299                 ratio := - 1 * (s[i, 1] / 6) / s[i, 2];
300                 s[i + 1, 2] := s[i, 1] * ((1 - zeta * alpha) * factor / 2 - s[i, 6]);
301                 END
302               ELSE
303                 BEGIN
304                 ratio := - 1 * (s[i, 6] * s[i, 1] / s[i, 2]);
305                 s[i + 1, 2] := s[i, 1] / 3;
306                 END;
307               END;
308             i := ntaum1;
309             s[i, 5] := 0.5;
310             entry := ratio * s[i - 1, 3] + s[i, 2] + s[i, 1] / 3;
311             s[i + 1, 2] := s[i, 1] / 6;
312             s[i + 1, 4] := ratio * s[i - 1, 4] + s[i, 4];
313
```

```
314             IF z = 0.5 THEN
315               BEGIN
316               ratio := s[i, 1] / s[i - 1, 2];
317               s[i, 2] := ratio * s[i - 1, 3] + s[i, 1] + s[i - 1, 1];
318               s[i, 3] := - 1 * s[i - 1, 1];
319               END
320             ELSE IF z > 0.5 THEN
321               BEGIN
322               factr2 := onemzt * (alpha * (sqr(onemzt) - 1) + 1) / (alpha * (sqr(onemzt) * onemzt - 1) +
323                         1);
324               ratio := factr2 * s[i, 1] / s[i - 1, 2];
325               s[i, 2] := ratio * s[i - 1, 3] + factr2 * s[i - 1, 1] + s[i, 1];
326               s[i, 3] := - 1 * factr2 * s[i - 1, 1];
327               END
328             ELSE
329               BEGIN
330               ratio := s[i, 1] * 6 * s[i - 1, 6] * alpha / s[i - 1, 2];
331               s[i, 2] := ratio * s[i - 1, 3] + s[i, 1] + s[i - 1, 1];
332               s[i, 3] := - 1 * s[i - 1, 1];
333               END;
334
335             s[i, 4] := ratio * s[i - 1, 4];
336             ratio := - 1 * entry / s[i, 2];
337             s[i + 1, 2] := ratio * s[i, 3] + s[i + 1, 2];
338             s[i + 1, 4] := ratio * s[i, 4] + s[i + 1, 4];
339
340             { Back Substitution }
341
342             s[xx.number_of_points, 4] := s[xx.number_of_points, 4] / s[xx.number_of_points, 2];
343             FOR i := i DOWNTO 2 DO s[i, 4] := (s[i, 4] - s[i, 3] * s[i + 1, 4]) / s[i, 2];
344             s[1, 4] := (s[1, 4] - s[1, 3] * s[2, 4] - entry3 * s[3, 4]) / s[1, 2];
345
346             { Construct Polynomial Pieces }
347
348             break[1] := xx.xdata[1];
349             m := 1;
350
351             FOR i := 1 TO ntaum1 DO
352               BEGIN
353               coef[1, m] := xx.ydata[i];
354               coef[3, m] := s[i, 4];
355               divdif := (xx.ydata[i + 1] - xx.ydata[i]) / s[i, 1];
356               z := s[i, 5];
357               IF z = 0.5 THEN docalc1
358               ELSE IF z > 0.5 THEN docalc2
359               ELSE docalc3;
360               m := m + 1;
361               break[m] := xx.xdata[i + 1];
362               END;
363
364             xx.number_of_knots := m - 1;
365             xx.order := 4;
```

```
366
367           9:
368             BEGIN
369             IF error THEN
370               BEGIN
371               writeln;
372               writeln('Procedure SPLINE');
373               write('ERROR was encountered... Press RETURN to continue ');
374               readln;
375               END;
376             END;
377           END; { Procedure SPLINE }
378
379
380         FUNCTION Interp;
381
382           VAR
383             j, fmmjdr, n, p: integer;
384             temp, dx: real;
385
386           LABEL
387             9;
388
389
390           BEGIN { Procedure INTERP }
391             temp := 0;
392             IF error THEN GOTO 9;
393
394             fmmjdr := k - order;
395             IF fmmjdr <= 0 THEN
396               BEGIN
397               error := true;
398               GOTO 9;
399               END;
400
401             IF (x < break[1]) OR (x > break[m + 1]) THEN
402               BEGIN
403               error := true;
404               GOTO 9;
405               END;
406
407             WHILE x > break[index + 1] DO index := index + 1;
408             dx := x - break[index];
409             p := fmmjdr;
410             FOR n := k DOWNTO (order + 1) DO
411               BEGIN
412               temp := temp / p * dx + coef[n, index];
413               p := p - 1;
414               END;
415
416           9:
417             Interp := temp;
```

```
  418
  419         END;
```

*** No lines with errors detected ***

APPENDIX C

CONVERT PROGRAM

The convert program permits a user to convert data between various data types used in the laboratory. The program is commonly used to convert between ASCII xy data and binary format. The program consists of the following module:

C1. CONVRT.PAS: a data file conversion program.

```
1       PROGRAM convert;
2
3        { This code is for a main program
4          Main Program File Name :  CONVRT.PAS
5          Other subprograms needed :  none
6          Compiling Command : pas CONVRT
7          Linking Command :  link CONVRT,PAS2.FPP
8          Hardware Required :  none
9
10      Software Modifications/ Date/ Programmer
11         Program Written/ 08-16-84/ R.M. Miller
12         User notified of attempts to read non-existing files
13           and given an opportunity to re-enter a filename/ 09-11-84/ R.M. Miller
14         Option added to allow a user to calculate the mean and standard deviation
15           of spectral data files/ 11-29-84/ R.M. Miller
16         Smoothing option added/ 1984/ R.M. Miller
17
18                           BRIEF PROGRAM DESCRIPTION
19
20         This program converts data between ASCII and binary formats for
21      color program data (77 data point files) and converts data from the
22      Tracor (512 point files) into the X,Y format (wavelength,value) for
23      plotting. 512 point data can also be converted into 77 point data.
24      Data may also be averaged and the standard deviation determined as
25      a function of wavelength.  Smoothing is available for 77 point color
26      data.
27
28         end of informational heading }
29
30        CONST
31          stringmax = 7;
32          version = 3.3;
33          top = 5;
34          space = 3;
35          numberofwavelengths = 81;
36          noofwavelengths = 77;
37          display = true;
38
39        TYPE
40          string78 = PACKED ARRAY [1..78] OF char;
41          string = PACKED ARRAY [1..40] OF char;
42          string7 = PACKED ARRAY [1..7] OF char;
43          string6 = PACKED ARRAY [1..6] OF char;
44          string10 = PACKED ARRAY [1..10] OF char;
45          string14 = PACKED ARRAY [1..14] OF char;
46          data2 =
47            RECORD
48              time: real;
49              val: ARRAY [0..511] OF real;
50              descrip: PACKED ARRAY [1..40] OF char;
51              code: string78;
52            END;
53          data =
```

```
54            RECORD
55              descrip: string;
56              inc: integer;
57              val: ARRAY [1..numberofwavelengths] OF real;
58            END;
59
60        VAR
61          enter: data;
62          fn, fnr, fnw: string14;
63          f: FILE OF data;
64          q: char;
65          len: integer;
66          out, t: text;
67          sample: data2;
68          i, j: integer;
69          value: real;
70
71        LABEL
72          1;
73
74
75        PROCEDURE prompt(x: integer);
76
77          VAR
78            i: integer;
79
80          BEGIN
81            write(chr(27), 'H', chr(27), 'J                    Data Conversion Program Version ',
82                  version: 3: 1);
83            FOR i := 1 TO x DO writeln;
84          END;
85
86
87        PROCEDURE readch(VAR q: char);
88
89          BEGIN
90            q := ' ';
91            IF NOT eoln THEN readln(q)
92            ELSE readln;
93            IF q IN ['a'..'z'] THEN q := chr(ord(q) - 32);
94            IF q = '?' THEN GOTO 1;
95          END;
96
97
98        PROCEDURE debug(s: PACKED ARRAY [lower..upper: integer] OF char);
99
100         VAR
101           i: integer;
102           q: char;
103
104         BEGIN
105           IF display THEN
```

```
106             BEGIN
107             writeln;
108             FOR i := lower TO upper DO write(s[i]);
109             writeln;
110             write('Press RETURN to continue');
111             readch(q);
112             writeln;
113             END;
114         END;
115
116
117       PROCEDURE savefile;
118
119         VAR
120           i: integer;
121           q: char;
122
123         BEGIN
124           writeln;
125           write('enter filename for data  (DEV:FILNAM) :  ');
126           readln(fnw);
127           REPEAT
128             BEGIN
129             reset(f, fnw, 'DAT', len);
130             IF len <> - 1 THEN
131               BEGIN
132               close(f);
133               writeln('WARNING!!!! file name : ', fnw, '  already exists');
134               write(' D(elete this file and use same name -OR- E(nter new name [E]:  ');
135               readch(q);
136               IF q <> 'D' THEN
137                 BEGIN
138                 write('enter new filename for data  (DEV:FILNAM) :  ');
139                 readln(fnw);
140                 END;
141               END;
142             END;
143           UNTIL (q = 'D') OR (len = - 1);
144         END;
145
146
147       PROCEDURE Optronics;
148
149         VAR
150           start, stop, i: integer;
151           startwave, stopwave, incwave: real;
152
153         BEGIN
154           FOR i := 1 TO numberofwavelengths DO enter.val[i] := 0.0;
155           i := 1;
156           REPEAT
157             BEGIN
```

```
158             prompt(10);
159             IF i = - 1 THEN writeln('Filename: ', fnr, '  NOT FOUNO !!!  Please');
160             writeln('enter file name of Optronics data you wish to convert');
161             write('  (OEV:FILNAM) :  ');
162             readln(fnr);
163             IF fnr[1] = '?' THEN GOTO 1;
164             reset(t, fnr, 'orm', i);
165             ENO;
166           UNTIL i <> - 1;
167           readln(t, enter.descrip);
168           readln(t, startwave, stopwave, incwave);
169           enter.inc := round(incwave);
170           start := round(startwave);
171           stop := round(stopwave);
172           WHILE start < 380 DO
173             BEGIN
174             readln(t, stopwave);
175             start := start + enter.inc;
176             ENO;
177           IF ((enter.inc MOO 5) <> 0) OR ((start MOO 5) <> 0) THEN
178             BEGIN
179             writeln('!!! OPTRONICS ERROR !!! Invalid conditions. ');
180             writeln;
181             writeln('Starting wavelength = ', startwave: 6: 1, ' nm');
182             writeln('Wavelength Increment= ', incwave: 6: 1, ' nm');
183             writeln;
184             write('press RETURN to continue');
185             readch(q);
186             GOTO 1;
187             ENO;
188           REPEAT
189             readln(t, enter.val[(start - 380) OIV 5 + 1]);
190             start := start + enter.inc;
191           UNTIL (start > 760) OR (start > stop);
192           close(t);
193           savefile;
194           rewrite(f, fnw, 'OAT');
195           f^ := enter;
196           put(f);
197           close(f);
198         ENO;
199
200
201       PROCEOURE textbin;
202
203         VAR
204           i: integer;
205
206         BEGIN
207           FOR i := 1 TO numberofwavelengths OO enter.val[i] := 0.0;
208           i := 1;
209           REPEAT
```

```
210             BEGIN
211             prompt(10);
212             IF i = - 1 THEN writeln('Filename: ', fnr, '  NOT FOUND !!!  Please');
213             writeln('enter file name of text data you wish to convert');
214             write('  (DEV:FILNAM) :  ');
215             readln(fnr);
216             IF fnr[1] = '?' THEN GOTO 1;
217             reset(t, fnr, 'dat', i);
218             END;
219           UNTIL i <> - 1;
220           readln(t, enter.descrip);
221           readln(t, enter.inc);
222           FOR i := 1 TO ((380 DIV enter.inc) + 1) DO
223             BEGIN
224             readln(t, enter.val[(enter.inc DIV 5) * (i - 1) + 1]);
225             END;
226           close(t);
227           savefile;
228           rewrite(f, fnw, 'DAT');
229           f^ := enter;
230           put(f);
231           close(f);
232         END;
233
234
235       PROCEDURE smooth;
236
237         VAR
238           n, m, i, j, k, ka, wave, number, numpts: integer;
239           nsum: real;
240           ndata, mdata: ARRAY [1..100] OF real;
241           np: ARRAY [1..9] OF real;
242           q: char;
243
244         BEGIN
245           i := 1;
246           REPEAT
247             prompt(10);
248             IF i = - 1 THEN writeln('Filename: ', fnr, '  NOT FOUND !!!  Please Re-');
249             write('enter filename (DEV:FILNAM.EXT) :  ');
250             readln(fnr);
251             IF fnr[1] = '?' THEN GOTO 1;
252             reset(f, fnr, , i);
253           UNTIL (i <> - 1);
254           get(f);
255           enter := f^;
256           REPEAT
257             prompt(10);
258             writeln('Select one of the following :');
259             writeln;
260             writeln('5:': 10, '   5 point least squares cubic');
261             writeln('7:': 10, '   7 point least squares cubic');
```

```
262             writeln('9:': 10, '   9 point least squares cubic');
263             writeln;
264             write('enter choice (5,7,9) [5] :  ');
265             readch(q);
266             IF q = '7' THEN number := 3
267             ELSE IF q = '9' THEN number := 4
268             ELSE number := 2;
269             FOR i := 1 TO number DO ndata[i] := enter.val[1];
270             j := number + 1;
271             wave := 380;
272             REPEAT
273               ndata[j] := enter.val[((wave - 380) DIV 5) + 1];
274               j := j + 1;
275               wave := wave + enter.inc;
276             UNTIL wave > 760;
277             wave := wave - enter.inc;
278             FOR i := j TO (j + number - 1) DO ndata[i] := enter.val[((wave - 380) DIV 5) + 1];
279             n := j + number - 1;
280             m := n - 2 * number;
281             FOR i := 2 TO (2 * number + 1) DO np[i] := ndata[i - 1];
282             FOR i := 1 TO m DO
283               BEGIN
284               FOR k := 1 TO (2 * number) DO np[k] := np[k + 1];
285               np[2 * number + 1] := ndata[i + 2 * number];
286               IF number = 4 THEN
287                 BEGIN
288                 mdata[i] := (59 * np[5] + 54 * (np[4] + np[6]) + 39 * (np[3] + np[7]) + 14 * (np[2] +
289                             np[8]) - 21 * (np[1] + np[9])) / 231.0;
290                 END
291               ELSE IF number = 3 THEN
292                 BEGIN
293                 mdata[i] := (7 * np[4] + 6 * (np[3] + np[5]) + 3 * (np[2] + np[6]) - 2 * (np[1] +
294                             np[7])) / 21.0;
295                 END
296               ELSE mdata[i] := (17 * np[3] + 12 * (np[2] + np[4]) - 3 * (np[1] + np[5])) / 35.0;
297               END;
298             wave := 380;
299             FOR i := 1 TO m DO
300               BEGIN
301               enter.val[((wave - 380) DIV 5) + 1] := mdata[i];
302               wave := wave + enter.inc;
303               END;
304             savefile;
305             rewrite(f, fnw);
306             f^ := enter;
307             put(f);
308             close(f);
309             prompt(10);
310             write('do you wish to smooth this data an additional time ? (Y,N) [N] :  ');
311             readch(q);
312           UNTIL q <> 'Y';
313         END;
```

```
314
315
316         PROCEDURE meansdev;
317
318           VAR
319             i, n, inc: integer;
320             sum, sum2: data;
321             q: char;
322
323           BEGIN
324             n := 0;
325             inc := 0;
326             prompt(2);
327             writeln('This routine will average spectral data files. After specifying the');
328             writeln('desired data file names, type a  $  to do the calculations');
329             writeln;
330             FOR i := 1 TO numberofwavelengths DO sum.val[i] := 0.0;
331             sum2 := sum;
332             REPEAT
333               i := 1;
334               REPEAT
335                 IF i = - 1 THEN writeln('Filename: ', fnr, '  NOT FOUND !!!  Please Re-');
336                 write('enter filename (DEV:FILNAM.EXT) : ');
337                 readln(fnr);
338                 IF fnr[1] = '?' THEN GOTO 1;
339                 IF fnr[1] <> '$' THEN reset(f, fnr, , i);
340               UNTIL (i <> - 1) OR (fnr[1] = '$');
341               IF fnr[1] <> '$' THEN
342                 BEGIN
343                 get(f);
344                 writeln('    File Description :  ', f^.descrip);
345                 IF f^.inc > inc THEN inc := f^.inc;
346                 n := n + 1;
347                 FOR i := 1 TO 77 DO
348                   BEGIN
349                   sum.val[i] := sum.val[i] + f^.val[i];
350                   sum2.val[i] := sum2.val[i] + sqr(f^.val[i]);
351                   END;
352                 END;
353             UNTIL fnr[1] = '$';
354             IF n < 2 THEN
355               BEGIN
356               writeln('WARNING !!! you must enter at least 2 data files to average');
357               write('press RETURN to continue');
358               readch(q);
359               END
360             ELSE
361               BEGIN
362               writeln;
363               FOR i := 1 TO 77 DO enter.val[i] := sum.val[i] / n;
364               writeln('First enter a filename for the average spectral values');
365               savefile;
```

```
366             writeln('Number of spectral data files averaged :  ', n: 3);
367             write('enter a descrption for these averaged data :  ');
368             readln(enter.descrip);
369             rewrite(f, fnw);
370             enter.inc := inc;
371             f^ := enter;
372             put(f);
373             close(f);
374             FOR i := 1 TO 77 DO enter.val[i] := sqrt(sum2.val[i] / n - sqr(sum.val[i] / n));
375             writeln('Now enter a filename for the standard deviations of the spectral values');
376             savefile;
377             writeln('Number of spectral data files averaged :  ', n: 3);
378             write('enter a descrption for these standard deviations :  ');
379             readln(enter.descrip);
380             rewrite(f, fnw);
381             f^ := enter;
382             put(f);
383             close(f);
384             END;
385         END;
386
387
388       PROCEDURE bintext;
389
390         VAR
391           i, j, field, ext: integer;
392           format: boolean;
393
394         BEGIN
395           i := 1;
396           REPEAT
397             BEGIN
398             prompt(10);
399             IF i = - 1 THEN writeln('Filename: ', fnr, '  NOT FOUND !!!  Please');
400             writeln('enter file name of binary data you wish to convert');
401             write('  (DEV:FILNAM) :  ');
402             readln(fnr);
403             IF fnr[1] = '?' THEN GOTO 1;
404             reset(f, fnr, 'dat', i);
405             END;
406           UNTIL i <> - 1;
407           write('Do you wish to format data : ? (Y/N) [N] :  ');
408           readch(q);
409           format := false;
410           IF q = 'Y' THEN
411             BEGIN
412             format := true;
413             write('enter field width :  ');
414             readln(field);
415             write('enter extension   :  ');
416             readln(ext);
417             END;
```

```
418             write('convert all data ? (Y/N) [Y] :  ');
419             readch(q);
420             j := 1;
421             IF q = 'N' THEN
422               BEGIN
423               REPEAT
424                 BEGIN
425                 write('enter wavelength interval (5 - 50) in nm :  ');
426                 readln(j);
427                 j := j DIV 5;
428                 END;
429               UNTIL ((j > 0) AND (j < 11));
430               END;
431             get(f);
432             enter := f^;
433             close(f);
434             savefile;
435             rewrite(t, fnw, 'DAT');
436             writeln(t, enter.descrip);
437             writeln(t, enter.inc);
438             i := 1;
439             WHILE i <= noofwavelengths DO
440               BEGIN
441               IF format THEN writeln(t, enter.val[i]: field: ext)
442               ELSE writeln(t, enter.val[i]);
443               i := i + j;
444               END;
445             close(t);
446           END;
447
448
449         PROCEDURE tracnv;
450
451           VAR
452             i: integer;
453
454           BEGIN
455             i := 1;
456             REPEAT
457               BEGIN
458               prompt(10);
459               IF i = - 1 THEN writeln('Filename: ', fn, '  NOT FOUND !!!  Please');
460               write('enter filename of TRACOR data (DEV:FILNAM) :  ');
461               readln(fn);
462               IF fn[1] = '?' THEN GOTO 1;
463               reset(t, fn, 'REC', i);
464               END;
465             UNTIL i <> - 1;
466             rewrite(out, fn, 'PRT');
467             FOR i := 0 TO 511 DO
468               BEGIN
469               readln(t, value);
```

```
470             writeln(out, (240.8 + i * (870.8 - 240.8) / 511), ',', value);
471             END;
472           close(t);
473           close(out);
474         END;
475
476
477       FUNCTION wave(cell: integer): real;
478
479         BEGIN
480           wave := 240.8 + cell * (870.8 - 240.8) / 511;
481         END;
482
483
484       PROCEDURE trareduce;
485
486         VAR
487           i, cell: integer;
488           value: ARRAY [0..511] OF real;
489           counter, extra, width, desiredwave, desiredwidth: real;
490
491         BEGIN
492           i := 1;
493           width := (870.8 - 240.8) / 511;
494           desiredwidth := 5.0;
495           REPEAT
496             BEGIN
497             prompt(10);
498             IF i = - 1 THEN writeln('Filename: ', fn, '  NOT FOUND !!!  Please');
499             write('enter filename of TRACOR data (DEV:FILNAM) :  ');
500             readln(fn);
501             IF fn[1] = '?' THEN GOTO 1;
502             reset(t, fn, 'REC', i);
503             END;
504           UNTIL i <> - 1;
505           FOR i := 0 TO 511 DO readln(t, value[i]);
506           close(t);
507           cell := 100;
508           desiredwave := 380;
509           WHILE ((wave(cell) + width / 2.0) < (desiredwave - desiredwidth / 2.0)) DO cell := cell + 1;
510           cell := cell + 1;
511           extra := wave(cell) + width / 2.0 - (desiredwave - desiredwidth / 2.0);
512           i := 0;
513           REPEAT
514             BEGIN
515             i := i + 1;
516             counter := extra;
517             enter.val[i] := extra / width * value[cell];
518             WHILE (counter + width) <= desiredwidth DO
519               BEGIN
520               cell := cell + 1;
521               enter.val[i] := enter.val[i] + value[cell];
```

```
522                 counter := counter + width;
523                 END;
524               IF (counter >= 5.0) THEN
525                 BEGIN
526                 extra := 0.0;
527                 END
528               ELSE
529                 BEGIN
530                 extra := desiredwave + desiredwidth / 2.0 - (wave(cell) - width / 2.0);
531                 enter.val[i] := enter.val[i] + extra / width * value[cell];
532                 extra := width - extra;
533                 END;
534               desiredwave := desiredwave + desiredwidth;
535               enter.val[i] := enter.val[i] / 5.0 * width;
536               END;
537             UNTIL i = 77;
538             enter.inc := 5;
539             writeln;
540             write('enter a description for this data :  ');
541             readln(enter.descrip);
542             savefile;
543             rewrite(f, fnw, 'DAT');
544             f^ := enter;
545             put(f);
546             close(f);
547           END;
548
549
550         BEGIN
551         1:
552           REPEAT
553             BEGIN
554             prompt(top);
555             i := 10;
556             writeln(' ': i, 'A: convert 77 point ASCII data to COLOR binary format');
557             writeln(' ': i, 'B: convert 77 point COLOR binary data to ASCII format');
558             writeln;
559             writeln(' ': i, 'X: convert 512 point TRACOR data file to  X,Y format');
560             writeln(' ': i, 'T: convert 512 point TRACOR data file to 77 point COLOR binary data');
561             writeln;
562             writeln(' ': i, 'O: convert Optronics data to 77 point COLOR binary data');
563             writeln;
564             writeln(' ': i, 'S: smooth 77 point COLOR binary data');
565             writeln;
566             writeln(' ': i, 'C: calculate mean and standard deviation of 77 point COLOR');
567             writeln(' ': i, '   binary data files');
568             writeln;
569             writeln;
570             writeln;
571             REPEAT
572               BEGIN
573               write(chr(27), 'A', chr(27), 'KSelect A,B,X,T,O,S,C or Q(uit :  ');
```

```
574             readch(q);
575             END;
576           UNTIL (q IN ['A', 'B', 'X', 'T', 'S', 'C', 'O', 'Q']);
577           CASE q OF
578             'A': textbin;
579             'B': bintext;
580             'X': tracnv;
581             'T': trareduce;
582             'C': meansdev;
583             'O': Optronics;
584             'S': smooth;
585             'Q': ;
586             END;
587           END;
588         UNTIL (q = 'Q');
589         prompt(top);
590         writeln('Bye...');
591       END.
```

*** No lines with errors detected ***

APPENDIX D

COLR PROGRAM

The original color program permits a user to enter, correct, display, retrieve, and save spectral data from the Munsell Color Science Laboratory Goniospectrophotometer and other sources. The program also provides color calculations, plots on a DEC LA100 printer, and spectral manipulation with simple mathematical functions. The program consists of the following modules:

D1. COLR.PAS: the main program.

D2. COLR00.PAS: a module to automatically print the results of up to 10 spectra.

D3. COLR01.PAS: a module to manipulate Planckian radiator data.

D4. COLR02.PAS: a module to calculate the distribution temperature of a light source.

D5. COLR03.PAS: a module to calculate the correlated color temperature of a light source.

D6. COLR04.PAS: a module to select color matching functions and display a fancy menu.

D7. COLR05.PAS: a module to print a file and color coordinates.

D8. COLR06.PAS: a module to calculate the dominant wavelength and excitation purity of a sample.

D9. COLR07.PAS: a module to plot spectra on a DEC LA100 printer.

D10. COLR08.PAS: a module to calculate the color coordinates of sources and samples.

D11. COLR09.PAS: a module to correct and display spectral data.

D12. COLR10.PAS: a module to load and change program options.

D13. COLR11.PAS: a module to enter spectral data by hand.

D14. COLR12.PAS: a module to manipulate spectral data using simple mathematical functions.

```
1       PROGRAM colr;
2
3        { This code is for a main program
4          Main Program File Name : COLR.PAS
5          Other subprograms needed : COLR00 - COLR12
6          Compiling Command : pas COLR
7          Linking Command :  @COLR.LNK
8          Hardware Required :  none
9
10      Software Modifications/ Date/ Programmer
11         ------------------------------------------------------------------
12         Procedures written/modified/ 03-84/ R.M. Miller
13         Added code to trap input io errors for strings, characters, reals,
14           and integers/ 10-20-84/ R.M. Miller
15         Procedure DOCHROM corrected to pass the correct number of copies
16           to PCHROM/ 05-13-85/ R.M. Miller
17
18
19                              BRIEF PROGRAM DESCRIPTION
20
21      COLR provides a tool for the entry, display, correction, and plotting of
22      spectral data.  Routines are also available to calculate the color of
23      light sources and samples, create daylight and blackbody spectra, and
24      manipulate spectra with simple math functions.
25
26         end of informational heading }
27
28        CONST
29          startingwavelength = 380;
30          endingwavelength = 760;
31          mininc = 5;
32          numberofwavelengths = 81;
33          noofwavelengths = 77;
34          stringmax = 40;
35          top = 4;
36          plotmax = 5;
37          disponret = true;
38
39        TYPE
40          string = PACKED ARRAY [1..stringmax] OF char;
41          string101 = PACKED ARRAY [1..101] OF char;
42          string14 = PACKED ARRAY [1..14] OF char;
43          string6 = PACKED ARRAY [1..6] OF char;
44          string3 = PACKED ARRAY [1..3] OF char;
45          string2 = PACKED ARRAY [1..2] OF char;
46          data =
47            RECORD
48              descrip: string;
49              inc: integer;
50              val: ARRAY [1..numberofwavelengths] OF real;
51            END;
52          plotd = ARRAY [1..plotmax] OF data;
53          chrom = ARRAY [0..50] OF real;
```

```
  54          r30 = ARRAY [1..30] OF real;
  55
  56        VAR
  57          enter, source, sample, x10, y10, z10: data;
  58          f: FILE OF data;
  59
  60          degree: string2;
  61          defdev, defext, defsydev, defsyext: string3;
  62          chgoptkey, exitkey, key: string6;
  63          user: string;
  64          fn: string14;
  65
  66          number, len, i, is, j, k, l, waveinc: integer;
  67
  68          cx, cy, cz, tx, ty, tz, u, v, su, sv, ctemp, cctemp, c1, c2, p, r, s, scx, scy, scz, stx, sty,
  69           stz, version, lstar, astar, bstar, ustar, vstar, cuv, cab, hab, huv, uprime, suprime, vprime,
  70           svprime, excpurity, dominantwavelength: real;
  71
  72          q, qm, qt: char;
  73
  74          la100, ff, spooled, havesource, rbok, cie1931, exitok, plotsok, complementary, justsource,
  75           hueaberror, hueuverror: boolean;
  76
  77        LABEL
  78          1;
  79          {                                                                  }
  80          { ------------------------------------------------------------------ }
  81          { externally defined procedures that are external to the main program }
  82          { ------------------------------------------------------------------ }
  83          {                                                                  }
  84          {                      COLR00.PAS Procedures                       }
  85
  86
  87        PROCEDURE autoprint;
  88          EXTERNAL;
  89        {                                                                    }
  90        {                     COLR01.PAS Procedures                          }
  91
  92
  93        PROCEDURE rbratio(VAR rbok, la100, spooled: boolean;
  94                          VAR user: string);
  95          EXTERNAL;
  96
  97
  98        PROCEDURE radexc;
  99          EXTERNAL;
 100
 101
 102        PROCEDURE blackbd(VAR enter, source, sample: data;
 103                          VAR havesource: boolean);
 104          EXTERNAL;
 105        {                                                                    }
```

```
106         {                       COLR02.PAS Procedures                       }
107
108
109         PROCEDURE distributiontemp(VAR source: data);
110           EXTERNAL;
111         {                                                                   }
112         {                       COLR03.PAS Procedures                       }
113
114
115         PROCEDURE corcol(VAR u, v, cctemp: real);
116           EXTERNAL;
117         {                                                                   }
118         {                       COLR04.PAS Procedures                       }
119
120
121         PROCEDURE noctrlc;
122           EXTERNAL;
123
124
125         PROCEDURE fancy1(version: real);
126           EXTERNAL;
127
128
129         PROCEDURE clearvalues(VAR source, sample: data;
130                               VAR havesource: boolean);
131           EXTERNAL;
132
133
134         PROCEDURE getxyz(VAR x10, y10, z10: data;
135                          VAR b1: boolean;
136                          VAR degree: string2;
137                          VAR defsydev, defsyext: string3);
138           EXTERNAL;
139
140
141         PROCEDURE chgxyz(VAR x10, y10, z10: data;
142                          VAR cie1931: boolean;
143                          VAR degree: string2;
144                          VAR defsydev, defsyext: string3);
145           EXTERNAL;
146         {                                                                   }
147         {                       COLR05.PAS Procedures                       }
148
149
150         PROCEDURE printfile(VAR s1: data;
151                             spooled, la100, ff: boolean;
152                             VAR user: string);
153           EXTERNAL;
154
155
156         PROCEDURE pchrom(VAR source, sample: data;
157                          VAR cx, cy, cz, tx, ty, tz, u, v, lstar, astar, bstar, cab, hab, ustar, vstar,
```

```
158                          cuv, huv, domwav, purity: real;
159                        VAR degree: string2;
160                        VAR user: string;
161                        VAR la100, ff, spooled, complementary, domwaverror, hueaberror,
162                        hueuverror: boolean;
163                        number: integer);
164          EXTERNAL;
165
166
167        PROCEDURE doprint(VAR enter, source, sample: data;
168                          VAR la100, ff, spooled: boolean;
169                          VAR user: string);
170          EXTERNAL;
171        {                                                                  }
172        {                       COLR06.PAS Procedures                      }
173
174
175        PROCEDURE domwav(VAR x, y, xilluminant, yilluminant, dominantwavelength, excpurity: real;
176                         VAR complementary, domwaverror: boolean;
177                         defsydev: string3);
178          EXTERNAL;
179        {                                                                  }
180        {                       COLR07.PAS Procedures                      }
181
182
183        PROCEDURE plot(number: integer;
184                       VAR s: plotd;
185                       la100, ff, spooled: boolean;
186                       VAR user, yaxislabel: string);
187          EXTERNAL;
188
189
190        PROCEDURE doplots(VAR enter, source, sample: data;
191                          VAR la100, ff, spooled: boolean;
192                          VAR user: string);
193          EXTERNAL;
194        {                                                                  }
195        {                       COLR08.PAS Procedures                      }
196
197
198        PROCEDURE ciecalc(VAR source, sample: data;
199                          VAR havesource, la100, spooled: boolean;
200                          VAR user: string);
201          EXTERNAL;
202
203
204        PROCEDURE hueangle(VAR x, y, h: real;
205                           VAR error: boolean);
206          EXTERNAL;
207
208
209        PROCEDURE cieluv(VAR tx, ty, tz, stx, sty, stz, uprime, vprime, suprime, svprime, lstar, ustar,
```

```
210                          vstar, cuv, huv: real;
211                         VAR hueangleerroruv: boolean);
212         EXTERNAL;
213
214
215       PROCEDURE cielab(VAR tx, ty, tz, stx, sty, stz, lstar, astar, bstar, cab, hab: real;
216                         VAR hueangleerrorab: boolean);
217         EXTERNAL;
218       {                                                                    }
219       {                       COLR09.PAS Procedures                        }
220
221
222       PROCEDURE correctfile(VAR enter: data);
223         EXTERNAL;
224
225
226       PROCEDURE displayfile(VAR s: data;
227                             page: integer);
228         EXTERNAL;
229
230
231       PROCEDURE cdata(VAR enter, source, sample: data;
232                       VAR havesource: boolean;
233                       newfile: boolean);
234         EXTERNAL;
235       {                                                                    }
236       {                       COLR10.PAS Procedures                        }
237
238
239       PROCEDURE chgopt(VAR exitok, la100, ff, plotsok, rbok, spooled: boolean;
240                         VAR chgoptkey: string6;
241                         VAR defdev, defsydev, defext, defsyext: string3);
242         EXTERNAL;
243
244
245       PROCEDURE getopt(VAR exitok, la100, ff, plotsok, rbok, spooled: boolean;
246                         VAR chgoptkey, exitkey: string6;
247                         VAR version: real;
248                         VAR defdev, defsydev, defext, defsyext: string3);
249         EXTERNAL;
250       {                                                                    }
251       {                       COLR11.PAS Procedures                        }
252
253
254       PROCEDURE enterfile(VAR enter: data);
255         EXTERNAL;
256       {                                                                    }
257       {                       COLR12.PAS Procedures                        }
258
259
260       PROCEDURE multfile(VAR s1, s2, s3: data;
261                           d: integer);
```

```
262         EXTERNAL;
263
264
265       PROCEDURE mfile(VAR enter, source, sample: data;
266                       VAR havesource: boolean;
267                       defsydev, defsyext: string3);
268         EXTERNAL;
269
270
271       FUNCTION density(VAR r1: real): real;
272         EXTERNAL;
273       {                                                                }
274       {               Pascal-2 V2.1D External Procedures               }
275
276
277       PROCEDURE timestamp(VAR day, month, year, hour, minute, second: integer);
278         EXTERNAL;
279
280
281       FUNCTION readsn: char;
282         EXTERNAL;
283       {                                                                }
284       { ---------------------------------------------------------------- }
285       { externally defined procedures that are part of main program      }
286       { ---------------------------------------------------------------- }
287       { NOTE: CLRPRO.PAS also includes the type and constant declarations  }
288       {       for the main program COLR.PAS                            }
289       {                                                                }
290       {                           CLRPRO.PAS                           }
291
292
293       PROCEDURE chromaticity(VAR s1, s2: data;
294                              VAR cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
295                              sv: real);
296         EXTERNAL;
297
298
299       PROCEDURE concat(VAR filename: string14;
300                        dev: string3;
301                        filnam: string6;
302                        ext: string3);
303         EXTERNAL;
304
305
306       PROCEDURE dochrom(printonly: boolean;
307                         number: integer);
308         EXTERNAL;
309
310
311       PROCEDURE fillsource(VAR source: data;
312                            VAR cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
313                            sv: real);
```

```
314           EXTERNAL;
315
316
317         PROCEDURE getfilename(VAR temp: string14);
318           EXTERNAL;
319
320
321         PROCEDURE gotoxy(x2, x1: integer);
322           EXTERNAL;
323
324
325         PROCEDURE header(VAR out: text;
326                          VAR la100, spooled: boolean;
327                          VAR user: string);
328           EXTERNAL;
329
330
331         PROCEDURE jump(x2: integer);
332           EXTERNAL;
333
334
335         FUNCTION plank(w1: integer;
336                        t1: real): real;
337           EXTERNAL;
338
339
340         PROCEDURE prompt(x1: integer);
341           EXTERNAL;
342
343
344         FUNCTION raise(r1, r2: real): real;
345           EXTERNAL;
346
347
348         PROCEDURE readch(VAR q: char);
349           EXTERNAL;
350
351
352         PROCEDURE readre(s: PACKED ARRAY [lower..upper: integer] OF char;
353                          VAR r: real);
354           EXTERNAL;
355
356
357         PROCEDURE readin(s: PACKED ARRAY [lower..upper: integer] OF char;
358                          VAR r: integer);
359           EXTERNAL;
360
361
362         PROCEDURE readst(VAR s: PACKED ARRAY [lower..upper: integer] OF char);
363           EXTERNAL;
364
365
```

```
366         PROCEDURE retrievefile(VAR name: string14;
367                                VAR s1: data);
368           EXTERNAL;
369
370
371         PROCEDURE return(x2: integer);
372           EXTERNAL;
373
374
375         PROCEDURE rstring(VAR s: string);
376           EXTERNAL;
377
378
379         PROCEDURE savefile(enter: data);
380           EXTERNAL;
381
382
383         FUNCTION validwave(x1: integer): boolean;
384           EXTERNAL;
385
386
387         PROCEDURE Wonval(VAR v1: real);
388           EXTERNAL;
389
390
391         PROCEDURE Wval(w1: integer;
392                        VAR v1: real);
393           EXTERNAL;
394
395
396         PROCEDURE gotoxy;
397
398           BEGIN
399             write(chr(27), 'Y', chr(x1 + 32), chr(x2 + 32));
400           END;
401
402
403         PROCEDURE readch;
404
405           VAR
406             i: integer;
407
408           BEGIN
409             IF ioerror(input) THEN
410               BEGIN
411               IF iostatus(input) = 17 THEN
412                 BEGIN
413                 reset(input);
414                 noioerror(input);
415                 END;
416               END;
417             q := readsn;
```

```
418           i := ord(q);
419           IF i < 0 THEN i := i + 128;
420           IF i = 13 THEN q := readsn;
421           IF (i = 10) OR (i = 27) OR (i = 13) THEN q := ' ';
422           writeln(q);
423           IF q IN ['a'..'z'] THEN q := chr(ord(q) - 32);
424           IF q = '?' THEN GOTO 1;
425         END;
426
427
428       PROCEDURE readre;
429        {  (s: packed array[lower..upper:integer] of char; var r: real);  }
430
431         VAR
432           error: boolean;
433           i: integer;
434           esc: char;
435
436         BEGIN
437           esc := chr(27);
438           writeln;
439           REPEAT
440             write(esc, 'A', esc, 'K');
441             FOR i := lower TO upper DO write(s[i]);
442             error := false;
443             IF NOT eoln THEN read(r)
444             ELSE error := true;
445             error := error OR ioerror(input);
446             readln;
447             error := error OR ioerror(input);
448             IF error THEN
449               BEGIN
450               write(esc, 'A', esc, 'K');
451               write('Real Input Error !!!  Press ? to abort -or- any other key to try again :  ');
452               readch(q);
453               END;
454           UNTIL NOT error;
455         END;
456
457
458       PROCEDURE readin;
459        {  (s: packed array[lower..upper:integer] of char; var r: integer);  }
460
461         VAR
462           error: boolean;
463           esc: char;
464           q: char;
465           i: integer;
466
467         BEGIN
468           esc := chr(27);
469           writeln;
```

```
470             REPEAT
471               write(esc, 'A', esc, 'K');
472               FOR i := lower TO upper DO write(s[i]);
473               error := false;
474               IF NOT eoln THEN read(r)
475               ELSE error := true;
476               error := error OR ioerror(input);
477               readln;
478               error := error OR ioerror(input);
479               IF error THEN
480                 BEGIN
481                 write(esc, 'A', esc, 'K');
482                 write('Integer Input Error !!!  Press ? to abort -or- any other key to try again :  ');
483                 readch(q);
484                 END;
485             UNTIL NOT error;
486           END;
487
488
489         PROCEDURE readst;
490          {   (var s: packed array[lower..upper:integer] of char);   }
491
492           VAR
493             error: boolean;
494             i: integer;
495             temp: string;
496
497           BEGIN
498             error := false;
499             read(temp);
500             error := error OR ioerror(input);
501             readln;
502             error := error OR ioerror(input);
503             IF temp[1] = '?' THEN GOTO 1;
504             IF error THEN
505               FOR i := lower TO upper DO s[i] := ' '
506             ELSE
507               BEGIN
508               IF (upper - lower + 1) > stringmax THEN
509                 BEGIN
510                 FOR i := lower TO (lower + stringmax - 1) DO s[i] := temp[i - lower + 1];
511                 FOR i := (lower + stringmax) TO upper DO s[i] := ' ';
512                 END
513               ELSE FOR i := lower TO upper DO s[i] := temp[i - lower + 1]
514               END;
515           END;
516
517
518         PROCEDURE return;
519
520           VAR
521             i3: integer;
```

```
522             qr: char;
523
524           BEGIN
525             FOR i3 := 0 TO x2 DO writeln;
526             write(' ': 25, 'press any key to continue');
527             readch(qr);
528           END;
529
530
531         PROCEDURE jump;
532
533           VAR
534             i3: integer;
535
536           BEGIN
537             FOR i3 := 1 TO x2 DO writeln;
538           END;
539
540
541         PROCEDURE prompt;
542
543           BEGIN
544             writeln(chr(27), 'H', chr(27), 'J', 'Munsell Color Software Version ': 49, version: 3: 1);
545             jump(x1);
546           END;
547
548
549         PROCEDURE rstring;
550
551           BEGIN
552             readst(s);
553           END;
554
555
556         PROCEDURE concat;
557
558           VAR
559             i: integer;
560
561           BEGIN
562             FOR i := 1 TO 3 DO
563               BEGIN
564               filename[i] := dev[i];
565               filename[i + 11] := ext[i]
566               END;
567             FOR i := 1 TO 6 DO filename[i + 4] := filnam[i];
568             filename[4] := ':';
569             filename[11] := '.';
570           END;
571
572
573         PROCEDURE Wonval;
```

```
574
575           BEGIN
576             IF (((v1 < 1000.0) AND (v1 > 0.001)) OR (v1 = 0.0)) THEN write(v1: 10: 4)
577             ELSE write(v1: 10: - 3);
578           END;
579
580
581         PROCEDURE Wval;
582
583           BEGIN
584             write(w1: 3, '>');
585             Wonval(v1);
586           END;
587
588
589         PROCEDURE printtwo(VAR out: text;
590                            N: integer);
591
592           BEGIN
593             write(out, N DIV 10: 1, N MOD 10: 1);
594           ENO;
595
596
597         PROCEOURE header;
598
599           VAR
600             qt: char;
601             day, month, year, hour, minute, second: integer;
602
603           BEGIN
604             IF NOT spooled THEN
605               BEGIN
606               prompt(top);
607               writeln('Make sure that the PRINTER is turned on and that ');
608               writeln('  the LOCAL button is not depressed...');
609               writeln;
610               writeln;
611               write(' ': 24, 'press any key to print the file ');
612               readch(qt);
613               END;
614             writeln(out);
615             IF la100 THEN write(out, chr(27), '[5w');
616             writeln(out, 'RIT Munsell Color Science Laboratory  Software Version ', version: 3: 1);
617             IF la100 THEN write(out, chr(27), '[7w');
618             timestamp(day, month, year, hour, minute, second);
619             printtwo(out, day);
620             CASE month OF
621               1: write(out, '-Jan-');
622               2: write(out, '-Feb-');
623               3: write(out, '-Mar-');
624               4: write(out, '-Apr-');
625               5: write(out, '-May-');
```

```
626             6: write(out, '-Jun-');
627             7: write(out, '-Jul-');
628             8: write(out, '-Aug-');
629             9: write(out, '-Sep-');
630             10: write(out, '-Oct-');
631             11: write(out, '-Nov-');
632             12: write(out, '-Dec-');
633             OTHERWISE write(out, '-XXX-');
634             END;
635           write(out, year: 4, ' ');
636           printtwo(out, hour);
637           write(out, ':');
638           printtwo(out, minute);
639           write(out, ':');
640           printtwo(out, second);
641           writeln(out, ' ': 2, 'User ID :  ', user);
642           IF la100 THEN write(out, chr(27), '[0w');
643           writeln(out);
644         END;
645
646
647       PROCEDURE getfilename;
648
649         VAR
650           nc, np, i, j, colon, period, endpos: integer;
651           dev, ext: string3;
652           fn: string6;
653           error: boolean;
654           ch: char;
655
656         BEGIN
657           REPEAT
658             BEGIN
659             FOR i := 1 TO 14 DO temp[i] := chr(0);
660             readst(temp);
661             IF temp[1] = '?' THEN GOTO 1;
662             FOR i := 1 TO 14 DO IF temp[i] = ' ' THEN temp[i] := chr(0);
663             FOR i := 1 TO 14 DO
664               BEGIN
665               IF temp[i] = chr(0) THEN
666                 BEGIN
667                 FOR j := i TO 13 DO temp[j] := temp[j + 1];
668                 temp[14] := chr(0);
669                 END;
670               END;
671             nc := 0;
672             np := 0;
673             endpos := 15;
674             colon := 0;
675             FOR i := 1 TO 14 DO
676               BEGIN
677               IF temp[i] = ':' THEN
```

```
678                 BEGIN
679                 nc := nc + 1;
680                 colon := i;
681                 END;
682               IF temp[i] = '.' THEN
683                 BEGIN
684                 np := np + 1;
685                 period := i;
686                 END;
687               IF temp[15 - i] = chr(0) THEN endpos := 15 - i;
688               END;
689             IF np = 0 THEN period := endpos;
690             error := false;
691             FOR i := 1 TO (endpos - 1) DO
692               BEGIN
693               ch := temp[i];
694               IF NOT (ch IN ['a'..'z', 'A'..'Z', '0'..'9', ':', '.']) THEN error := true;
695               END;
696             IF (period < colon) OR (colon = 1) OR (period = 1) OR (colon > 4) OR
697                ((period - colon) > 7) OR ((endpos - period) > 4) OR (period = colon + 1) THEN
698               error := true;
699             IF error THEN
700               BEGIN
701               prompt(1);
702               writeln('          Filename Entered by User :  ', temp);
703               writeln('WARNING!!! This file name is not in the proper format');
704               writeln('           The filename should be six characters long');
705               writeln('           with no intermediate spaces. The only legal');
706               writeln('           characters are letters and numbers. If you');
707               writeln('           wish a file can also be specified as :');
708               writeln('                                  DEV:FILNAM.EXT');
709               writeln('Enter ? to return to main menu');
710               write('  -or-  enter filename again (?,filename) :  ');
711               END
712             ELSE
713               BEGIN
714               FOR i := 1 TO 3 DO
715                 BEGIN
716                 ext[i] := chr(0);
717                 END;
718               dev := ext;
719               FOR i := 1 TO 6 DO fn[i] := chr(0);
720               IF colon > 1 THEN FOR i := 1 TO (colon - 1) DO dev[i] := temp[i]
721               ELSE dev := defdev;
722               FOR i := (colon + 1) TO (period - 1) DO fn[i - colon] := temp[i];
723               IF endpos > period THEN
724                 FOR i := (period + 1) TO (endpos - 1) DO ext[i - period] := temp[i]
725               ELSE ext := defext;
726               END;
727             END;
728           UNTIL NOT error;
729           concat(temp, dev, fn, ext);
```

```
730         END;
731
732
733       PROCEDURE retrievefile;
734
735         VAR
736           i: integer;
737
738         BEGIN
739           REPEAT
740             BEGIN
741             reset(f, name, , len);
742             IF len = - 1 THEN
743               BEGIN
744               prompt(1);
745               writeln(' ': 10, name, ' is not a valid file name');
746               jump(1);
747               write(' ': 10, 'enter ? to return to menu -or-');
748               writeln;
749               write(' ': 15, 'enter valid file name (xxxxxx) :  ');
750               getfilename(name);
751               END;
752             END;
753           UNTIL len <> - 1;
754           get(f);
755           s1 := f^;
756           FOR i := (noofwavelengths + 1) TO numberofwavelengths DO s1.val[i] := 0.0;
757           close(f);
758         END;
759
760
761       PROCEDURE getfile;
762
763         VAR
764           i: integer;
765
766         BEGIN
767           prompt(top);
768           write('enter file name to retrieve (XXXXXX) :  ');
769           getfilename(fn);
770           retrievefile(fn, enter);
771           sample := enter;
772           IF disponret THEN
773             BEGIN
774             displayfile(enter, 0);
775             gotoxy(0, 11);
776             write(' ': 25, 'press any key to continue');
777             readch(qt);
778             displayfile(enter, 1);
779             gotoxy(10, 11);
780             write(' ': 25, 'press any key to continue');
781             readch(qt);
```

```
782             END;
783         END;
784
785
786       PROCEDURE savefile;
787
788         VAR
789           i, len: integer;
790           f: FILE OF data;
791           fn: string14;
792           q: char;
793
794         BEGIN
795           prompt(top);
796           write('enter filename for data (xxxxxx) :  ');
797           getfilename(fn);
798           REPEAT
799             BEGIN
800             reset(f, fn, , len);
801             IF len <> - 1 THEN
802               BEGIN
803               close(f);
804               prompt(1);
805               writeln('WARNING!!!! file name : ', fn, '  already exists');
806               jump(2);
807               write(' D(elete this file and use same name -OR- E(nter new name (D,E) [E]:  ');
808               readch(q);
809               IF q <> 'D' THEN
810                 BEGIN
811                 jump(1);
812                 write('enter new file name (xxxxxx) :  ');
813                 getfilename(fn);
814                 END;
815               END;
816             END;
817           UNTIL (q = 'D') OR (len = - 1);
818           len := 1;
819           rewrite(f, fn, , len);
820           f^ := enter;
821           put(f);
822           close(f);
823         END;
824
825
826       FUNCTION validwave;
827
828         VAR
829           j: integer;
830
831         BEGIN
832           j := round(10 * (x1 / 10.0 - trunc(x1 / 10.0)));
833           IF (x1 >= startingwavelength) AND (x1 <= endingwavelength) AND ((j = 5) OR (j = 0)) THEN
```

```
834             validwave := true
835           ELSE validwave := false;
836         END;
837
838
839       FUNCTION raise;
840        {  raises r1 to the r2 power                                    }
841
842         BEGIN
843           raise := exp(r2 * ln(r1));
844         END;
845
846
847       FUNCTION plank;
848
849         VAR
850           wt: real;
851
852         BEGIN
853           wt := w1 * 1E-9;
854           plank := 3.741E-16 * raise(wt, - 5.0) / (exp(1.4388E-2 / t1 / wt) - 1);
855         END;
856
857
858       FUNCTION sumfile(VAR s1, s2, s3: data): real;
859
860         VAR
861           sum: real;
862           i1, minc: integer;
863
864         BEGIN
865           sum := 0;
866           FOR i1 := 1 TO noofwavelengths DO sum := sum + s1.val[i1] * s2.val[i1] * s3.val[i1];
867           minc := D;
868           IF s1.inc > minc THEN minc := s1.inc;
869           IF s2.inc > minc THEN minc := s2.inc;
870           IF s3.inc > minc THEN minc := s3.inc;
871           sumfile := sum * minc;
872         END;
873
874
875       PROCEDURE tristimulus(VAR s1, s2: data;
876                             VAR tx, ty, tz: real;
877                             VAR justsource: boolean);
878
879         VAR
880           ones: data;
881           i: integer;
882           tys: real;
883
884         BEGIN
885           FOR i := 1 TO noofwavelengths DO ones.val[i] := 1.0;
```

```
886             ones.inc := 5;
887             tys := sumfile(s1, ones, y10);
888             IF justsource THEN
889               BEGIN
890               tx := sumfile(s1, ones, x10) / tys;
891               ty := sumfile(s1, ones, y10) / tys;
892               tz := sumfile(s1, ones, z10) / tys;
893               END
894             ELSE
895               BEGIN
896               tx := sumfile(s1, s2, x10) / tys;
897               ty := sumfile(s1, s2, y10) / tys;
898               tz := sumfile(s1, s2, z10) / tys;
899               END;
900           END;
901
902
903         PROCEDURE chromaticity;
904
905           VAR
906             justsource: boolean;
907
908           BEGIN
909             justsource := true;
910             tristimulus(s1, s2, stx, sty, stz, justsource);
911             justsource := false;
912             tristimulus(s1, s2, tx, ty, tz, justsource);
913             cx := tx / (tx + ty + tz);
914             cy := ty / (tx + ty + tz);
915             cz := tz / (tx + ty + tz);
916             u := 4 * cx / ( - 2 * cx + 12 * cy + 3);
917             v := 6 * cy / ( - 2 * cx + 12 * cy + 3);
918             scx := stx / (stx + sty + stz);
919             scy := sty / (stx + sty + stz);
920             scz := stz / (stx + sty + stz);
921             su := 4 * scx / ( - 2 * scx + 12 * scy + 3);
922             sv := 6 * scy / ( - 2 * scx + 12 * scy + 3);
923           END;
924
925
926         PROCEDURE fillsource;
927
928           VAR
929             q: char;
930             fn: string14;
931             i: integer;
932             ones: data;
933
934           BEGIN
935             FOR i := 1 TO 77 DO ones.val[i] := 1.0;
936             ones.inc := 5;
937             REPEAT
```

```
938             BEGIN
939             prompt(2);
940             writeln('Illuminants currently available');
941             writeln;
942             writeln(' ': 10, 'D:  CIE Illuminant D65');
943             writeln(' ': 10, 'A:  CIE Illuminant A');
944             writeln(' ': 10, 'C:  CIE Illuminant C');
945             writeln;
946             write('Select D, A, C, or R(etrieve another file (D,A,C,R) :  ');
947             readch(q);
948             END;
949           UNTIL q IN ['D', 'A', 'C', 'R'];
950           CASE q OF
951             'D': concat(fn, defsydev, 'CIED65', defext);
952             'A': concat(fn, defsydev, 'CIEA  ', defext);
953             'C': concat(fn, defsydev, 'CIEC  ', defext);
954             'R':
955               BEGIN
956               prompt(2);
957               write(' ': 15, 'enter a source filename (xxxxxx) : ');
958               getfilename(fn);
959               END;
960             END;
961           retrievefile(fn, source);
962           havesource := true;
963           chromaticity(source, ones, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
964                        sv);
965         END;
966
967
968       PROCEDURE getsource;
969
970         VAR
971           q: char;
972
973         BEGIN
974           prompt(1);
975           writeln('WARNING...  you have attempted an operation that requires a source');
976           jump(2);
977           writeln('      a source has not been entered or retrieved');
978           jump(1);
979           writeln(' ': 10, 'enter ? to return to menu');
980           writeln;
981           write(' ': 15, '-or- : press any other key to retrieve a source :  ');
982           readch(q);
983           fillsource(source, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su, sv);
984         END;
985
986
987       PROCEDURE cortemp(VAR s1: data);
988
989         VAR
```

```
990           i1: integer;
991           q: char;
992
993         BEGIN
994           IF NOT havesource THEN getsource;
995           prompt(top);
996           chromaticity(s1, sample, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su, sv);
997           corcol(su, sv, cctemp);
998           writeln('Correlated Color Temperature :   ', cctemp: 10: 3, ' K');
999           jump(2);
1000          writeln('source description :  ', source.descrip);
1001          return(2);
1002        END;
1003
1004
1005      PROCEOURE dispdom(VAR cx, cy, scx, scy, dominantwavelength, excpurity: real;
1006                        VAR complementary, domwaverror: boolean;
1007                        printonly: boolean);
1008
1009        VAR
1010          q: char;
1011          i: integer;
1012
1013        BEGIN
1014          domwav(cx, cy, scx, scy, dominantwavelength, excpurity, complementary, domwaverror, defsydev);
1015          IF NOT printonly THEN
1016            BEGIN
1017            prompt(1);
1018            writeln('Chromaticity coordinates :');
1019            writeln('       Source       x : ', scx: 6: 4, '     y : ', scy: 6: 4);
1020            writeln('       Sample       x : ', cx: 6: 4, '     y : ', cy: 6: 4);
1021            writeln;
1022            IF NOT domwaverror THEN
1023              BEGIN
1024              write(' ': 4);
1025              IF complementary THEN write('Complementary')
1026              ELSE write('Oominant');
1027              write(' Wavelength :  ', dominantwavelength: 6: 2, ' nm');
1028              IF complementary THEN writeln(' C')
1029              ELSE writeln;
1030              writeln;
1031              writeln(' ': 4, 'Excitation Purity : ', (excpurity * 100): 6: 2, ' %');
1032              writeln;
1033              IF complementary THEN
1034                BEGIN
1035                writeln('** ** NOTE:  This is a COMPLEMENTARY  wavelength  ** **');
1036                FOR i := 1 TO 20 OO write(chr(7));
1037                END;
1038              END
1039            ELSE
1040              BEGIN
1041              writeln;
```

```
1042                writeln('Dominant wavelength cannot be computed');
1043                writeln('  Invalid Conditions -- See the Laboratory Instructor');
1044                END;
1045              END;
1046            IF (domwaverror AND printonly) THEN
1047              writeln('WARNING!!! Dominant Wavelength Calculation Error Occurred');
1048          END;
1049
1050
1051        PROCEDURE dochrom;
1052
1053          VAR
1054            q: char;
1055            domwaverror, noscreenio: boolean;
1056
1057          BEGIN
1058            IF NOT printonly THEN
1059              BEGIN
1060              prompt(top);
1061              write('Should output be directed to the P(rinter or T(erminal (P,T) [P] :  ');
1062              readch(q);
1063              IF q = 'T' THEN noscreenio := false
1064              ELSE noscreenio := true;
1065              END
1066            ELSE noscreenio := true;
1067            IF NOT havesource THEN getsource;
1068            chromaticity(source, sample, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
1069                         sv);
1070            cielab(tx, ty, tz, stx, sty, stz, lstar, astar, bstar, cab, hab, hueaberror);
1071            cieluv(tx, ty, tz, stx, sty, stz, uprime, vprime, suprime, svprime, lstar, ustar, vstar, cuv,
1072                   huv, hueuverror);
1073            IF NOT noscreenio THEN
1074              BEGIN
1075              prompt(1);
1076              writeln('Tristimulus Values :': 30, 'X :  ', tx: 9: 5);
1077              writeln(' ': 8, degree: 2, ' ': 20, 'Y :  ', ty: 9: 5);
1078              writeln(' ': 30, 'Z :  ', tz: 9: 5);
1079              writeln('Chromaticity Coordinates :': 30, 'x :  ', cx: 9: 7, ' ': 10, 'u :  ', u: 9: 7);
1080              writeln(' ': 30, 'y :  ', cy: 9: 7, ' ': 10, 'v :  ', v: 9: 7);
1081              writeln(' ': 30, 'z :  ', cz: 9: 7);
1082              writeln('Sample Description :  ', sample.descrip);
1083              writeln('Source Description :  ', source.descrip);
1084              return(0);
1085              prompt(1);
1086              writeln('Chromaticity Coordinates :', ' ': 6, 'u'' :  ', u: 9: 7, ' ': 10, 'v'' :  ',
1087                      (v * 1.5): 9: 7);
1088              writeln('CIELAB Coordinates');
1089              writeln('L* :  ': 10, lstar: 9: 5, ' ': 10, 'a* :  ': 10, astar: 9: 5, ' ': 10, 'b* :  ':
1090                      10, bstar: 9: 5);
1091              write('C*ab :  ': 10, cab: 9: 5, ' ': 10);
1092              IF NOT hueaberror THEN write('Hab :  ': 10, hab: 9: 5, ' degrees');
1093              writeln;
```

```
1094              writeln('CIELUV Coordinates');
1095              writeln('L* :  ': 10, lstar: 9: 5, ' ': 10, 'u* :  ': 10, ustar: 9: 5, ' ': 10, 'v* :  ':
1096                    10, vstar: 9: 5);
1097              write('C*uv :  ': 10, cuv: 9: 5, ' ': 10);
1098              IF NOT hueuverror THEN write('Huv :  ': 10, huv: 9: 5, ' degrees');
1099              return(2);
1100              END;
1101            IF cie1931 THEN
1102              BEGIN
1103              dispdom(cx, cy, scx, scy, dominantwavelength, excpurity, complementary, domwaverror,
1104                      noscreenio);
1105              END
1106            ELSE IF NOT noscreenio THEN
1107              BEGIN
1108              writeln('dominant wavelength calculations are only available for data');
1109              writeln('calculated using the 1931 observer');
1110              return(2);
1111              domwaverror := true;
1112              END
1113            ELSE domwaverror := true;
1114            IF NOT noscreenio THEN
1115              BEGIN
1116              writeln;
1117              writeln('the coordinate information can be printed on the system printer');
1118              write(' P(rint these values -or- any other key to continue (P) :  ');
1119              readch(q);
1120              END;
1121            IF (q = 'P') OR (noscreenio) THEN
1122              BEGIN
1123              pchrom(source, sample, cx, cy, cz, tx, ty, tz, u, v, lstar, astar, bstar, cab, hab, ustar,
1124                     vstar, cuv, huv, dominantwavelength, excpurity, degree, user, la100, ff, spooled,
1125                     complementary, domwaverror, hueaberror, hueuverror, number);
1126              END;
1127          END;
1128
1129
1130        PROCEDURE getuser(VAR user: string);
1131
1132          BEGIN
1133            prompt(4);
1134            write('Enter User ID :');
1135            gotoxy(0, 8);
1136            writeln('This User ID will be printed on all computer output');
1137            gotoxy(16, 5);
1138            readst(user);
1139          END;
1140
1141
1142        PROCEDURE menu;
1143
1144          VAR
1145            i: integer;
```

```
1146            mainmenu, domwaverror: boolean;
1147
1148          CONST
1149            i1 = 10;
1150
1151          BEGIN
1152            mainmenu := true;
1153            REPEAT
1154              BEGIN
1155              prompt(1);
1156              IF mainmenu THEN
1157                BEGIN
1158                writeln('E:  enter data file                    T:  calculate D.T. of source');
1159                writeln('R:  retrieve data file                 U:  calculate C.C.T. of source');
1160                writeln('C:  calculate chromaticities           D:  display source or sample');
1161                writeln('L:  CIELab/CIELuv  Color Differences   O:  correct data file');
1162                writeln('B:  dominant wavelength/ exc. purity   P:  print/plot data file');
1163                writeln('X:  exit from this program             S:  change to supplemental menu');
1164                END
1165              ELSE
1166                BEGIN
1167                writeln('V:  calculate R/B ratio values         G:  spec. radiant excitance calc.');
1168                writeln('M:  multiply data files                A:  select 2 or 10 degree observer');
1169                writeln('K:  create blackbody source            H:  change program options');
1170                writeln('Z:  zero source and sample arrays      J:  autoprint of calculations');
1171                writeln('N:  change user name                   S:  change to main menu');
1172                END;
1173              writeln;
1174              writeln;
1175              writeln;
1176              REPEAT
1177                write(chr(27), 'A', chr(27), 'K', ' ': 10, 'select one of the above :  ');
1178                readch(qm);
1179              UNTIL qm IN
1180                    ['A', 'B', 'C', 'D', 'E', 'G', 'H', 'J', 'L', 'K', 'M', 'N', 'O', 'P', 'R', 'S', 'T',
1181                     'U', 'V', 'X', 'Z'];
1182              CASE qm OF
1183                'E':
1184                  BEGIN
1185                  enterfile(enter);
1186                  cdata(enter, source, sample, havesource, true);
1187                  END;
1188                'O': cdata(enter, source, sample, havesource, false);
1189                'M': mfile(enter, source, sample, havesource, defsydev, defsyext);
1190                'N': getuser(user);
1191                'J': autoprint;
1192                'C':
1193                  BEGIN
1194                  IF NOT havesource THEN getsource;
1195                  dochrom(false, 0);
1196                  END;
1197                'L': ciecalc(source, sample, havesource, la100, spooled, user);
```

```
1198                'B':
1199                  BEGIN
1200                  prompt(top);
1201                  writeln('Dominant Wavelength and Excitation Purity Calculations');
1202                  writeln('  are only available for chromaticity points calculated');
1203                  writeln('  using the 1931 CIE Color Matching Functions');
1204                  writeln;
1205                  writeln;
1206                  write('Was your data calculated using the 1931 CMF ? (Y,N) [Y] :  ');
1207                  readch(q);
1208                  IF q <> 'N' THEN
1209                    BEGIN
1210                    prompt(0);
1211                    writeln('Do you wish to R(etrieve the a source''s chromaticity coordinates');
1212                    write('  from a disk file -or- enter then by H(and (R,H) [R] :  ');
1213                    readch(q);
1214                    IF q = 'H' THEN
1215                      BEGIN
1216                      writeln('For the Source :');
1217                      readre('  enter chromaticity coordinate x :  ', scx);
1218                      readre('  enter chromaticity coordinate y :  ', scy);
1219                      END
1220                    ELSE
1221                      fillsource(source, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
1222                                 sv);
1223                    REPEAT
1224                      BEGIN
1225                      writeln('For the Sample :');
1226                      readre('  enter chromaticity coordinate x :  ', cx);
1227                      readre('  enter chromaticity coordinate y :  ', cy);
1228                      dispdom(cx, cy, scx, scy, dominantwavelength, excpurity, complementary, domwaverror,
1229                              false);
1230                      writeln;
1231                      write('Q(uit -or- any other key to enter another sample point (Q) :  ');
1232                      readch(q);
1233                      END;
1234                    UNTIL q = 'Q';
1235                    END;
1236                  END;
1237                'R':
1238                  BEGIN
1239                  prompt(top);
1240                  write('Retrieve a S(ource  -or-  sA(mple (S,A) [A] :  ');
1241                  q := 'A';
1242                  readch(q);
1243                  IF q <> 'S' THEN getfile
1244                  ELSE
1245                    fillsource(source, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su,
1246                               sv);
1247                  END;
1248                'K': blackbd(enter, source, sample, havesource);
1249                'T':
```

```
1250                BEGIN
1251                IF NOT havesource THEN getsource;
1252                distributiontemp(source);
1253                END;
1254               'U':
1255                BEGIN
1256                prompt(top);
1257                writeln('calculate Correlated Color of the S(ource in memory');
1258                write('  - or -  enter C(oordinates of source by hand ? (S,C) [S] :  ');
1259                readch(q);
1260                IF q <> 'C' THEN cortemp(source)
1261                ELSE
1262                  BEGIN
1263                  prompt(top);
1264                  writeln('Current Coordinates Available for CCT calculation');
1265                  writeln;
1266                  writeln('     1:     1931 x,y   chromaticity coordinates');
1267                  writeln('     2:     1960 u,v   chromaticity coordinates');
1268                  writeln('     3:     1976 u'',v'' chromaticity coordinates');
1269                  writeln;
1270                  write('select 1 - 3 :  [1] :  ');
1271                  readch(qt);
1272                  REPEAT
1273                    prompt(top);
1274                    writeln('Correlated Color Temperature Calculator');
1275                    jump(2);
1276                    CASE qt OF
1277                      '2':
1278                        BEGIN
1279                        readre('enter coordinate u :  ', u);
1280                        readre('enter coordinate v :  ', v);
1281                        END;
1282                      '3':
1283                        BEGIN
1284                        readre('enter coordinate u'' :  ', uprime);
1285                        readre('enter coordinate v'' :  ', vprime);
1286                        u := uprime;
1287                        v := vprime / 1.5;
1288                        END;
1289                      OTHERWISE
1290                        BEGIN
1291                        readre('enter coordinate x :  ', cx);
1292                        readre('enter coordinate y :  ', cy);
1293                        u := 4 * cx / ( - 2 * cx + 12 * cy + 3);
1294                        v := 6 * cy / ( - 2 * cx + 12 * cy + 3);
1295                        END;
1296                    END;
1297                    jump(2);
1298                    corcol(u, v, cctemp);
1299                    writeln('Correlated Color Temperature :   ', cctemp: 10: 3, ' K');
13D0                    jump(2);
13D1                    write('press RETURN to do another calculation or  Q(uit (Q) :  ');
```

```
1302                    readch(q);
1303                  UNTIL q = 'Q';
1304                  END;
1305                END;
1306              'A': chgxyz(x10, y10, z10, cie1931, degree, defsydev, defsyext);
1307              'H':
1308                chgopt(exitok, la100, ff, plotsok, rbok, spooled, chgoptkey, defdev, defsydev, defext,
1309                       defsyext);
1310              'D':
1311                BEGIN
1312                prompt(top);
1313                write('display S(ource or sA(mple (S,A) [A] :  ');
1314                readch(q);
1315                IF q = 'S' THEN enter := source
1316                ELSE enter := sample;
1317                FOR i := 0 TO 1 DO
1318                  BEGIN
1319                  displayfile(enter, i);
1320                  return(0);
1321                  END;
1322                END;
1323              'P':
1324                BEGIN
1325                q := 'D';
1326                IF plotsok THEN
1327                  BEGIN
1328                  prompt(top);
1329                  write('print D(ata or P(lot of data (D,P) [D] :  ');
1330                  readch(q);
1331                  END;
1332                IF q = 'P' THEN
1333                  BEGIN
1334                  doplots(enter, source, sample, la100, ff, spooled, user);
1335                  END
1336                ELSE
1337                  BEGIN
1338                  doprint(enter, source, sample, la100, ff, spooled, user);
1339                  END;
1340                END;
1341              'Z': clearvalues(source, sample, havesource);
1342              'G': radexc;
1343              'V': rbratio(rbok, la100, spooled, user);
1344              'S':
1345                IF mainmenu = false THEN mainmenu := true
1346                ELSE mainmenu := false;
1347              'X':
1348                BEGIN
1349                prompt(top);
1350                IF NOT exitok THEN
1351                  BEGIN
1352                  write('enter authorization code to EXIT :  ', chr(29), 'F');
1353                  readst(key);
```

```
1354                    writeln(chr(29), 'E');
1355                    IF key = exitkey THEN qm := 'X'
1356                    ELSE qm := 'A';
1357                    END;
1358                  END;
1359                END;
1360              END;
1361            UNTIL qm = 'X';
1362          END;
1363
1364
1365        BEGIN
1366          noctrlc;
1367          noioerror(input);
1368          user := 'R. Miller                                ';
1369          prompt(top);
1370          writeln('Please wait while necessary files are being loaded');
1371          number := 0;
1372          justsource := false;
1373          havesource := false;
1374          cie1931 := true;
1375          getopt(exitok, la100, ff, plotsok, rbok, spooled, chgoptkey, exitkey, version, defdev, defsydev,
1376                 defext, defsyext);
1377          getxyz(x10, y10, z10, cie1931, degree, defsydev, defsyext);
1378          clearvalues(source, sample, havesource);
1379          fancy1(version);
1380          getuser(user);
1381        1:
1382          BEGIN
1383          menu;
1384          END;
1385          writeln('Bye...');
1386        END.
```

*** No lines with errors detected ***

```
    1       PROGRAM colr00;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR00.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR00
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          Procedure made external but not overlayed/ 04-22-84/ R. M. Miller
   13          Input IO error trapping/ 10-20-84/ R.M. Miller
   14          Option added to chop any data points over one before printing
   15            spectral values/ 05-13-85/ R.M. Miller
   16
   17
   18                              BRIEF PROGRAM DESCRIPTION
   19
   20          Procedures for autocalulation of large data sets
   21
   22          end of informational heading }
   23
   24       { ------------------------------------------------------------------
   25        externally defined procedures that are external to the main program
   26        ------------------------------------------------------------------- }
   27
   28        %INCLUDE 'colrmn';
2   1       { COLRMN.PAS contains the main program definitions }
2   2
2   3       {$nolist}
2 369       {$list}
   29
   30
   31         PROCEDURE autoprint;
   32
   33           CONST
   34             maxfiles = 15;
   35
   36           VAR
   37             f: FILE OF data;
   38             i1, i, j, k, len, copies, number: integer;
   39             q: char;
   40             chop_data, auxff, printspectral, densities: boolean;
   41             filen: ARRAY [1..maxfiles] OF string14;
   42
   43           BEGIN
   44             prompt(top);
   45             writeln('Do you wish to print the spectral data along with the calculated');
   46             write('values ? (Y,N) [Y] :  ');
   47             readch(q);
   48             printspectral := q <> 'N';
   49             densities := false;
```

```
50            IF q <> 'N' THEN
51              BEGIN
52              writeln;
53              writeln('Do you wish spectral densities printed instead of spectral ');
54              write('reflectances/transmittances ? (Y,N) [N] :  ');
55              readch(q);
56              densities := q = 'Y';
57              writeln;
58              write('Should these spectral values never exceed 1.0 ? (Y,N) [N] :  ');
59              readch(q);
60              chop_data := q = 'Y';
61              END;
62            IF (ff) AND (printspectral) THEN
63              BEGIN
64              prompt(top);
65              writeln('do wish to print the spectral data and calculated values on one ');
66              write('page of paper ? (Y,N) [Y] :  ');
67              readch(q);
68              auxff := q = 'N';
69              END
70            ELSE auxff := false;
71            fillsource(source, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su, sv);
72            REPEAT
73              prompt(top);
74              readin('enter number of files to print calculations (1-15) :  ', j);
75            UNTIL (j > 0) AND (j < 16);
76            prompt(0);
77            FOR i := 1 TO j DO
78              BEGIN
79              REPEAT
80                write('enter filename (XXXXXX) :  ');
81                getfilename(filen[i]);
82                reset(f, filen[i], , len);
83                IF len = - 1 THEN
84                  BEGIN
85                  write('** Invalid Filename ** press RETURN to try again -or- ? to abort :  ');
86                  readch(q);
87                  END;
88              UNTIL len <> - 1;
89              END;
90            REPEAT
91              prompt(top);
92              readin('enter number of copies desired from each data set :  ', copies);
93            UNTIL (copies > 0) AND (copies < 30);
94            IF copies > 1 THEN
95              BEGIN
96              prompt(top);
97              write('Collated copies (Y,N) [Y] :  ');
98              readch(q);
99              IF q <> 'N' THEN
100               BEGIN
101               number := 1;
```

```
102                 END
103               ELSE
104                 BEGIN
105                 number := copies;
106                 copies := 1;
107                 END;
108               END
109             ELSE number := 1;
110             prompt(top);
111             writeln('Please wait while the calculations are being performed ....');
112             FOR k := 1 TO copies DO
113               BEGIN
114               FOR i := 1 TO j DO
115                 BEGIN
116                 retrievefile(filen[i], sample);
117                 IF chop_data THEN
118                   FOR i1 := 1 TO noofwavelengths DO IF sample.val[i1] >= 1.0 THEN sample.val[i1] := 1.0;
119                 IF densities THEN
120                   BEGIN
121                   FOR i1 := 1 TO noofwavelengths DO enter.val[i1] := density(sample.val[i1]);
122                   enter.descrip := 'Spec. Dens. of                ';
123                   FOR i1 := 16 TO stringmax DO enter.descrip[i1] := sample.descrip[i1 - 15];
124                   enter.inc := sample.inc;
125                   END;
126                 IF printspectral THEN
127                   BEGIN
128                   IF densities THEN printfile(enter, spooled, la100, auxff, user)
129                   ELSE printfile(sample, spooled, la100, auxff, user);
130                   END;
131                 dochrom(true, number);
132                 END;
133               END;
134             number := 0;
135           END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr01;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR01.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR01
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          Procedure Blackbd made external/ 04-22-84/ R.M. Miller
   13          Input IO Error Trapping/ 10-20-84/ R.M. Miller
   14          Writeln(out,chr(12)); changed to Page(out)/ 05-13-85/ R.M. Miller
   15
   16
   17
   18                              BRIEF PROGRAM DESCRIPTION
   19
   20          Procedures for Red/Blue calculations and printing of results
   21
   22          end of informational heading }
   23
   24       { ------------------------------------------------------------------
   25        externally defined procedures that are external to the main program
   26        ------------------------------------------------------------------ }
   27
   28        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   29
   30
   31         PROCEDURE rbratio(VAR rbok, la100, spooled: boolean;
   32                           VAR user: string);
   33           EXTERNAL;
   34
   35
   36         PROCEDURE radexc;
   37           EXTERNAL;
   38
   39
   40         PROCEDURE blackbd(VAR enter, source, sample: data;
   41                           VAR havesource: boolean);
   42           EXTERNAL;
   43
   44
   45         PROCEDURE rbratio;
   46
   47           VAR
   48             out: text;
   49             t0, t1, t2, rb, tinc: real;
```

```
50            line, number, i2, copies, rwave, bwave: integer;
51            q: char;
52
53          BEGIN
54            prompt(0);
55            IF rbok THEN
56              BEGIN
57              rewrite(out, 'LP:OUTPUT.OUT');
58              REPEAT
59                BEGIN
60                prompt(2);
61                readln('enter wavelength in nm for RED light :  ', rwave);
62                readln('enter wavelength in nm for BLUE light :  ', bwave);
63                END;
64              UNTIL (rwave > bwave) AND (rwave < 800.0) AND (bwave > 300.0);
65              writeln;
66              REPEAT
67                BEGIN
68                readre('enter lower temperature of range in K :  ', t1);
69                readre('enter higher temperature of range in K :  ', t2);
70                END;
71              UNTIL (t2 > t1) AND (t1 > 999.0) AND (t2 < 20000.0);
72              writeln;
73              REPEAT
74                BEGIN
75                readre('enter temperature increment  in K :  ', tinc);
76                number := round((t2 - t1) / tinc);
77                IF number > 100 THEN
78                  BEGIN
79                  writeln(number: 6, ' R/B values need to be printed using this increment');
80                  writeln;
81                  write('C(hange increment -or- press any other key to use current value(C) [C] :  ');
82                  readch(q);
83                  END;
84                END;
85              UNTIL (q <> 'C') OR (number <= 100);
86              writeln;
87              writeln;
88              write('M(ultiple copies -or- any other key for one copy (M) :  ');
89              readch(q);
90              IF q = 'M' THEN
91                BEGIN
92                readln('enter number of copies desired :  ', copies);
93                END
94              ELSE copies := 1;
95              FOR i2 := 1 TO copies DO
96                BEGIN
97                t0 := t1;
98                line := 51;
99                REPEAT
100               BEGIN
101               IF line >= 50 THEN
```

```
102                     BEGIN
103                     IF line <> 51 THEN page(out);
104                     header(out, la100, spooled, user);
105                     writeln(out, 'Red/Blue Ratio Calculations');
106                     writeln(out);
107                     writeln(out, 'Red wavelength in nm : ', rwave);
108                     writeln(out, 'Blue wavelength in nm : ', bwave);
109                     writeln(out);
110                     writeln(out, 'temperature in K': 20, 'R/B Ratio': 20);
111                     writeln(out);
112                     line := 0;
113                     END;
114                   rb := plank(rwave, t0) / plank(bwave, t0);
115                   writeln(out, t0: 20: 4, rb: 20: 6);
116                   t0 := t0 + tinc;
117                   line := line + 1;
118                   END;
119                 UNTIL t0 > t2;
120                 page(out);
121                 END;
122               close(out);
123               END;
124           END;
125
126
127         PROCEDURE radexc;
128
129           VAR
130             l: integer;
131             ctemp: real;
132
133           BEGIN
134             prompt(0);
135             writeln('spectral radiant excitance calculator');
136             jump(2);
137             writeln('this option calculates values of spectral radiant excitance');
138             writeln('for a blackbody radiator using Planck''s equation');
139             writeln;
140             writeln('the units are in watts per cubic meter');
141             return(1);
142             REPEAT
143               BEGIN
144               prompt(top);
145               writeln('enter 0 to exit OR');
146               readin('enter wavelength in nm (an integer) :  ', l);
147               IF l <> 0 THEN
148                 BEGIN
149                 writeln;
150                 readre('enter temperature in K :  ', ctemp);
151                 jump(1);
152                 write('The spectral radiant excitance is ', plank(l, ctemp): 10);
153                 writeln(' watts per cubic meter');
```

```
154                 return(2);
155                 END;
156               ENO;
157             UNTIL l = 0;
158           END;
159
160
161         PROCEDURE blackbd;
162
163           VAR
164             normalizer, ctemp: real;
165             w2, i: integer;
166             q: char;
167
168           BEGIN
169             prompt(top);
170             writeln('Black Body Radiator Calculation for Source');
171             jump(3);
172             readre('enter source temperature in Kelvin :  ', ctemp);
173             w2 := 560;
174             normalizer := 100 / plank(w2, ctemp);
175             FOR i := 1 TO noofwavelengths OO
176               BEGIN
177               w2 := startingwavelength + (i - 1) * mininc;
178               enter.val[i] := plank(w2, ctemp) * normalizer;
179               END;
180             enter.inc := 5;
181             prompt(top);
182             write('enter description for this data :  ');
183             Rstring(enter.descrip);
184             write('save this data to disk (Y,N) [N] :  ');
185             readch(q);
186             IF q = 'Y' THEN savefile(enter);
187             prompt(top);
188             write('is this S(ource  -OR-  sA(mple data -OR- N(either ? (S,A,N) [S] :  ');
189             readch(q);
190             IF q = 'A' THEN sample := enter
191             ELSE IF q <> 'N' THEN
192               BEGIN
193               source := enter;
194               havesource := true;
195               END;
196           END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr02;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR02.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR02
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          Input Error Trapping/ 10-20-84/ R.M. Miller
   13
   14
   15                              BRIEF PROGRAM DESCRIPTION
   16
   17          Procedures for Distribution temperature calculations
   18
   19          end of informational heading }
   20
   21       { ------------------------------------------------------------------------
   22        externally defined procedures that are external to the main program
   23        ------------------------------------------------------------------------ }
   24
   25        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   26
   27
   28         PROCEDURE distributiontemp(VAR source: data);
   29           EXTERNAL;
   30
   31
   32         PROCEDURE distributiontemp;
   33
   34           VAR
   35             temp, tempinc, oldsum, voldsum, bestguess, sum, sum1, sum2: real;
   36             v: ARRAY [1..numberofwavelengths] OF real;
   37             numbtries, wavet, index, i1, i2: integer;
   38             low: boolean;
   39             test, testsm: ARRAY [1..100] OF real;
   40
   41           BEGIN
   42             prompt(top);
   43             readre('enter highest temperature likely in Kelvin :  ', temp);
   44             prompt(1);
   45             temp := temp + 2000.0;
   46             numbtries := 0;
   47             writeln('starting at T = ', temp: 10: 4, 'K');
   48             tempinc := 1000.0;
   49             oldsum := 1.0E8;
```

```
50            sum := 1.0E7;
51            i2 := 0;
52            REPEAT
53              BEGIN
54              REPEAT
55                BEGIN
56                i2 := i2 + 1;
57                test[i2] := temp;
58                numbtries := numbtries + 1;
59                wavet := startingwavelength - mininc;
60                sum1 := 0;
61                sum2 := 0;
62                FOR i1 := 1 TO noofwavelengths DO
63                  BEGIN
64                  wavet := wavet + mininc;
65                  v[i1] := source.val[i1] / plank(wavet, temp);
66                  sum1 := sum1 + v[i1];
67                  sum2 := sum2 + sqr(v[i1]);
68                  END;
69                sum1 := sum1 / sum2;
70                voldsum := oldsum;
71                oldsum := sum;
72                sum := 0;
73                FOR i1 := 1 TO noofwavelengths DO sum := sum + sqr(1 - v[i1] * sum1);
74                write('T = ', temp: 10: 4, 'K    RMS Error = ', sum: 10: - 4);
75                writeln;
76                temp := temp - tempinc;
77                testsm[i2] := sum;
78                END;
79              UNTIL (sum > oldsum) OR ((sum = oldsum) AND (sum = voldsum)) OR (sum < 2.0E-7);
80              IF sum > oldsum THEN
81                BEGIN
82                IF i2 > 3 THEN
83                  BEGIN
84                  temp := test[i2 - 2];
85                  sum := testsm[i2 - 2];
86                  oldsum := testsm[i2 - 3];
87                  END
88                ELSE
89                  BEGIN
90                  temp := test[1];
91                  sum := testsm[1];
92                  oldsum := 1.0E9;
93                  END;
94                i2 := 1;
95                test[1] := temp;
96                testsm[1] := sum;
97                tempinc := tempinc / 2;
98                temp := temp - tempinc;
99                END
100             ELSE temp := temp + tempinc;
101             END;
```

```
102             UNTIL (numbtries > 100) OR (tempinc < 0.01) OR ((sum = oldsum) AND (sum = voldsum)) OR
103                   (sum < 2.0E-7);
104             writeln;
105             writeln;
106             writeln('Oistribution Temperature :  ', temp: 10: 4, ' K');
107             writeln;
108             writeln;
109             writeln('Source Description :  ', source.descrip);
110             return(2);
111           END;
```

*** No lines with errors detected ***

```
    1       PROGRAM colr03;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR03.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR03
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12           Input Error Trapping/ 10-20-84/ R.M. Miller
   13
   14
   15                              BRIEF PROGRAM DESCRIPTION
   16
   17          Procedures for Correlated color temperature calculations
   18          Adapted from Fortran routines written by A.R. Robertson
   19
   20          end of informational heading }
   21
   22       { ---------------------------------------------------------------------
   23        externally defined procedures that are external to the main program
   24        --------------------------------------------------------------------- }
   25
   26        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   27
   28
   29         PROCEDURE corcol(VAR u, v, cctemp: real);
   30           EXTERNAL;
   31
   32
   33         PROCEDURE duct(VAR uct: r30);
   34
   35           BEGIN
   36             uct[1] := 0.18065;
   37             uct[2] := 0.18132;
   38             uct[3] := 0.18208;
   39             uct[4] := 0.18293;
   40             uct[5] := 0.18388;
   41             uct[6] := 0.18494;
   42             uct[7] := 0.18611;
   43             uct[8] := 0.18739;
   44             uct[9] := 0.18879;
   45             uct[10] := 0.19031;
   46             uct[11] := 0.19461;
   47             uct[12] := 0.19960;
   48             uct[13] := 0.20523;
   49             uct[14] := 0.21140;
```

```
50            uct[15] := 0.21804;
51            uct[16] := 0.22507;
52            uct[17] := 0.23243;
53            uct[18] := 0.24005;
54            uct[19] := 0.24787;
55            uct[20] := 0.25585;
56            uct[21] := 0.26394;
57            uct[22] := 0.27210;
58            uct[23] := 0.28032;
59            uct[24] := 0.28854;
60            uct[25] := 0.29676;
61            uct[26] := 0.30496;
62            uct[27] := 0.31310;
63            uct[28] := 0.32119;
64            uct[29] := 0.32920;
65            uct[30] := 0.33713;
66          END;
67
68
69        PROCEDURE dvct(VAR vct: r30);
70
71          BEGIN
72            vct[1] := 0.26589;
73            vct[2] := 0.26845;
74            vct[3] := 0.27118;
75            vct[4] := 0.27407;
76            vct[5] := 0.27708;
77            vct[6] := 0.28020;
78            vct[7] := 0.28340;
79            vct[8] := 0.28666;
80            vct[9] := 0.28995;
81            vct[10] := 0.29325;
82            vct[11] := 0.30139;
83            vct[12] := 0.30918;
84            vct[13] := 0.31645;
85            vct[14] := 0.32309;
86            vct[15] := 0.32906;
87            vct[16] := 0.33436;
88            vct[17] := 0.33901;
89            vct[18] := 0.34305;
90            vct[19] := 0.34653;
91            vct[20] := 0.34948;
92            vct[21] := 0.35198;
93            vct[22] := 0.35405;
94            vct[23] := 0.35575;
95            vct[24] := 0.35713;
96            vct[25] := 0.35822;
97            vct[26] := 0.35906;
98            vct[27] := 0.35968;
99            vct[28] := 0.36011;
100           vct[29] := 0.36038;
101           vct[30] := 0.36051;
```

```
102           END;
103
104
105         PROCEDURE dslope(VAR slope: r30);
106
107           BEGIN
108             slope[1] := - 0.2548;
109             slope[2] := - 0.2687;
110             slope[3] := - 0.2854;
111             slope[4] := - 0.3047;
112             slope[5] := - 0.3267;
113             slope[6] := - 0.3515;
114             slope[7] := - 0.3790;
115             slope[8] := - 0.4094;
116             slope[9] := - 0.4426;
117             slope[10] := - 0.4787;
118             slope[11] := - 0.5517;
119             slope[12] := - 0.7043;
120             slope[13] := - 0.8484;
121             slope[14] := - 1.017;
122             slope[15] := - 1.216;
123             slope[16] := - 1.450;
124             slope[17] := - 1.728;
125             slope[18] := - 2.016;
126             slope[19] := - 2.465;
127             slope[20] := - 2.96;
128             slope[21] := - 3.576;
129             slope[22] := - 4.355;
130             slope[23] := - 5.365;
131             slope[24] := - 6.711;
132             slope[25] := - 8.572;
133             slope[26] := - 11.29;
134             slope[27] := - 15.56;
135             slope[28] := - 23.20;
136             slope[29] := - 40.41;
137             slope[30] := - 113.8;
138           END;
139
140
141         PROCEDURE dcct(VAR cct: r30);
142
143           BEGIN
144             cct[1] := 100000.0;
145             cct[2] := 50000.0;
146             cct[3] := 33333.0;
147             cct[4] := 25000.0;
148             cct[5] := 20000.0;
149             cct[6] := 16667.0;
150             cct[7] := 14285.0;
151             cct[8] := 12500.0;
152             cct[9] := 11111.0;
153             cct[10] := 10000.0;
```

```
154             cct[11] := 8000.0;
155             cct[12] := 6667.0;
156             cct[13] := 5714.0;
157             cct[14] := 5000.0;
158             cct[15] := 4444.0;
159             cct[16] := 4000.0;
160             cct[17] := 3636.0;
161             cct[18] := 3333.0;
162             cct[19] := 3077.0;
163             cct[20] := 2857.0;
164             cct[21] := 2667.0;
165             cct[22] := 2500.0;
166             cct[23] := 2353.0;
167             cct[24] := 2222.0;
168             cct[25] := 2105.0;
169             cct[26] := 2000.0;
170             cct[27] := 1905.0;
171             cct[28] := 1818.0;
172             cct[29] := 1739.0;
173             cct[30] := 1667.0;
174           END;
175
176
177         PROCEDURE corcol;
178
179           VAR
180             d, uct, vct, slope, cct: r30;
181             i1, i2, i3: integer;
182             dtest: real;
183
184           BEGIN
185             duct(uct);
186             dvct(vct);
187             dslope(slope);
188             dcct(cct);
189             FOR i1 := 1 TO 30 DO
190               d[i1] := (v - vct[i1] - (slope[i1] * (u - uct[i1]))) / sqrt(1 + sqr(slope[i1]));
191             i1 := 0;
192             REPEAT
193               BEGIN
194               i1 := i1 + 1;
195               dtest := d[i1] / d[i1 + 1];
196               END;
197             UNTIL (i1 = 29) OR (dtest < 0.0);
198             cctemp := 1 / cct[i1] + (d[i1] / (d[i1] - d[i1 + 1])) * (1 / cct[i1 + 1] - 1 / cct[i1]);
199             cctemp := 1 / cctemp + 0.05;
200           END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr04;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR04.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR04
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          procedure chgxyz added/ 04-22-84/ R.M. Miller
   13          Input Error Trapping/ 10-20-84/ R.M. Miller
   14          Logic of CHGXYZ changed/ 05-13-85/ R.M. Miller
   15
   16
   17                              BRIEF PROGRAM DESCRIPTION
   18
   19          Procedures for the fancy menu, initializing source and sample
   20          variables, and for loading the color matching functions into
   21          memory
   22
   23          end of informational heading }
   24
   25       { ---------------------------------------------------------------------
   26        externally defined procedures that are external to the main program
   27        --------------------------------------------------------------------- }
   28
   29        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   30
   31
   32         PROCEDURE noctrlc;
   33           EXTERNAL;
   34
   35
   36         PROCEDURE fancy1(version: real);
   37           EXTERNAL;
   38
   39
   40         PROCEDURE clearvalues(VAR source, sample: data;
   41                               VAR havesource: boolean);
   42           EXTERNAL;
   43
   44
   45         PROCEDURE getxyz(VAR x10, y10, z10: data;
   46                          VAR b1: boolean;
   47                          VAR degree: string2;
   48                          VAR defsydev, defsyext: string3);
   49           EXTERNAL;
```

```
50
51
52          PROCEDURE chgxyz(VAR x10, y10, z10: data;
53                           VAR cie1931: boolean;
54                           VAR degree: string2;
55                           VAR defsydev, defsyext: string3);
56            EXTERNAL;
57
58
59          PROCEDURE scca(VAR i3: integer);
60            NONPASCAL;
61
62
63          PROCEDURE noctrlc;
64
65            VAR
66              is: integer;
67
68            BEGIN
69              is := 0;
70              scca(is);
71            END;
72
73
74          PROCEDURE fancy1;
75
76            VAR
77              i, j: integer;
78
79            BEGIN
80              writeln(chr(29), 'J');
81              write(chr(27), 'H', chr(27), 'J');
82              FOR i := 1 TO 3 DO
83                BEGIN
84                FOR j := 1 TO 69 DO
85                  BEGIN
86                  write('=');
87                  END;
88                writeln;
89                END;
90              write(chr(27), 'Y', chr(9 + 32), chr(0 + 32));
91              FOR i := 1 TO 3 DO
92                BEGIN
93                FOR j := 1 TO 69 DO
94                  BEGIN
95                  write('=');
96                  END;
97                IF i <> 3 THEN writeln;
98                END;
99              write(chr(27), 'Y', chr(5 + 32), chr(6 + 32));
100             write('Welcome to the Color Measurement Program [Version ');
101             IF round(version * 10.0) >= 100 THEN write(version: 4: 1)
```

```
102             ELSE write(version: 3: 1);
103             writeln(') .....');
104             return(0);
105           END;
106
107
108         PROCEDURE clearvalues;
109
110           VAR
111             i1: integer;
112
113           BEGIN
114             havesource := false;
115             FOR i1 := 1 TO noofwavelengths DO
116               BEGIN
117               source.val[i1] := 0;
118               sample.val[i1] := 1;
119               END;
120             source.inc := 5;
121             sample.inc := 5;
122             FOR i1 := 1 TO stringmax DO
123               BEGIN
124               source.descrip[i1] := chr(0);
125               sample.descrip[i1] := chr(0);
126               END;
127           END;
128
129
130         PROCEDURE getxyz;
131
132           VAR
133             fn: string14;
134
135           BEGIN
136             IF NOT b1 THEN
137               BEGIN
138               concat(fn, defsydev, 'XBAR10', defsyext);
139               retrievefile(fn, x10);
140               concat(fn, defsydev, 'YBAR1D', defsyext);
141               retrievefile(fn, y10);
142               concat(fn, defsydev, 'ZBAR10', defsyext);
143               retrievefile(fn, z10);
144               degree := '10';
145               END
146             ELSE
147               BEGIN
148               concat(fn, defsydev, 'XBAR02', defsyext);
149               retrievefile(fn, x10);
150               concat(fn, defsydev, 'YBAR02', defsyext);
151               retrievefile(fn, y10);
152               concat(fn, defsydev, 'ZBAR02', defsyext);
153               retrievefile(fn, z10);
```

```
154             degree := '02';
155             END;
156         END;
157
158
159       PROCEDURE chgxyz;
160
161         VAR
162           qt: char;
163
164         BEGIN
165           prompt(top);
166           writeln('CIE Observers that can be selected :');
167           writeln;
168           writeln('    1931 CIE  2 degree observer');
169           writeln('    1964 CIE 10 degree observer');
170           jump(2);
171           writeln('current observer selected is ', degree, ' degree observer');
172           jump(2);
173           write('select T(wo degree, tE(n degree, or C(urrent observer (T,E,C) [C] :  ');
174           readch(qt);
175           IF qt = 'T' THEN
176             BEGIN
177             cie1931 := true;
178             getxyz(x10, y10, z10, cie1931, degree, defsydev, defsyext);
179             END
180           ELSE IF qt = 'E' THEN
181             BEGIN
182             cie1931 := false;
183             getxyz(x10, y10, z10, cie1931, degree, defsydev, defsyext);
184             END;
185         END;
```

*** No lines with errors detected ***

```
    1       PROGRAM colr05;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR05.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR05
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          Variable NUMBER passed to procedure pchrom to eliminate operator IO
   13          during an automatic data output session/ 12-20-83/ R.M. Miller
   14          Formfeed option added to printing routines/ 12-28-83/ R.M. Miller
   15
   16
   17                              BRIEF PROGRAM DESCRIPTION
   18
   19          Procedure for printing a data file and the color parameters
   20
   21          end of informational heading }
   22
   23       { ------------------------------------------------------------------
   24        externally defined procedures that are external to the main program
   25        ------------------------------------------------------------------ }
   26
   27        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   28
   29
   30         PROCEDURE printfile(VAR s1: data;
   31                             spooled, la100, ff: boolean;
   32                             VAR user: string);
   33           EXTERNAL;
   34
   35
   36         PROCEDURE pchrom(VAR source, sample: data;
   37                          VAR cx, cy, cz, tx, ty, tz, u, v, lstar, astar, bstar, cab, hab, ustar, vstar,
   38                           cuv, huv, domwav, purity: real;
   39                          VAR degree: string2;
   40                          VAR user: string;
   41                          VAR la100, ff, spooled, complementary, domwaverror, hueaberror,
   42                          hueuverror: boolean;
   43                          number: integer);
   44           EXTERNAL;
   45
   46
   47         PROCEDURE doprint(VAR enter, source, sample: data;
   48                           VAR la100, ff, spooled: boolean;
   49                           VAR user: string);
```

```
50          EXTERNAL;
51
52
53        PROCEDURE pchrom;
54
55          VAR
56            out: text;
57            sam, sou: string;
58            i, j: integer;
59            q: char;
60
61          BEGIN
62            sam := sample.descrip;
63            sou := source.descrip;
64            IF ord(sam[1]) < 32 THEN sam[1] := chr(32);
65            IF ord(sou[1]) < 32 THEN sou[1] := chr(32);
66            IF number = 0 THEN
67              BEGIN
68              prompt(4);
69              write('M(ultiple copies -or- any other key for one copy (M) :  ');
70              readch(q);
71              IF q = 'M' THEN
72                BEGIN
73                writeln;
74                writeln;
75                readin('          enter number of copies desired :  ', j);
76                END
77              ELSE j := 1;
78              END
79            ELSE j := number;
80            FOR i := 1 TO j DO
81              BEGIN
82              rewrite(out, 'LP:OUTPUT.OUT');
83              header(out, la100, spooled, user);
84              writeln(out, 'Sample Description [', sample.inc: 3, ' nm wavelength increment ] :  ', sam);
85              writeln(out, 'Source Description [', source.inc: 3, ' nm wavelength increment ] :  ', sou);
86              writeln(out);
87              writeln(out, 'Tristimulus Values :': 30, 'X :  ', tx: 9: 5);
88              writeln(out, '(': 13, degree: 2, 'degree)': 8, ' ': 7, 'Y :  ', ty: 9: 5);
89              writeln(out, ' ': 30, 'Z :  ', tz: 9: 5);
90              writeln(out);
91              writeln(out, 'Chromaticity Coordinates :': 30, 'x :  ', cx: 9: 7, ' ': 10, 'u :  ', u: 9:
92                    7);
93              writeln(out, ' ': 30, 'y :  ', cy: 9: 7, ' ': 10, 'v :  ', v: 9: 7);
94              writeln(out, ' ': 30, 'z :  ', cz: 9: 7);
95              writeln(out);
96              IF NOT domwaverror THEN
97                BEGIN
98                writeln(out);
99                writeln(out);
100               IF complementary THEN write(out, '** **  Complementary Wavelength :  ': 40)
101               ELSE write(out, 'Dominant Wavelength :  ': 40);
```

```
102                 write(out, domwav: 6: 2, ' nm');
103                 IF complementary THEN writeln(out, ' C')
104                 ELSE writeln(out);
105                 writeln(out);
106                 writeln(out, 'Excitation Purity :  ': 40, (purity * 100.0): 6: 4, ' Percent');
107                 END;
108               writeln(out);
109               writeln(out);
110               writeln(out, 'CIELAB Coordinates');
111               writeln(out);
112               writeln(out, ' ': 2, 'L* :  ': 10, lstar: 9: 5, ' ': 10, 'a* :  ': 10, astar: 9: 5, ' ': 10,
113                       'b* :  ': 10, bstar: 9: 5);
114               write(out, ' ': 2, 'C*ab :  ': 10, cab: 9: 5, ' ': 10);
115               IF NOT hueaberror THEN write(out, 'Hab :  ': 10, hab: 9: 5, ' degrees');
116               writeln(out);
117               writeln(out);
118               writeln(out, 'CIELUV Coordinates');
119               writeln(out);
120               writeln(out, ' ': 2, 'L* :  ': 10, lstar: 9: 5, ' ': 10, 'u* :  ': 10, ustar: 9: 5, ' ': 10,
121                       'v* :  ': 10, vstar: 9: 5);
122               write(out, ' ': 2, 'C*uv :  ': 10, cuv: 9: 5, ' ': 10);
123               IF NOT hueuverror THEN write(out, 'Huv :  ': 10, huv: 9: 5, ' degrees');
124               IF ff THEN page(out);
125               close(out);
126               END;
127           END;
128
129
130         PROCEDURE printfile;
131
132           VAR
133             qt: char;
134             out: text;
135             wave, i, j: integer;
136             v1: real;
137
138           BEGIN
139             rewrite(out, 'LP:OUTPUT.OUT');
140             header(out, la100, spooled, user);
141             writeln(out);
142             writeln(out, 'File Description :  ', s1.descrip);
143             writeln(out);
144             writeln(out, 'Wavelength Increment :  ', s1.inc, ' nm');
145             writeln(out);
146             writeln(out, '                File Data   (wavelengths in nm)');
147             FOR i := 1 TO 20 DO
148               BEGIN
149               FOR j := 0 TO 3 DO
150                 BEGIN
151                 wave := startingwavelength + (i + j * 20 - 1) * mininc;
152                 IF wave <= endingwavelength THEN
153                   BEGIN
```

```
154                 v1 := s1.val[i + j * 20];
155                 IF (((v1 < 1000.0) AND (v1 > 0.001)) OR (v1 = 0.0)) THEN
156                   BEGIN
157                   write(out, wave: 3, '>', v1: 10: 4)
158                   END
159                 ELSE write(out, wave: 3, '>', v1: 10: - 3);
160                 END;
161               IF j <> 3 THEN write(out, ' ': 4)
162               ELSE writeln(out);
163               END;
164             END;
165           IF ff THEN page(out);
166           close(out);
167         END;
168
169
170       PROCEDURE doprint;
171
172         VAR
173           qt: char;
174           fn: string14;
175
176         BEGIN
177           prompt(top);
178           write('print current S(ource or sA(mple data, or R(etrieve file (S,A,R) [R] :  ');
179           readch(qt);
180           IF qt = 'S' THEN enter := source
181           ELSE IF qt = 'A' THEN enter := sample
182           ELSE
183             BEGIN
184             prompt(top);
185             write('enter file name to print (xxxxxx) :  ');
186             getfilename(fn);
187             retrievefile(fn, enter);
188             END;
189           printfile(enter, spooled, la100, ff, user);
190         END;
```

*** No lines with errors detected ***

```
    1       PROGRAM colr06;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR06.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR06
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12         Input IO Error Trapping Added/ 10-20-84/ R.M. Miller
   13
   14                           BRIEF PROGRAM DESCRIPTION
   15
   16          Dominant Wavelength and Excitation Purity Calculations
   17
   18          end of informational heading }
   19
   20       { ---------------------------------------------------------------------
   21        externally defined procedures that are external to the main program
   22        ---------------------------------------------------------------------- }
   23
   24        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   25
   26
   27         PROCEDURE domwav(VAR x, y, xilluminant, yilluminant, dominantwavelength, excpurity: real;
   28                          VAR complementary, domwaverror: boolean;
   29                          defsydev: string3);
   30           EXTERNAL;
   31
   32
   33         PROCEDURE invtan(VAR invtn, xpt, ypt, xilluminant, yilluminant: real;
   34                          VAR domwaverror: boolean);
   35
   36           VAR
   37             dx, dy: real;
   38             q: char;
   39
   40           BEGIN
   41             dx := xpt - xilluminant;
   42             dy := ypt - yilluminant;
   43             IF dx = 0.0 THEN
   44               BEGIN
   45               IF dy > 0.0 THEN invtn := 1.570796327;
   46               IF dy < 0.0 THEN invtn := 4.712388981;
   47               IF dy = 0.0 THEN
   48                 BEGIN
   49                 writeln('Warning -- Invalid Conditions');
```

```
50                writeln('  Improper chromaticities have been encountered');
51                return(0);
52                domwaverror := true;
53                invtn := 0.0;
54                END;
55              END
56            ELSE IF dy = 0.0 THEN
57              BEGIN
58              IF dx > 0.0 THEN invtn := 0.0
59              ELSE invtn := 3.141592654;
60              END
61            ELSE
62              BEGIN
63              invtn := arctan(dy / dx);
64              IF dx < 0 THEN invtn := invtn + 3.141592654
65              ELSE IF dy < 0 THEN invtn := invtn + 6.283185308;
66              END;
67          END;
68
69
70        PROCEDURE domwav;
71
72          VAR
73            i, j, k, l, region: integer;
74            xchrom, ychrom: chrom;
75            f: FILE OF chrom;
76            fn: string14;
77            endptx, endpty, angles: ARRAY [0..6] OF real;
78            distance1, distance2, distance3, fraction, temp, m1, m2, xp, yp, angle, newangle, testangle,
79             testangle2: real;
80            regionfound: boolean;
81            q: char;
82
83          BEGIN
84            endptx[0] := 0.17334;
85            endpty[0] := 0.00480;
86            endptx[1] := 0.15664;
87            endpty[1] := 0.01771;
88            endptx[2] := 0.00817;
89            endpty[2] := 0.53842;
90            endptx[3] := 0.30160;
91            endpty[3] := 0.69231;
92            endptx[4] := 0.62704;
93            endpty[4] := 0.37249;
94            endptx[5] := 0.72599;
95            endpty[5] := 0.27401;
96            endptx[6] := 0.73469;
97            endpty[6] := 0.26531;
98            distance1 := sqrt(sqr(x - xilluminant) + sqr(y - yilluminant));
99            domwaverror := false;
100
101           { Calculate angle of region endpoint vectors and for point being evaluated }
```

```
102
103             invtan(angle, x, y, xilluminant, yilluminant, domwaverror);
104             FOR i := 0 TO 6 DO
105               BEGIN
106               IF NOT domwaverror THEN
107                 invtan(angles[i], endptx[i], endpty[i], xilluminant, yilluminant, domwaverror);
108               END;
109             { Determine if complementary wavelength }
110
111             IF NOT domwaverror THEN
112               BEGIN
113               complementary := false;
114               IF angles[6] > 2.356194491 THEN
115                 BEGIN
116                 IF (angle < angles[6]) AND (angle > angles[0]) THEN complementary := true;
117                 END
118               ELSE
119                 BEGIN
120                 IF angle > angles[0] THEN
121                   BEGIN
122                   IF (angle <= 3.141592654) OR (angle < angles[6]) THEN complementary := true;
123                   END;
124                 END;
125               IF complementary THEN
126                 BEGIN
127                 IF angle >= 3.141592654 THEN newangle := angle - 3.141592654
128                 ELSE newangle := angle + 3.141592654;
129                 END
130               ELSE newangle := angle;
131
132               { Find region that sample's dominant wavelength is in }
133
134               regionfound := false;
135               FOR i := 1 TO 6 DO
136                 BEGIN
137                 IF (newangle <= angles[i - 1]) AND (newangle >= angles[i]) THEN
138                   BEGIN
139                   region := i;
140                   regionfound := true;
141                   END;
142                 END;
143               IF NOT regionfound THEN
144                 BEGIN
145                 FOR i := 1 TO 6 DO
146                   BEGIN
147                   IF angles[i] > angles[i - 1] THEN
148                     BEGIN
149                     IF (newangle <= angles[i - 1]) OR (newangle >= angles[i]) THEN
150                       BEGIN
151                       region := i;
152                       regionfound := true;
153                       END;
```

```
154                     END;
155                   END;
156                 END;
157               END;
158             { Retrieve data for the desired region }
159
160             IF regionfound AND NOT domwaverror THEN
161               BEGIN
162               concat(fn, defsydev, 'CHROM2', 'CHR');
163               reset(f, fn);
164               FOR i := 1 TO region DO
165                 BEGIN
166                 xchrom := f^;
167                 get(f);
168                 ychrom := f^;
169                 get(f);
170                 END;
171               close(f);
172               i := - 1;
173               IF angles[region] < angles[region - 1] THEN
174                 BEGIN
175                 REPEAT
176                   BEGIN
177                   i := i + 1;
178                   IF NOT domwaverror THEN
179                     invtan(testangle, xchrom[i], ychrom[i], xilluminant, yilluminant, domwaverror);
180                   END;
181                 UNTIL (testangle <= newangle) OR domwaverror;
182                 END
183               ELSE
184                 BEGIN
185                 IF newangle > angles[region - 1] THEN newangle := newangle - 6.283185308;
186                 REPEAT
187                   BEGIN
188                   i := i + 1;
189                   IF NOT domwaverror THEN
190                     invtan(testangle, xchrom[i], ychrom[i], xilluminant, yilluminant, domwaverror);
191                   IF testangle > angles[region - 1] THEN testangle := testangle - 6.283185308;
192                   END;
193                 UNTIL (testangle <= newangle) OR domwaverror;
194                 END;
195               IF (testangle = newangle) AND NOT domwaverror THEN
196                 BEGIN
197                 dominantwavelength := region * 50.0 + i + 350.0;
198                 IF NOT complementary THEN
199                   BEGIN
200                   excpurity := distance1 / sqrt(sqr(xchrom[i] - xilluminant) + sqr(ychrom[i] -
201                                                 yilluminant));
202                   END;
203                 END
204               ELSE IF NOT domwaverror THEN
205                 BEGIN
```

```
206                invtan(testangle2, xchrom[i - 1], ychrom[i - 1], xilluminant, yilluminant, domwaverror);
207                IF testangle2 > angles[region - 1] THEN testangle2 := testangle2 - 6.283185308;
208                fraction := (testangle2 - newangle) / (testangle2 - testangle);
209                dominantwavelength := region * 50.0 + i + 349.0 + fraction;
210                IF NOT complementary THEN
211                  BEGIN
212                  distance2 := sqrt(sqr(xchrom[i] - xilluminant) + sqr(ychrom[i] - yilluminant));
213                  distance3 := sqrt(sqr(xchrom[i - 1] - xilluminant) + sqr(ychrom[i - 1] - yilluminant));
214                  excpurity := distance1 / (distance3 + fraction * (distance2 - distance3));
215                  END;
216                END;
217              IF complementary AND NOT domwaverror THEN
218                BEGIN
219                m2 := (endpty[6] - endpty[0]) / (endptx[6] - endptx[0]);
220                IF (x - xilluminant) = 0.0 THEN
221                  BEGIN
222                  xp := x;
223                  END
224                ELSE
225                  BEGIN
226                  m1 := (y - yilluminant) / (x - xilluminant);
227                  xp := (m1 * x - m2 * endptx[0] - y + endpty[0]) / (m1 - m2);
228                  END;
229                yp := m2 * (xp - endptx[6]) + endpty[6];
230                distance2 := sqrt(sqr(xp - xilluminant) + sqr(yp - yilluminant));
231                excpurity := distance1 / distance2;
232                END;
233              IF (excpurity > 1.0) AND NOT domwaverror THEN
234                BEGIN
235                writeln;
236                writeln('WARNING -- Excitation Purity is greater than 100 %');
237                return(0);
238                END;
239              END;
240            IF NOT regionfound OR domwaverror THEN
241              BEGIN
242              writeln('WARNING -- Dominant Wavelelength not found');
243              return(0);
244              dominantwavelength := 0.0;
245              excpurity := 0.0;
246              END;
247          END;
```

*** No lines with errors detected ***

```
    1       PROGRAM colr07;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR07.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR07
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Oate/ Programmer
   12           Selection of numbering veritcal axis changed/ 04-22-84/ R.M. Miller
   13          Input IO Error Trapping/ 10-20-84/ R.M. Miller
   14          User prompts changed/ 05-13-85/ R.M. Miller
   15
   16
   17                              BRIEF PROGRAM OESCRIPTION
   18
   19          Procedure for producing spectral plots
   20
   21          end of informational heading }
   22
   23       { ------------------------------------------------------------------
   24        externally defined procedures that are external to the main program
   25        ------------------------------------------------------------------ }
   26
   27        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   28
   29
   30         PROCEDURE plot(number: integer;
   31                        VAR s: plotd;
   32                        la100, ff, spooled: boolean;
   33                        VAR user, yaxislabel: string);
   34           EXTERNAL;
   35
   36
   37         PROCEDURE doplots(VAR enter, source, sample: data;
   38                           VAR la100, ff, spooled: boolean;
   39                           VAR user: string);
   40           EXTERNAL;
   41
   42
   43         PROCEDURE gotocolumn(VAR out: text;
   44                              column: integer);
   45
   46           BEGIN
   47             write(out, chr(27), '[');
   48             IF column >= 100 THEN write(out, column: 3)
   49             ELSE IF column >= 10 THEN write(out, '0': 1, column: 2)
```

```
50            ELSE write(out, '00': 2, column: 1);
51            write(out, '\');
52          END;
53
54
55        PROCEDURE printval(VAR out: text;
56                           VAR tick: integer;
57                           value: real);
58
59          BEGIN
60            IF (((value < 1000.0) AND (value > 0.001)) OR (value = 0.0)) THEN
61              BEGIN
62              write(out, value: 10: 4, ' -')
63              END
64            ELSE write(out, value: 10: - 3, ' -');
65            tick := - 1;
66          END;
67
68
69        PROCEDURE plot;
70
71          VAR
72            out: text;
73            i, j, k, tick, column: integer;
74            max, min, yinc: real;
75            sc: ARRAY [1..plotmax, 1..numberofwavelengths] OF integer;
76            q: char;
77            ylabeltemp: string101;
78
79          BEGIN
80            IF la100 THEN
81              BEGIN
82              rewrite(out, 'LP:OUTPUT.OUT');
83              header(out, la100, spooled, user);
84              FOR i := 1 TO 101 DO ylabeltemp[i] := ' ';
85              FOR i := 1 TO stringmax DO
86                BEGIN
87                IF ord(yaxislabel[i]) <> 0 THEN ylabeltemp[2 * i + 4] := yaxislabel[i];
88                END;
89              writeln(out, ' ': 20, 'Plot of ', yaxislabel, ' vs. Wavelength');
90              write(out, chr(27), '[4w', chr(27), '[3z');
91              min := 9.99E20;
92              max := - 9.99E20;
93              FOR i := 1 TO noofwavelengths DO
94                BEGIN
95                FOR j := 1 TO number DO
96                  BEGIN
97                  IF s[j].val[i] > max THEN max := s[j].val[i];
98                  IF s[j].val[i] < min THEN min := s[j].val[i];
99                  END;
100               END;
101             writeln('Plotting Options :');
```

```
102             writeln(' ': 5, 'R: Reflectance/Transmittance plot with range of 0 - 1.0');
103             writeln(' ': 5, 'D: Define plotting limits for Y axis');
104             writeln(' ': 5, 'P: scale limits to plot all data');
105             writeln;
106             write('select one of the above (R,D,P) [P] :  ');
107             readch(q);
108             CASE q OF
109               'R':
110                 BEGIN
111                 min := 0.0;
112                 max := 1.0;
113                 END;
114               'D':
115                 BEGIN
116                 REPEAT
117                   BEGIN
118                   writeln(chr(27), 'H', chr(27), 'J');
119                   writeln;
120                   readre('enter lower limit for Y axis :  ', min);
121                   writeln;
122                   readre('enter upper limit for Y axis :  ', max);
123                   END;
124                 UNTIL max > min;
125                 END;
126               OTHERWISE
127                 BEGIN
128                 IF max = min THEN
129                   BEGIN
130                   writeln;
131                   writeln('WARNING - this plot is of a straight horizonal line');
132                   writeln('          of value :  ', min);
133                   max := min + 1.0;
134                   return(0);
135                   END;
136                 END;
137             END;
138           yinc := (max - min) / 100.0;
139           FOR i := 1 TO noofwavelengths DO
140             BEGIN
141             FOR j := 1 TO number DO sc[j, i] := round((s[j].val[i] - min) / yinc);
142             END;
143           tick := - 1;
144           FOR i := 0 TO 100 DO
145             BEGIN
146             write(out, ylabeltemp[i + 1]: 1, ' ': 3);
147             tick := tick + 1;
148             IF (i MOD 10) = 0 THEN printval(out, tick, max - (max - min) * i / 100)
149             ELSE
150               BEGIN
151               IF tick >= 4 THEN
152                 BEGIN
153                 write(out, '-': 12);
```

```
154                   tick := - 1;
155                   END
156                 ELSE write(out, '|': 12);
157                 END;
158               FOR j := 1 TO noofwavelengths DO
159                 BEGIN
160                 FOR k := 1 TO number DO
161                   BEGIN
162                   IF (sc[k, j] = (100 - i)) THEN
163                     BEGIN
164                     column := round((j - 1) * 5 / 2) + 16;
165                     gotocolumn(out, column);
166                     write(out, (k - 1): 1);
167                     END;
168                   END;
169                 END;
170               IF i <> 100 THEN writeln(out);
171               END;
172             write(out, chr(27), '[016`|');
173             FOR i := 1 TO 38 DO
174               BEGIN
175               FOR j := 1 TO 4 DO write(out, '-');
176               write(out, '|');
177               END;
178             writeln(out);
179             writeln(out);
180             write(out, chr(27), '[015`');
181             FOR i := 0 TO 19 DO
182               BEGIN
183               gotocolumn(out, 15 + i * 10);
184               write(out, (380 + i * 20): 3);
185               END;
186             writeln(out);
187             writeln(out);
188             writeln(out, chr(27), '[100`Wavelength (nm)');
189             writeln(out);
190             write(out, chr(27), '[0w', chr(27), '[0z');
191             writeln(out);
192             FOR i := 1 TO number DO
193               writeln(out, 'Sample Number ', (i - 1): 1, '  Wavelength Increment = ', s[i].inc: 2,
194                       ' nm   File Description : ', s[i].descrip);
195             IF ff THEN page(out);
196             close(out);
197             END;
198         END;
199
200
201       PROCEDURE doplots;
202
203         VAR
204           qt, q: char;
205           fn: string14;
```

```
206             copies, number, i, j: integer;
207             yaxislabel: string;
208             s: plotd;
209
210           BEGIN
211             prompt(top);
212             writeln('This plotting routine can overlay up to 5 sets of data on one graph');
213             writeln;
214             write('How many sets of data do you wish this graph to contain [1] :  ');
215             readch(qt);
216             IF qt IN ['2'..'5'] THEN number := ord(qt) - 48
217             ELSE number := 1;
218             FOR i := 1 TO number DO
219               BEGIN
220               prompt(top);
221               IF number > 1 THEN writeln('For sample number ', i: 2);
222               write('print current S(ource or sA(mple data, or R(etrieve file (S,A,R) [R] :  ');
223               readch(qt);
224               IF qt = 'S' THEN enter := source
225               ELSE IF qt = 'A' THEN enter := sample
226               ELSE
227                 BEGIN
228                 prompt(top);
229                 write('enter file name to print (xxxxxx) :  ');
230                 getfilename(fn);
231                 retrievefile(fn, enter);
232                 END;
233               s[i] := enter;
234               END;
235             prompt(top);
236             write('enter Y AXIS LABEL (40 characters) :  ');
237             rstring(yaxislabel);
238             prompt(top);
239             write('M(ultiple copies of plot -or- press any other key for one copy (M) :  ');
240             readch(q);
241             IF q = 'M' THEN
242               BEGIN
243               REPEAT
244                 BEGIN
245                 writeln;
246                 readin('enter number of copies desired  (01-10) ', copies);
247                 END;
248               UNTIL (copies > 0) AND (copies < 11);
249               END
250             ELSE copies := 1;
251             FOR i := 1 TO copies DO plot(number, s, la100, ff, spooled, user, yaxislabel);
252           END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr08;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR08.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR08
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12           Input IO Error Checking/ 10-20-84/ R.M. Miller
   13          Function Check Ratio added to calculate values of f(ti/tin) for
   14            CIELAB and L* Calculations for when ti/tin < 0.008856/ 05-13-85/
   15            R.M. Miller
   16
   17
   18                            BRIEF PROGRAM DESCRIPTION
   19
   20          Procedure calculating CIELab and CIELuv coordinates and
   21            color differences
   22
   23          end of informational heading }
   24
   25       { ------------------------------------------------------------------
   26        externally defined procedures that are external to the main program
   27        ------------------------------------------------------------------ }
   28
   29        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   30
   31
   32         PROCEDURE ciecalc(VAR source, sample: data;
   33                           VAR havesource, la100, spooled: boolean;
   34                           VAR user: string);
   35           EXTERNAL;
   36
   37
   38         PROCEDURE hueangle(VAR x, y, h: real;
   39                            VAR error: boolean);
   40           EXTERNAL;
   41
   42
   43         PROCEDURE cieluv(VAR tx, ty, tz, stx, sty, stz, uprime, vprime, suprime, svprime, lstar, ustar,
   44                          vstar, cuv, huv: real;
   45                          VAR hueangleerroruv: boolean);
   46           EXTERNAL;
   47
   48
   49         PROCEDURE cielab(VAR tx, ty, tz, stx, sty, stz, lstar, astar, bstar, cab, hab: real;
```

```
50                          VAR hueangleerrorab: boolean);
51          EXTERNAL;
52
53
54        PROCEDURE spacestring(VAR s1: string);
55
56          VAR
57            i, ascii: integer;
58
59          BEGIN
60            FOR i := 1 TO stringmax OO
61              BEGIN
62              ascii := ord(s1[i]);
63              IF ascii < O THEN ascii := ascii + 128;
64              IF (ascii < 32) OR (ascii > 126) THEN s1[i] := chr(32);
65              ENO;
66          ENO;
67
68
69        PROCEOURE pcielabluv(VAR out: text;
70                             descrip: string;
71                             lstar, astar, bstar, cab, hab, ustar, vstar, cuv, huv: real;
72                             hueaberror, hueuverror: boolean);
73
74          BEGIN
75            writeln(out);
76            writeln(out, 'Sample :  ', descrip);
77            writeln(out);
78            writeln(out, 'CIELAB Coordinates');
79            writeln(out, 'L* :  ': 10, lstar: 9: 5, ' ': 10, 'a* :  ': 10, astar: 9: 5, ' ': 10, 'b* :  ':
80                    10, bstar: 9: 5);
81            write(out, 'C*ab :  ': 10, cab: 9: 5, ' ': 10);
82            IF NOT hueaberror THEN write(out, 'Hab :  ': 10, hab: 9: 5, ' degrees');
83            writeln(out);
84            writeln(out);
85            writeln(out, 'CIELUV Coordinates');
86            writeln(out, 'L* :  ': 10, lstar: 9: 5, ' ': 10, 'u* :  ': 10, ustar: 9: 5, ' ': 10, 'v* :  ':
87                    10, vstar: 9: 5);
88            write(out, 'C*uv :  ': 10, cuv: 9: 5, ' ': 10);
89            IF NOT hueuverror THEN write(out, 'Huv :  ': 10, huv: 9: 5, ' degrees');
90            writeln(out);
91          END;
92
93
94        FUNCTION check_ratio(ratio: real): real;
95
96          BEGIN
97            IF ratio < 0.008856 THEN ratio := 7.787 * ratio + 16 / 116
98            ELSE ratio := exp(ln(ratio) / 3);
99            check_ratio := ratio;
100         END;
101
```

```
102
103         PROCEDURE cielab;
104
105           VAR
106             q: char;
107             xratio, yratio, zratio: real;
108
109           BEGIN
110             xratio := check_ratio(tx / stx);
111             yratio := check_ratio(ty / sty);
112             zratio := check_ratio(tz / stz);
113             lstar := 116 * yratio - 16.0;
114             astar := 500.0 * (xratio - yratio);
115             bstar := 200.0 * (yratio - zratio);
116             cab := sqrt(sqr(astar) + sqr(bstar));
117             hueangle(astar, bstar, hab, hueangleerrorab);
118           END;
119
120
121         PROCEDURE checkval(VAR tx, ty, tz: real);
122
123           VAR
124             q: char;
125
126           BEGIN
127             REPEAT
128               BEGIN
129               IF (tx > 2.0) OR (ty > 1.1) OR (tz > 2.0) THEN
130                 BEGIN
131                 writeln('These tristimulus values seem very large');
132                 writeln('The tristimulus value Y cannot be greater than 1.0');
133                 writeln('Were these tristimulus values calculated for the');
134                 write('  source having  Y = 100  as with the ACS system ? (Y,N) [Y] :  ');
135                 readch(q);
136                 IF q <> 'N' THEN
137                   BEGIN
138                   tx := tx / 100.0;
139                   ty := ty / 100.0;
140                   tz := tz / 100.0;
141                   END
142                 ELSE
143                   BEGIN
144                   write('Enter ? to return to menu -or- T(ry entering again (?,T) [T] :  ');
145                   readch(q);
146                   writeln;
147                   readre('enter tristimulus value X :  ', tx);
148                   readre('enter tristimulus value Y :  ', ty);
149                   readre('enter tristimulus value Z :  ', tz);
150                   END;
151               END;
152             UNTIL NOT ((tx > 2.0) OR (ty > 1.1) OR (tz > 2.0));
153
```

```
154         END;
155
156
157       PROCEDURE ciecalc;
158
159         VAR
160           q, qt: char;
161           fn: string14;
162           out: text;
163           i, j, index, standard, numberofsets: integer;
164           cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su, sv, suprime, svprime, uprime,
165            vprime: real;
166           x, y, z, lstar, astar, bstar, ustar, vstar, cab, hab, cuv, huv: ARRAY [1..9] OF real;
167           xyzbyhand: boolean;
168           hueaberror, hueuverror: ARRAY [1..9] OF boolean;
169           deab, deuv: ARRAY [1..36] OF real;
170           descrip: ARRAY [0..9] OF string;
171
172         BEGIN
173           prompt(top);
174           writeln('Do you wish to R(etrieve a source''s tristimulus values');
175           write('  from a disk file -or- enter then by H(and (R,H) [R] :  ');
176           readch(q);
177           IF q = 'H' THEN
178             BEGIN
179             prompt(top);
180             xyzbyhand := true;
181             writeln('For the Source :');
182             readre('enter tristimulus value X :  ', stx);
183             readre('enter tristimulus value Y :  ', sty);
184             readre('enter tristimulus value Z :  ', stz);
185             writeln;
186             write('enter description for source :  ');
187             rstring(descrip[0]);
188             checkval(stx, sty, stz);
189             END
190           ELSE
191             BEGIN
192             xyzbyhand := false;
193             fillsource(source, cx, cy, cz, scx, scy, scz, tx, ty, tz, stx, sty, stz, u, v, su, sv);
194             descrip[0] := source.descrip;
195             END;
196           REPEAT
197             BEGIN
198             prompt(top);
199             write('How many sets of sample data will be entered (9 max.) ? [2] :  ');
200             readch(q);
201             IF q IN ['3'..'9'] THEN numberofsets := ord(q) - 48
202             ELSE numberofsets := 2;
203             prompt(top);
204             FOR i := 1 TO numberofsets DO
205               BEGIN
```

```
206                 prompt(1);
207                 writeln('For sample number ', i: 2);
208                 IF xyzbyhand THEN q := 'E'
209                 ELSE
210                   BEGIN
211                   writeln('use current sA(mple data, E(nter XYZ,');
212                   write('  -or-   R(etrieve file (A,E,R) [R] :  ');
213                   readch(q);
214                   ENO;
215                 IF q = 'E' THEN
216                   BEGIN
217                   writeln;
218                   readre('enter tristimulus value X :  ', x[i]);
219                   readre('enter tristimulus value Y :  ', y[i]);
220                   readre('enter tristimulus value Z :  ', z[i]);
221                   checkval(x[i], y[i], z[i]);
222                   ENO
223                 ELSE IF q <> 'A' THEN
224                   BEGIN
225                   prompt(top);
226                   write('enter file name to retrieve (xxxxxx) :  ');
227                   getfilename(fn);
228                   retrievefile(fn, sample);
229                   ENO;
230                 IF q <> 'E' THEN
231                   BEGIN
232                   chromaticity(source, sample, cx, cy, cz, scx, scy, scz, x[i], y[i], z[i], stx, sty, stz,
233                                u, v, su, sv);
234                   descrip[i] := sample.descrip;
235                   ENO
236                 ELSE
237                   BEGIN
238                   writeln;
239                   write('enter description for this sample :  ');
240                   rstring(descrip[i]);
241                   ENO;
242                 cielab(x[i], y[i], z[i], stx, sty, stz, lstar[i], astar[i], bstar[i], cab[i], hab[i],
243                       hueaberror[i]);
244                 cieluv(x[i], y[i], z[i], stx, sty, stz, uprime, vprime, suprime, svprime, lstar[i],
245                       ustar[i], vstar[i], cuv[i], huv[i], hueuverror[i]);
246                 prompt(0);
247                 writeln('Sample :  ', descrip[i]);
248                 writeln;
249                 writeln('CIELAB Coordinates');
250                 writeln('L* :  ': 10, lstar[i]: 9: 5, ' ': 10, 'a* :  ': 10, astar[i]: 9: 5, ' ': 10,
251                         'b* :  ': 10, bstar[i]: 9: 5);
252                 write('C*ab :  ': 10, cab[i]: 9: 5, ' ': 10);
253                 IF NOT hueaberror[i] THEN write('Hab :  ': 10, hab[i]: 9: 5, ' degrees');
254                 writeln;
255                 writeln;
256                 writeln('CIELUV Coordinates');
257                 writeln('L* :  ': 10, lstar[i]: 9: 5, ' ': 10, 'u* :  ': 10, ustar[i]: 9: 5, ' ': 10,
```

```
258                     'v* :  ': 10, vstar[i]: 9: 5);
259               write('C*uv :  ': 10, cuv[i]: 9: 5, ' ': 10);
260               IF NOT hueuverror[i] THEN write('Huv :  ': 10, huv[i]: 9: 5, ' degrees');
261               writeln;
262               write('press any key to continue ': 25);
263               readch(q);
264               END;
265             FOR i := 0 TO numberofsets DO spacestring(descrip[i]);
266             REPEAT
267               BEGIN
268               IF numberofsets > 2 THEN
269                 BEGIN
270                 prompt(3);
271                 writeln('This option calculates the color difference between samples.');
272                 writeln('One sample can be compared to the rest -or- all possible ');
273                 writeln('combinations can be compared. Compare O(ne sample to the rest');
274                 write('-or- A(ll possible ? (O,A) [A] :  ');
275                 readch(q);
276                 END
277               ELSE q := 'A';
278               IF q = 'O' THEN
279                 BEGIN
280                 REPEAT
281                   BEGIN
282                   prompt(O);
283                   FOR i := 1 TO numberofsets DO writeln('Sample', i: 2, ' ': 5, descrip[i]);
284                   write('select sample number that others will be compared to :  ');
285                   readch(q);
286                   IF q IN ['1'..'9'] THEN standard := ord(q) - 48
287                   ELSE standard := 0;
288                   END;
289                 UNTIL (standard >= 1) AND (standard <= numberofsets);
290                 FOR i := 1 TO numberofsets DO
291                   BEGIN
292                   deab[i] := sqrt(sqr(lstar[i] - lstar[standard]) + sqr(astar[i] - astar[standard]) +
293                                   sqr(bstar[i] - bstar[standard]));
294                   deuv[i] := sqrt(sqr(lstar[i] - lstar[standard]) + sqr(ustar[i] - ustar[standard]) +
295                                   sqr(vstar[i] - vstar[standard]));
296                   END;
297                 prompt(O);
298                 writeln('SAMPLES     STANDARD :  ', descrip[standard]: 20, 'dEab': 8, 'dEuv': 20);
299                 FOR i := 1 TO numberofsets DO
300                   BEGIN
301                   IF i <> standard THEN
302                     writeln(descrip[i]: 40, ' ': 4, deab[i]: 8: 4, ' ': 12, deuv[i]: 8: 4);
303                   END;
304                 write('P(rint these values -or- press any other key to continue (P) :  ');
305                 readch(q);
306                 IF q = 'P' THEN
307                   BEGIN
308                   rewrite(out, 'LP:OUTPUT.OUT');
309                   header(out, la100, spooled, user);
```

```
310                   writeln(out);
311                   FOR i := 1 TO numberofsets DO
312                     BEGIN
313                     pcielabluv(out, descrip[i], lstar[i], astar[i], bstar[i], cab[i], hab[i], ustar[i],
314                                vstar[i], cuv[i], huv[i], hueaberror[i], hueuverror[i]);
315                     END;
316                   writeln(out);
317                   writeln(out, ' ': 16, 'CIELab and CIELuv Color Difference Calculations');
318                   writeln(out);
319                   writeln(out, 'Source Description   :  ', descrip[0]);
320                   writeln(out);
321                   writeln(out, 'SAMPLES     STANDARD :  ', descrip[standard]: 20, 'dEab': 8, 'dEuv':
322                           20);
323                   FOR i := 1 TO numberofsets DO
324                     BEGIN
325                     IF i <> standard THEN
326                       writeln(out, descrip[i]: 40, ' ': 4, deab[i]: 8: 4, ' ': 12, deuv[i]: 8: 4);
327                     END;
328                   page(out);
329                   close(out);
330                   END;
331                 END
332               ELSE
333                 BEGIN
334                 index := 0;
335                 FOR i := 1 TO (numberofsets - 1) DO
336                   BEGIN
337                   prompt(0);
338                   writeln('SAMPLES     STANDARD :  ', descrip[i]: 20, 'dEab': 8, 'dEuv': 20);
339                   FOR j := (i + 1) TO numberofsets DO
340                     BEGIN
341                     index := index + 1;
342                     deab[index] := sqrt(sqr(lstar[i] - lstar[j]) + sqr(astar[i] - astar[j]) +
343                                         sqr(bstar[i] - bstar[j]));
344                     deuv[index] := sqrt(sqr(lstar[i] - lstar[j]) + sqr(ustar[i] - ustar[j]) +
345                                         sqr(vstar[i] - vstar[j]));
346                     writeln(descrip[j]: 40, ' ': 4, deab[index]: 8: 4, ' ': 12, deuv[index]: 8: 4);
347                     END;
348                   IF i < (numberofsets - 1) THEN
349                     BEGIN
350                     write(' ': 25, 'press any key to continue ');
351                     readch(q);
352                     END;
353                   END;
354                 write(' ': 10, 'P(rint these values -or- any other key to continue :  ');
355                 readch(q);
356                 IF q = 'P' THEN
357                   BEGIN
358                   rewrite(out, 'LP:OUTPUT.OUT');
359                   header(out, la100, spooled, user);
360                   writeln(out);
361                   FOR i := 1 TO numberofsets DO
```

```
362                     BEGIN
363                     pcielabluv(out, descrip[i], lstar[i], astar[i], bstar[i], cab[i], hab[i], ustar[i],
364                             vstar[i], cuv[i], huv[i], hueaberror[i], hueuverror[i]);
365                     END;
366                   writeln(out);
367                   writeln(out, ' ': 16, 'CIELab and CIELuv Color Difference Calculations');
368                   writeln(out);
369                   writeln(out, 'Source Description   :  ', descrip[0]);
370                   writeln(out);
371                   index := 0;
372                   FOR i := 1 TO (numberofsets - 1) DO
373                     BEGIN
374                     writeln(out, 'SAMPLES     STANDARD :  ', descrip[i]: 20, 'dEab': 8, 'dEuv': 20);
375                     FOR j := (i + 1) TO numberofsets DO
376                       BEGIN
377                       index := index + 1;
378                       writeln(out, descrip[j]: 40, ' ': 4, deab[index]: 8: 4, ' ': 12, deuv[index]: 8:
379                               4);
380                       END;
381                     writeln(out);
382                     END;
383                   page(out);
384                   close(out);
385                   END;
386                 END;
387               prompt(top);
388               q := 'Y';
389               write('Do you wish to do further calculations with this data (Y,N) [Y] :  ');
390               readch(q);
391               END;
392             UNTIL (q = 'N');
393             q := 'Y';
394             prompt(top);
395             write('Do you wish further calculations with this same illuminant ? (Y,N) [Y] :  ');
396             readch(q);
397             END;
398           UNTIL (q = 'N');
399         END;
400
401
402       PROCEDURE hueangle;
403
404         BEGIN
405           error := false;
406           h := 0.0;
407           IF x = 0.0 THEN
408             BEGIN
409             IF y > 0.0 THEN h := 1.570796327;
410             IF y < 0.0 THEN h := 4.712388981;
411             IF y = 0.0 THEN error := true;
412             END
413           ELSE IF y = 0.0 THEN
```

```
414              BEGIN
415              IF x > 0.0 THEN h := 0.0
416              ELSE h := 3.141592654;
417              END
418            ELSE
419              BEGIN
420              h := arctan(y / x);
421              IF x < 0 THEN h := h + 3.141592654
422              ELSE IF y < 0 THEN h := h + 6.283185308;
423              END;
424            h := 180.0 * h / 3.141592654;
425          END;
426
427
428        PROCEDURE cieluv;
429
430          VAR
431            q: char;
432            yratio: real;
433
434          BEGIN
435            yratio := check_ratio(ty / sty);
436            lstar := 116 * yratio - 16.0;
437            uprime := 4.0 * tx / (tx + 15.0 * ty + 3.0 * tz);
438            vprime := 9.0 * ty / (tx + 15.0 * ty + 3.0 * tz);
439            suprime := 4.0 * stx / (stx + 15.0 * sty + 3.0 * stz);
440            svprime := 9.0 * sty / (stx + 15.0 * sty + 3.0 * stz);
441            ustar := 13.0 * lstar * (uprime - suprime);
442            vstar := 13.0 * lstar * (vprime - svprime);
443            cuv := sqrt(sqr(ustar) + sqr(vstar));
444            hueangle(ustar, vstar, huv, hueangleerroruv);
445          END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr09;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR09.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR09
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12           Input IO Error Trapping/ 10-2D-84/ R.M. Miller
   13          User prompts changed/ 05-13-85/ R.M. Miller
   14
   15
   16                              BRIEF PROGRAM DESCRIPTION
   17
   18          Procedures correcting and displaying files
   19
   20          end of informational heading }
   21
   22       { ------------------------------------------------------------------------
   23        externally defined procedures that are external to the main program
   24        ------------------------------------------------------------------------ }
   25
   26        %INCLUDE 'clrpro';
2  1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2  2
2  3       {$nolist}
2 143       {$list}
   27
   28
   29         PROCEDURE correctfile(VAR enter: data);
   30           EXTERNAL;
   31
   32
   33         PROCEDURE displayfile(VAR s: data;
   34                               page: integer);
   35           EXTERNAL;
   36
   37
   38         PROCEDURE cdata(VAR enter, source, sample: data;
   39                         VAR havesource: boolean;
   40                         newfile: boolean);
   41           EXTERNAL;
   42
   43
   44         PROCEDURE displayfile;
   45
   46           VAR
   47             wave, lower, i, j: integer;
   48
   49           BEGIN
```

```
50            write(chr(27), 'H', chr(27), 'J');
51            IF page = 1 THEN lower := 11
52            ELSE lower := 1;
53            FOR i := lower TO (lower + 9) DO
54              BEGIN
55              FOR j := 0 TO 3 DO
56                BEGIN
57                wave := startingwavelength + (i + j * 20 - 1) * mininc;
58                IF wave <= endingwavelength THEN wval(wave, s.val[i + j * 20]);
59                IF j <> 3 THEN write(' ': 4)
60                ELSE writeln;
61                END;
62              END;
63          END;
64
65
66        PROCEDURE correctfile;
67
68          VAR
69            page, count, wave, xpos, ypos: integer;
70            temp: real;
71            q: char;
72
73          BEGIN
74            page := 0;
75            q := 'A';
76            REPEAT
77              BEGIN
78              IF q = 'N' THEN page := page + 1;
79              displayfile(enter, page);
80              REPEAT
81                BEGIN
82                gotoxy(0, 10);
83                writeln;
84                gotoxy(0, 10);
85                write('C(orrections needed   N(o corrections needed   R(edisplay (C,N,R) :  ');
86                readch(q);
87                gotoxy(0, 10);
88                write(chr(27), 'J');
89                gotoxy(2, 10);
90                IF q = 'C' THEN
91                  BEGIN
92                  readin('enter wavelength you wish to correct :  ', wave);
93                  write(chr(27), 'A');
94                  readre('enter correct value :  ', temp);
95                  gotoxy(0, 10);
96                  write(chr(27), 'J');
97                  IF validwave(wave) THEN
98                    BEGIN
99                    count := (wave - startingwavelength) DIV 5 + 1;
100                   enter.val[count] := temp;
101                   xpos := trunc((count - 1) / 20.0);
```

```
                    ypos := count - 20 * xpos - 1;
                    IF page = 1 THEN ypos := ypos - 10;
                    xpos := 18 * xpos;
                    IF ypos >= 0 THEN
                      BEGIN
                      gotoxy(xpos, ypos);
                      wval(wave, enter.val[count]);
                      END;
                    END
                  ELSE
                    BEGIN
                    write(wave: 3, ' is not a valid wavelength  press any key to continue :  ');
                    readch(q);
                    gotoxy(0, 10);
                    write(chr(27), 'J');
                    END;
                  END;
              END;
            UNTIL (q = 'N') OR (q = 'R');
            END;
          UNTIL (q = 'N') AND (page = 1);
        END;


      PROCEDURE cdata;

        VAR
          fn: string14;
          f: FILE OF data;
          q, qt: char;
          len: integer;

        BEGIN
          IF newfile THEN
            BEGIN
            correctfile(enter);
            savefile(enter);
            END
          ELSE
            BEGIN
            prompt(top);
            write('enter file name to correct (xxxxxx) :  ');
            getfilename(fn);
            retrievefile(fn, enter);
            correctfile(enter);
            prompt(top);
            writeln('file description :  ', enter.descrip);
            write('is this correct ? (Y,N) [Y] :  ');
            readch(qt);
            IF qt = 'N' THEN
              BEGIN
              jump(2);
```

```
154                writeln('description is limited to 40 characters');
155                write('enter correct description :  ');
156                Rstring(enter.descrip);
157                END;
158              prompt(top);
159              writeln('wavelength increment for file :  ', enter.inc: 4);
160              write('is this correct ? (Y,N) [Y] :  ');
161              readch(qt);
162              IF qt = 'N' THEN
163                BEGIN
164                jump(1);
165                readin('enter correct wavelength increment :  ', enter.inc);
166                END;
167              prompt(top);
168              write('save these changes to disk (Y,N) [Y] :  ');
169              readch(q);
170              IF q <> 'N' THEN
171                BEGIN
172                write('save to the same file name (Y,N) [Y] :  ');
173                readch(q);
174                IF q <> 'N' THEN
175                  BEGIN
176                  len := 1;
177                  rewrite(f, fn, , len);
178                  f^ := enter;
179                  put(f);
180                  close(f);
181                  END
182                ELSE savefile(enter); ;
183                END;
184              END;
185            prompt(top);
186            write('is this S(ource  -OR-  sA(mple data -OR- N(either ? (S,A,N) [A] :  ');
187            readch(qt);
188            IF qt = 'S' THEN
189              BEGIN
190              source := enter;
191              havesource := true;
192              END
193            ELSE IF qt <> 'N' THEN sample := enter;
194          END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr10;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR10.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR10
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          default system and data devices and extensions added/ 04-22-84/
   13            R.M. Miller
   14          permanet default device set for DAT:CLROPT.OPT/ 04-23-84/ R.M. Miller
   15          Input IO Error Trapping/ 10-20-84/ R.M. Miller
   16
   17
   18                              BRIEF PROGRAM DESCRIPTION
   19
   20          Procedures for getting and changing program options
   21
   22          end of informational heading }
   23
   24       { ---------------------------------------------------------------------
   25        externally defined procedures that are external to the main program
   26        ---------------------------------------------------------------------- }
   27
   28        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   29
   30
   31         PROCEDURE chgopt(VAR exitok, la100, ff, plotsok, rbok, spooled: boolean;
   32                          VAR chgoptkey: string6;
   33                          VAR defdev, defsydev, defext, defsyext: string3);
   34           EXTERNAL;
   35
   36
   37         PROCEDURE getopt(VAR exitok, la100, ff, plotsok, rbok, spooled: boolean;
   38                          VAR chgoptkey, exitkey: string6;
   39                          VAR version: real;
   40                          VAR defdev, defsydev, defext, defsyext: string3);
   41           EXTERNAL;
   42
   43
   44         PROCEDURE chgopt;
   45
   46           VAR
   47             key: string6;
   48             authorized: boolean;
   49             q: char;
```

```
 50
 51          BEGIN
 52            authorized := false;
 53            REPEAT
 54              BEGIN
 55              prompt(0);
 56              writeln('Software Options :': 35);
 57              writeln;
 58              writeln('0:  Authorization not needed to EXIT :  ', exitok);
 59              writeln('1:  DEC LA100 printer is the system printer :  ', la100);
 60              writeln('2:  The system printer is a spooled device :  ', spooled);
 61              writeln('3:  Red/Blue calculations are authorized :  ', rbok);
 62              writeln('4:  Default Devices     System: ', defsydev, '   Data: ', defdev);
 63              writeln('5:  Default Extensions  System: ', defsyext, '   Data: ', defext);
 64              writeln('6:  Printed plots are authorized :  ', plotsok);
 65              writeln('7:  Formfeeds enabled :  ', ff);
 66              writeln;
 67              IF NOT authorized THEN
 68                BEGIN
 69                write('enter authorization code to change values : ', chr(29), 'F');
 70                readst(key);
 71                writeln(chr(29), 'E');
 72                IF key = chgoptkey THEN authorized := true;
 73                END;
 74              IF authorized THEN
 75                BEGIN
 76                write('enter item number to change or Q(uit (1..7,Q) :  ');
 77                readch(q);
 78                CASE q OF
 79                  '0':
 80                    IF exitok = false THEN exitok := true
 81                    ELSE exitok := false;
 82                  '1':
 83                    IF la100 = false THEN la100 := true
 84                    ELSE la100 := false;
 85                  '2':
 86                    IF spooled = false THEN spooled := true
 87                    ELSE spooled := false;
 88                  '3':
 89                    IF rbok = false THEN rbok := true
 90                    ELSE rbok := false;
 91                  '4':
 92                    BEGIN
 93                    prompt(top);
 94                    writeln('Default Devices     System: ', defsydev, '   Data: ', defdev);
 95                    writeln;
 96                    write('Enter new default system device :  ');
 97                    readst(defsydev);
 98                    write('Enter new default data device :  ');
 99                    readst(defdev);
100                    END;
101                  '5':
```

```
102                     BEGIN
103                     prompt(top);
104                     writeln('Default Extensions  System: ', defsyext, '   Oata: ', defext);
105                     writeln;
106                     write('Enter new default system extension :  ');
107                     readst(defsyext);
108                     write('Enter new default data extension :  ');
109                     readst(defext);
110                     ENO;
111                   '6':
112                     IF plotsok = false THEN plotsok := true
113                     ELSE plotsok := false;
114                   '7':
115                     IF ff = false THEN ff := true
116                     ELSE ff := false;
117                   'Q': ;
118                   OTHERWISE
119                     BEGIN
120                     prompt(top);
121                     writeln(q: 2, ' is NOT a valid option');
122                     write('Q(uit -or- press any other key to try again (Q) :  ');
123                     readch(q);
124                     ENO;
125                 ENO;
126               ENO;
127             ENO;
128           UNTIL (q = 'Q') OR (NOT authorized);
129         ENO;
130
131
132       PROCEOURE readopt(VAR f: text;
133                         VAR value: boolean);
134
135         VAR
136           q: char;
137
138         BEGIN
139           IF NOT eof(f) THEN
140             BEGIN
141             readln(f, q);
142             value := (q = 'T') OR (q = 't');
143             ENO
144           ELSE value := false;
145         ENO;
146
147
148       PROCEOURE getopt;
149
150         VAR
151           q: char;
152           f: text;
153           len: integer;
```

```
154
155          BEGIN
156            len := 0;
157            reset(f, 'DAT:CLROPT', '.OPT', len);
158            IF len <> - 1 THEN
159              BEGIN
160              readln(f, version);
161              readln(f, chgoptkey);
162              readln(f, exitkey);
163              readln(f, defsydev);
164              readln(f, defdev);
165              readln(f, defsyext);
166              readln(f, defext);
167              readopt(f, exitok);
168              readopt(f, la100);
169              readopt(f, ff);
170              readopt(f, plotsok);
171              readopt(f, rbok);
172              readopt(f, spooled);
173              close(f);
174              END
175            ELSE
176              BEGIN
177
178              { if the options file is not available, use the default values      }
179
180              version := 3.0;
181              chgoptkey := 'XXXXXX';
182              exitkey := 'EXITOK';
183              defsydev := 'DK0';
184              defdev := 'DK0';
185              defsyext := 'DAT';
186              defext := 'DAT';
187              exitok := true;
188              la100 := true;
189              ff := true;
190              plotsok := true;
191              rbok := true;
192              spooled := true;
193              END;
194          END;

*** No lines with errors detected ***
```

```
    1       PROGRAM colr11;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR11.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR11
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12           Radiometer support removed from procedure/ 04-22-84/ R.M. Miller
   13          Input IO Error Trapping/ 10-20-84/ R.M. Miller
   14          User prompts changed/ 05-13-85/ R.M. Miller
   15
   16
   17
   18                              BRIEF PROGRAM DESCRIPTION
   19
   20          Procedures for entering files
   21
   22          end of informational heading }
   23
   24       { ------------------------------------------------------------------
   25        externally defined procedures that are external to the main program
   26        ------------------------------------------------------------------ }
   27
   28        %INCLUDE 'clrpro';
2  1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2  2
2  3       {$nolist}
2 143       {$list}
   29
   30
   31         PROCEOURE enterfile(VAR enter: data);
   32           EXTERNAL;
   33
   34
   35         PROCEDURE enterfile;
   36
   37           VAR
   38             start, count, wave, xpos, ypos, page, i, j, tempwave: integer;
   39             temp: PACKED ARRAY [1..26] OF char;
   40             mult, value: real;
   41             q, qt: char;
   42
   43           BEGIN
   44             REPEAT
   45               BEGIN
   46               prompt(2);
   47               writeln('valid wavelenth increments : 5, 10, 15, 20, 25, 30, 35, 40, 45, 50');
   48               writeln;
   49               readin('select wavelength increment :  ', enter.inc);
```

```
50              END;
51            UNTIL enter.inc IN [5, 10, 15, 20, 25, 30, 35, 40, 45, 50];
52            FOR i := 1 TO numberofwavelengths DO enter.val[i] := 0;
53            jump(1);
54            writeln('the description is limited to 40 characters');
55            write('enter description :  ');
56            Rstring(enter.descrip);
57            prompt(top);
58            writeln('the default starting wavelength is 380 nm');
59            writeln;
60            write('C(hange the starting value or any other key to continue (C) :  ');
61            readch(q);
62            IF q = 'C' THEN
63              BEGIN
64              writeln;
65              readin('enter starting wavelength (must be >= 380) :  ', start);
66              WHILE NOT validwave(start) DO
67                BEGIN
68                prompt(top);
69                writeln(start: 3, ' is not a valid wavelength');
70                jump(2);
71                writeln('the range is from ', startingwavelength: 3, ' nm to ', endingwavelength: 3,
72                        ' nm');
73                jump(2);
74                readin('enter starting wavelength :  ', start);
75                ENO;
76              END
77            ELSE start := startingwavelength;
78            xpos := 0;
79            ypos := 0;
80            wave := start;
81            page := 0;
82            REPEAT
83              BEGIN
84              IF (xpos = 0) AND (ypos = 0) THEN prompt(1);
85              count := (wave - startingwavelength) DIV 5 + 1;
86              gotoxy(0, 10);
87              write(chr(27), 'J');
88              gotoxy(15, 10);
89              temp := 'enter value for     nm :  ';
90              tempwave := wave DIV 100;
91              temp[17] := chr(48 + tempwave);
92              tempwave := wave - tempwave * 100;
93              temp[18] := chr(48 + (tempwave DIV 10));
94              temp[19] := chr(48 + (tempwave MOD 10));
95              readre(temp, enter.val[count]);
96              gotoxy(xpos, ypos + 2);
97              wval(wave, enter.val[count]);
98              wave := wave + enter.inc;
99              xpos := xpos + 20;
100             IF xpos > 60 THEN
101               BEGIN
```

```
  102                xpos := 0;
  103                ypos := ypos + 1;
  104                IF ypos > 6 THEN
  105                  BEGIN
  106                  ypos := 0;
  107                  page := page + 1;
  108                  END;
  109                END;
  110              END;
  111            UNTIL wave > endingwavelength;
  112          END;
```

*** No lines with errors detected ***

```
    1       PROGRAM colr12;
    2
    3        {$nomain}
    4        { This code is for a sub program
    5          Sub-Program File Name :  COLR12.PAS
    6          Other subprograms needed :
    7          Compiling Command : pas COLR12
    8          Linking Command :  @colr.lnk
    9          Hardware Required :  none
   10
   11       Software Modifications/ Date/ Programmer
   12          Input IO Error Trapping/ 10-20-84/ R.M. Miller
   13
   14
   15                              BRIEF PROGRAM DESCRIPTION
   16
   17          Procedures for multiplying, dividing, et. files
   18
   19          end of informational heading }
   20
   21       { ------------------------------------------------------------------
   22        externally defined procedures that are external to the main program
   23        ------------------------------------------------------------------ }
   24
   25        %INCLUDE 'clrpro';
2    1       { HEADER FILE: CLRPRO.PAS containing definitions from COLR.PAS }
2    2
2    3       {$nolist}
2  143       {$list}
   26
   27
   28         PROCEDURE multfile(VAR s1, s2, s3: data;
   29                            d: integer);
   30           EXTERNAL;
   31
   32
   33         PROCEDURE mfile(VAR enter, source, sample: data;
   34                         VAR havesource: boolean;
   35                         defsydev, defsyext: string3);
   36           EXTERNAL;
   37
   38
   39         FUNCTION density(VAR r1: real): real;
   40           EXTERNAL;
   41
   42
   43         FUNCTION density;
   44
   45           BEGIN
   46             IF r1 = 0.0 THEN density := 0.0
   47             ELSE density := - 1.0 * ln(r1) / ln(10.0);
   48           END;
   49
```

```
 50
 51       PROCEDURE multfile;
 52        {  case d of        0:  s3:=s1*s2;                              }
 53        {                   1:  s3:=s1/s2;                              }
 54        {                   2:  s3:=s2/s1;                              }
 55
 56         VAR
 57           i1: integer;
 58           q: char;
 59
 60         BEGIN
 61           FOR i1 := 1 TO noofwavelengths DO
 62             BEGIN
 63     { case d of        0:  s3:=s1*s2;
 64                        1:  s3:=s1/s2;
 65                        2:  s3:=s2/s1;
 66                        3:  s3:=s1+s2;
 67                        4:  s3:=s1-s2;
 68                        5:  s3:=s2-s1;                              }
 69             CASE d OF
 70               0: s3.val[i1] := s2.val[i1] * s1.val[i1];
 71               1:
 72                 BEGIN
 73                 IF s2.val[i1] = 0.0 THEN s3.val[i1] := 0.0
 74                 ELSE s3.val[i1] := s1.val[i1] / s2.val[i1];
 75                 END;
 76               2:
 77                 BEGIN
 78                 IF s1.val[i1] = 0.0 THEN s3.val[i1] := 0.0
 79                 ELSE s3.val[i1] := s2.val[i1] / s1.val[i1];
 80                 END;
 81               3: s3.val[i1] := s2.val[i1] + s1.val[i1];
 82               4: s3.val[i1] := s1.val[i1] - s2.val[i1];
 83               5: s3.val[i1] := s2.val[i1] - s1.val[i1];
 84               END;
 85             END;
 86           IF s1.inc >= s2.inc THEN s3.inc := s1.inc
 87           ELSE s3.inc := s2.inc;
 88           writeln;
 89           writeln('1:   first file description :  ', s1.descrip);
 90           writeln('2:  second file description :  ', s2.descrip);
 91           writeln;
 92           write('Select description 1,2, or E(nter new description (1,2,E) [E] :  ');
 93           readch(q);
 94           IF q = '1' THEN s3.descrip := s1.descrip
 95           ELSE IF q = '2' THEN s3.descrip := s2.descrip
 96           ELSE
 97             BEGIN
 98             write('enter 40 character description for new file :  ');
 99             Rstring(s3.descrip);
100             END;
101         END;
```

```
102
103
104         PROCEDURE mfile;
105
106           VAR
107             m1, m2: data;
108             i1: integer;
109             k: real;
110             fn1, fn2: string14;
111             q, qt: char;
112
113           BEGIN
114             prompt(top);
115             writeln('              File Maintenance Menu');
116             writeln;
117             writeln('M: multiply,divide, add, or subtract two data files');
118             writeln('D: calculate densities of one file');
119             writeln('C: convert data collected on radiometer');
120             writeln;
121             write('Enter option (M,D,C) [M] :  ');
122             readch(q);
123             prompt(top);
124             CASE q OF
125               'D':
126                 BEGIN
127                 write('enter file name for density calculations (xxxxxx) :  ');
128                 getfilename(fn1);
129                 retrievefile(fn1, m2);
130                 FOR i1 := 1 TO noofwavelengths DO m2.val[i1] := density(m2.val[i1]);
131                 m1.descrip := 'Spectral Densities of Sample           ';
132                 FOR i1 := 31 TO stringmax DO m1.descrip[i1] := m2.descrip[i1 - 30];
133                 writeln('original file description :  ', m2.descrip);
134                 write('Select O(riginal description or E(nter new description (O,E) [E] :  ');
135                 readch(qt);
136                 IF qt <> 'O' THEN
137                   BEGIN
138                   write('enter 40 character description for new file :  ');
139                   Rstring(m2.descrip);
140                   END;
141                 END;
142               'C':
143                 BEGIN
144                 writeln('Data Conversion Options');
145                 writeln;
146                 writeln('M: Convert data collected on the Munsell Spectroradiometer');
147                 writeln('E: Convert data collected on the EG&G Spectroradiometer');
148                 writeln('N: Convert data collected on another spectroradiometer');
149                 writeln;
150                 write('Select one of the above (M,E,N) [M] :  ');
151                 readch(q);
152                 IF q = 'E' THEN concat(fn1, defsydev, 'EGGCAL', defsyext)
153                 ELSE IF q = 'N' THEN
```

```
154                 BEGIN
155                 write('enter filename of Radiometer Conversion Values (xxxxxx) :  ');
156                 getfilename(fn1);
157                 END
158               ELSE concat(fn1, defsydev, 'CONVRT', defsyext);
159               retrievefile(fn1, m1);
160               prompt(top);
161               write('enter filename of the data to convert (xxxxxx) :  ');
162               getfilename(fn2);
163               retrievefile(fn2, m2);
164               multfile(m1, m2, m2, 0);
165               END;
166             OTHERWISE
167               BEGIN
168               write('enter first file name for file manipulation (xxxxxx) :  ');
169               getfilename(fn1);
170               retrievefile(fn1, m1);
171               write('enter second file name for file manipulation (xxxxxx) :  ');
172               getfilename(fn2);
173               retrievefile(fn2, m2);
174               prompt(2);
175
176     { case d of       0:  s3:=s1*s2;
177                       1:  s3:=s1/s2;
178                       2:  s3:=s2/s1;
179                       3:  s3:=s1+s2;
180                       4:  s3:=s1-s2;
181                       5:  s3:=s2-s1;                              }
182               REPEAT
183                 BEGIN
184                 writeln('Select from the following options :');
185                 writeln;
186                 writeln(' 0:  ': 10, 'Multiply ', fn1, ' by ', fn2);
187                 writeln(' 1:  ': 10, 'Divide ', fn1, ' by ', fn2);
188                 writeln(' 2:  ': 10, 'Divide ', fn2, ' by ', fn1);
189                 writeln(' 3:  ': 10, 'Add ', fn1, ' to ', fn2);
190                 writeln(' 4:  ': 10, 'Subtract ', fn2, ' from ', fn1);
191                 writeln(' 5:  ': 10, 'Subtract ', fn1, ' from ', fn2);
192                 writeln;
193                 write('enter 0..5 :  ');
194                 readch(q);
195                 END;
196               UNTIL q IN ['0'..'5'];
197               i1 := ord(q) - 48;
198               multfile(m1, m2, m2, i1);
199               prompt(top);
200               write('do you wish to multiply this data by a constant value (Y,N) [N] ? :  ');
201               readch(q);
202               IF q = 'Y' THEN
203                 BEGIN
204                 writeln;
205                 writeln;
```

```
206                 readre('enter constant value :  ', k);
207                 FOR i1 := 1 TO noofwavelengths DO m2.val[i1] := m2.val[i1] * k;
208                 END;
209               END;
210             END;
211           prompt(top);
212           write('save these changes to disk (Y,N) [Y] :  ');
213           readch(q);
214           IF q <> 'N' THEN
215             BEGIN
216             enter := m2;
217             savefile(enter); ;
218             END;
219           prompt(top);
220           write('is this S(ource  -OR-  sA(mple data -OR- N(either ? (S,A,N) [A] :  ');
221           readch(q);
222           IF q = 'S' THEN
223             BEGIN
224             source := m2;
225             havesource := true;
226             END
227           ELSE IF q <> 'N' THEN sample := m2;
228         END;
```

*** No lines with errors detected ***

APPENDIX E

COLOR PROGRAM

The new color program permits a user to enter, correct, display, plot, retrieve, and save spectral data. The program also provides color calculations, spectra manipulation with several mathematical functions, calculation of color rendering index, calculation of ASTM type weights from light source data, taut cubic spline interpolation of x,y data, and the ability to manually enter the color coordinates of samples (xyY, XYZ, CIELAB, CIELUV, uv, and u'v') for color difference calculations. There is also a feature that permits a FORTRAN subroutine to be linked with the main program for custom color difference equations. The program consists of the following modules:

E1. COLOR.PAS: the main program.

E2. SPECTRA.HDR: a library of spectra handling routines.

E3. MILPK4.HDR: a library of user interface routines.

E4. SPLINE.HDR: taut cubic spline and interpolation routines.

```
1       PROGRAM COLOR;
2
3         CONST
4           { name of program that will be displayed at the top of the screen }
5           program_name = 'Color Program V1.0';
6           { when read from the keyboard, this character causes a goto 1 }
7           abort_char = '?';
8           { should program terminate on ctrl-Z from the keyboard ? }
9           stop_on_eof = true;
10          { should single character reads be used in this program }
11          singlechar = true;
12
13          { include the user i/o routines }
14          {$nolist}
16          {$list}
17
18          { include the spectra handling routines }
19          {$nolist}
21          {$list}
22
23        TYPE
24          two_show_coordinate_data = ARRAY [1..2] OF show_coordinate_data;
25
26          program_data =
27            RECORD
28                              { weights
29                                 1..3 hold weights for 5 nm,
30                                 4..6 hold weights for 10 nm,
31                                and 7..9 hold weights for 20 nm }
32              weights: ARRAY [1..9] OF spectral_data;
33              samples: ARRAY [1..10] OF spectral_data;
34              sources: ARRAY [1..10] OF spectral_data;
35                              { observer
36                                 1..3 hold 5 nm observer functions }
37              observer: ARRAY [1..3] OF spectral_data;
38              file_math_temps: ARRAY [1..10] OF spectral_data;
39              illuminant: ARRAY [1..1] OF spectral_data;
40                              { SOURCE
41                                 holds a spectral array of a source to use for color calculations }
42              source: ARRAY [1..1] OF spectral_data;
43                              { CMF
44                                 1..3 hold 1 nm color matching functions, 4..5 hold spectral chromaticities }
45              cmf: ARRAY [1..5] OF big_spectral_data;
46
47              observer_name: STRING [5];
48              illuminant_name: STRING [20];
49              use_illuminant_for_color_calc: boolean;
50              main_q: char;
51                              { show_color_set
52                                  1 holds values that control the display of color coordinates
53                                  2 holds values that control the display of color differences }
54              show_color_set: two_show_coordinate_data;
55              { default name of file or device for reports }
56              default_output_filename: fn;
57              file_io_set: file_io_set_type;
```

```
58            END;
59
60          saved_data_type = (variable_data, spectral);
61
62          saved_data_record =
63            RECORD
64              CASE data_type: saved_data_type OF
65                variable_data:
66                  (observer_name: STRING [5];
67                   illuminant_name: STRING [20];
68                   use_illuminant_for_color_calc: boolean;
69                   show_color_set: two_show_coordinate_data;
70                   default_output_filename: fn);
71                spectral:
72                  (spec: spectral_data);
73            END;
74
75        VAR
76          data: program_data;
77
78
79        PROCEDURE strip_trailing_blanks(VAR s: PACKED ARRAY [low..high: integer] OF char);
80      {
81       WHERE
82           s = string to have trailing blanks stripped off
83       RESULTS
84           s = string without trailing blanks
85      }
86
87          VAR
88            i: integer;
89
90          BEGIN
91            IF low = 0 THEN
92              BEGIN
93              i := ord(s[0]);
94              IF i > high THEN
95                i := high;
96              END
97            ELSE IF low = 1 THEN
98              i := high;
99            IF low IN [0, 1] THEN
100             BEGIN
101             i := i + 1;
102             REPEAT
103               i := i - 1;
104             UNTIL (i = 1) OR (s[i] <> ' ');
105             IF s[i] = ' ' THEN
106               i := i - 1;
107             IF low = 0 THEN
108               s[0] := chr(i);
109             FOR i := i + 1 TO high DO
110               s[i] := chr(0);
111             END;
```

```
112         END;
113
114
115       PROCEDURE load_weights_cmf_illuminants(VAR data: program_data);
116     {
117      WHERE
118          data = main program data record
119      RESULTS
120          Weights, color matching functions, and an illuminant are loaded if the selected ones are available
121     }
122
123         VAR
124           error: boolean;
125           i, j, error_code: integer;
126           temp_cmf: ARRAY [1..3] OF big_spectral_data;
127           r: real;
128           filename: fn;
129           name: STRING [70];
130
131         BEGIN
132           prompt(6);
133           writec('please wait while weights, observer, and illuminant');
134           writec('spectra are read from disk');
135           error := false;
136           WITH data DO
137             BEGIN
138             { read observer functions }
139             name := 'CLR:CMF.' + observer_name + '_5NM';
140             stass(filename, name);
141             strip_trailing_blanks(filename);
142             observer[1].filename := filename;
143             retrieve_mspectra(observer, 1, 2, 3, error_code);
144             error := error_code <> 0;
145             IF error_code <> 0 THEN
146               BEGIN
147               writeln('Error loading 5 NM CMF named :  "', observer[1].filename, '"  CODE = ', error_code: 0);
148               FOR i := 1 TO 3 DO
149                 zero_spectra(observer[i]);
150               END;
151
152             name := 'CLR:CMF.' + observer_name + '_1NM';
153             stass(filename, name);
154             strip_trailing_blanks(filename);
155             temp_cmf[1].filename := filename;
156             retrieve_big_mspectra(temp_cmf, error_code);
157             FOR i := 1 TO 3 DO
158               cmf[i] := temp_cmf[i];
159             error := error OR (error_code <> 0);
160             IF error_code <> 0 THEN
161               BEGIN
162               writeln('Error loading 1 NM CMF named :  "', cmf[1].filename, '"  CODE = ', error_code: 0);
163               FOR i := 1 TO 5 DO
164                 FOR j := 360 TO 830 DO
165                   cmf[i].val[j] := 0;
```

```
166               END
167             ELSE
168               FOR j := 360 TO 830 DO
169                 BEGIN
170                 r := 0;
171                 FOR i := 1 TO 3 DO
172                   r := r + cmf[i].val[j];
173                 FOR i := 4 TO 5 DO
174                   cmf[i].val[j] := cmf[i - 3].val[j] / r;
175                 END;
176
177             { read illuminant }
178             name := 'CLR:' + illuminant_name;
179             stass(filename, name);
180             strip_trailing_blanks(filename);
181             illuminant[1].filename := filename;
182             retrieve_mspectra(illuminant, 1, 1, 1, error_code);
183             error := error OR (error_code <> 0);
184             IF error_code <> 0 THEN
185               BEGIN
186               writeln('Error loading illuminant named :  "', illuminant[1].filename, '"  CODE = ', error_code: 0);
187               zero_spectra(illuminant[1]);
188               END;
189
190             { read 5 nm weights }
191             name := 'CLR:' + illuminant_name + '.' + observer_name + '_5NM_WEIGHTS';
192             stass(filename, name);
193             strip_trailing_blanks(filename);
194             weights[1].filename := filename;
195             retrieve_mspectra(weights, 1, 1, 3, error_code);
196             error := error OR (error_code <> 0);
197             IF error_code <> 0 THEN
198               BEGIN
199               writeln('Error loading  5 NM weights named :  "', weights[1].filename, '"  CODE = ', error_code: 0);
200               FOR i := 1 TO 3 DO
201                 zero_spectra(weights[i]);
202               END;
203
204             { read 10 nm weights }
205             name := 'CLR:' + illuminant_name + '.' + observer_name + '_10NM_WEIGHTS';
206             stass(filename, name);
207             strip_trailing_blanks(filename);
208             weights[4].filename := filename;
209             retrieve_mspectra(weights, 4, 1, 3, error_code);
210             error := error OR (error_code <> 0);
211             IF error_code <> 0 THEN
212               BEGIN
213               writeln('Error loading 10 NM weights named :  "', weights[4].filename, '"  CODE = ', error_code: 0);
214               FOR i := 4 TO 6 DO
215                 zero_spectra(weights[i]);
216               END;
217
218             { read 20 nm weights }
219             name := 'CLR:' + illuminant_name + '.' + observer_name + '_20NM_WEIGHTS';
```

```
220             stass(filename, name);
221             strip_trailing_blanks(filename);
222             weights[7].filename := filename;
223             retrieve_mspectra(weights, 7, 1, 3, error_code);
224             error := error OR (error_code <> 0);
225             IF error_code <> 0 THEN
226               BEGIN
227               writeln('Error loading 2D NM weights named :  "', weights[7].filename, '"  CODE = ', error_code: 0);
228               FOR i := 7 TO 9 DO
229                 zero_spectra(weights[i]);
230               END;
231
232             IF error THEN
233               BEGIN
234               jump(2);
235               writec('ERRORS WERE ENCOUNTERED WHILE READING DATA FILES');
236               writec('check the WEIGHTS menu to see what files loaded');
237               return(2);
238               END;
239             END;
240         END;
241
242
243       PROCEDURE init_data(VAR data: program_data);
244       {
245        WHERE
246            data = main program data record
247        RESULTS
248            main program data record is zeroed
249       }
250
251           VAR
252             i: integer;
253             sh: show_coordinate_type;
254
255           BEGIN
256             WITH data DO
257               BEGIN
258               FOR i := 1 TO 9 DO
259                 zero_spectra(weights[i]);
260               FOR i := 1 TO 10 DO
261                 zero_spectra(samples[i]);
262               FOR i := 1 TO 10 DO
263                 zero_spectra(sources[i]);
264               FOR i := 1 TO 3 DO
265                 zero_spectra(observer[i]);
266               FOR i := 1 TO 10 DO
267                 zero_spectra(file_math_temps[i]);
268               zero_spectra(illuminant[1]);
269               zero_spectra(source[1]);
270               main_q := 'S';
271               use_illuminant_for_color_calc := true;
272               observer_name := '02DEG';
273               illuminant_name := 'D65';
```

```
274             stass(default_output_filename, 'color_report.lis');
275             strip_trailing_blanks(default_output_filename);
276             FOR i := 1 TO 2 DO
277               FOR sh := tristimulus TO cieluv_2 DO
278                 show_color_set[i][sh] := 1;
279             FOR i := 1 TO 2 DO
280               FOR sh := special_1 TO special_5 DO
281                 show_color_set[i][sh] := 0;
282             file_io_set_init(file_io_set);
283             END;
284           load_weights_cmf_illuminants(data);
285         END;
286
287
288       PROCEDURE save_data(s: PACKED ARRAY [low..high: integer] OF char;
289                           VAR data: program_data);
290     {
291      WHERE
292          s = filename to write main program data record to
293          data = main program data record
294      RESULTS
295          main program data record is saved to the disk file
296     }
297
298         VAR
299           f: FILE OF saved_data_record;
300           ss: PACKED ARRAY [1..70] OF char;
301           i: integer;
302           temp: saved_data_record;
303
304         BEGIN
305           strass(ss, s);
306           i := 0;
307           rewrite(f, ss, 'color_program_saved_variables.dat', i);
308           IF i <> -1 THEN
309             BEGIN
310             WITH data DO
311               BEGIN
312
313               { write program variables that should be saved }
314               temp.data_type := variable_data;
315               temp.observer_name := observer_name;
316               temp.illuminant_name := illuminant_name;
317               temp.use_illuminant_for_color_calc := use_illuminant_for_color_calc;
318               temp.show_color_set := show_color_set;
319               temp.default_output_filename := default_output_filename;
320               write(f, temp);
321
322               { write spectral data that should be saved }
323               temp.data_type := spectral;
324
325               FOR i := 1 TO 10 DO
326                 BEGIN
327                 temp.spec := samples[i];
```

```
328                 write(f, temp);
329                 END;
330               FOR i := 1 TO 10 DO
331                 BEGIN
332                 temp.spec := sources[i];
333                 write(f, temp);
334                 END;
335               FOR i := 1 TO 10 DO
336                 BEGIN
337                 temp.spec := file_math_temps[i];
338                 write(f, temp);
339                 END;
340               temp.spec := source[1];
341               write(f, temp);
342               END;
343             close(f);
344             END
345           ELSE
346             BEGIN
347             prompt(10);
348             writec('Error Writing Program Variables File');
349             jump(2);
350             writec('Variables NOT Saved');
351             return(2);
352             END;
353         END;
354
355
356       PROCEDURE read_data(s: PACKED ARRAY [low..high: integer] OF char;
357                           VAR data: program_data;
358                           init_on_error: boolean);
359       {
360        WHERE
361            s = filename of main program data record that had been saved to disk
362            data = main program data record
363            init_on_error = TRUE to zero main program data record if selected file is not available
364        RESULTS
365            main program data record is loaded from the file, or zeroed if the file is not available (and init_on_error is TRUE)
366       }
367
368           VAR
369             f: FILE OF saved_data_record;
370             ss: PACKED ARRAY [1..70] OF char;
371             i: integer;
372             temp: saved_data_record;
373
374           BEGIN
375             strass(ss, s);
376             reset(f, ss, 'color_program_saved_variables.dat', i);
377             IF i <> - 1 THEN
378               BEGIN
379               WITH data DO
380                 BEGIN
381                 { read program variables that should be saved }
```

```
382               read(f, temp);
383               observer_name := temp.observer_name;
384               illuminant_name := temp.illuminant_name;
385               use_illuminant_for_color_calc := temp.use_illuminant_for_color_calc;
386               show_color_set := temp.show_color_set;
387               default_output_filename := temp.default_output_filename;
388
389               { read spectral data that should be saved }
390
391               FOR i := 1 TO 10 DO
392                 BEGIN
393                 read(f, temp);
394                 samples[i] := temp.spec;
395                 END;
396               FOR i := 1 TO 10 DO
397                 BEGIN
398                 read(f, temp);
399                 sources[i] := temp.spec;
400                 END;
401               FOR i := 1 TO 10 DO
402                 BEGIN
403                 read(f, temp);
404                 file_math_temps[i] := temp.spec;
405                 END;
406               read(f, temp);
407               source[1] := temp.spec;
408               END;
409             close(f);
410             load_weights_cmf_illuminants(data);
411             file_io_set_init(data.file_io_set);
412             data.main_q := 'S';
413             END
414           ELSE
415             BEGIN
416             prompt(10);
417             writec('Error Reading Program Variables File');
418             jump(2);
419             writec('Variables NOT Read From File');
420             IF init_on_error THEN
421               BEGIN
422               jump(2);
423               init_data(data);
424               writec('Variables Initialized to Default Values');
425               END;
426             return(2);
427             END;
428         END;
429
430
431       PROCEDURE ask_to_make_report(choice: char;
432                                    VAR out: text;
433                                    default_output_filename: fn;
434                                    VAR make_report: boolean);
435     {
```

```
436       WHERE
437           choice = character containing default response to report prompt
438           out = text file variable for report
439           default_output_filename = default filename for report
440       RESULTS
441           make_report = TRUE if a report was selected
442       }
443
444           VAR
445             i: integer;
446
447           BEGIN
448             IF NOT (choice IN ['Y', 'N']) THEN
449               choice := 'N';
450             jump(1);
451             getchar('send a R(eport to a file or printer ? (Y,N)', choice, use_default, ['Y', 'N']);
452             make_report := choice = 'Y';
453             IF make_report THEN
454               BEGIN
455               jump(1);
456               i := 0;
457               REPEAT
458                 IF i = - 1 THEN
459                   writec('ERROR opening file or device... please try again');
460                 readst('enter filename or device or report', default_output_filename, use_default, null_ok);
461                 i := 0;
462                 rewrite(out, default_output_filename, 'default_report.lis', i);
463               UNTIL i <> - 1;
464               END;
465           END;
466
467
468         PROCEDURE weights_menu(VAR data: program_data);
469       {
470       WHERE
471           data = main program data record
472       RESULTS
473           weights arrays may be manipulated
474       }
475
476           VAR
477             q, q1: char;
478             i: integer;
479             extra_info: STRING [10];
480             temp_file_io_set: file_io_set_type;
481
482           BEGIN
483             WITH data DO
484               BEGIN
485
486           { disable the READ, ENTER, CORRECT, and SAVE SPECTRA options since
487               - weights are read by another routine
488               - users should not routine need such options for weights  }
489               temp_file_io_set := file_io_set - ['R', 'E', 'C', 'S'];
```

```
490
491             REPEAT
492               prompt(1);
493               writec('W E I G H T S   M E N U');
494               jump(1);
495               writeln('     OBSERVER :  ', observer_name, ' selected for loading');
496               writeln('   ILLUMINANT :  ', illuminant_name, ' selected for loading');
497               writec('---------------------------------------');
498               writec('S P E C T R A   L O A D E D');
499               write(' 5 NM WEIGHTS :  ');
500               IF weights[1].start = 0 THEN
501                 writeln('  <empty>')
502               ELSE
503                 writeln(weights[1].descrip);
504               write('10 NM WEIGHTS :  ');
505               IF weights[4].start = 0 THEN
506                 writeln('  <empty>')
507               ELSE
508                 writeln(weights[4].descrip);
509               write('20 NM WEIGHTS :  ');
510               IF weights[7].start = 0 THEN
511                 writeln('  <empty>')
512               ELSE
513                 writeln(weights[7].descrip);
514               write('     1 NM CMF :  ');
515               IF cmf[1].val[550] < 0.01 THEN
516                 writeln('  <empty>')
517               ELSE
518                 writeln(cmf[1].descrip);
519               write('     5 NM CMF :  ');
520               IF observer[1].start = 0 THEN
521                 writeln('  <empty>')
522               ELSE
523                 writeln(observer[1].descrip);
524               write('   ILLUMINANT :  ');
525               IF illuminant[1].start = 0 THEN
526                 writeln('  <empty>')
527               ELSE
528                 writeln(illuminant[1].descrip);
529               writeln;
530               writeln;
531               writec('D(isplay or  P(lot weights and cmfs');
532               writec('-----------------------------------------------');
533               writec('S(elect a new illuminant or observer function');
534               jump(1);
535               getchar('select one of the above -or- Q(uit to the main menu', q, no_default, ['D', 'P', 'S', 'Q']);
536               CASE q OF
537                 'D', 'P':
538                   BEGIN
539                   gotoxy(0, 14);
540                   erase(eoscreen);
541                   writec('0:  5 nm Weights,  1:  10 nm weights,  2:  20 nm weights');
542                   writec('3:  CMFs           4:  Illuminant     ');
543                   jump(1);
```

```
544                     getchar('select one of the above', q1, no_default, ['0'..'4']);
545                     CASE q OF
546                       '0':
547                         CASE q1 OF
548                           '0':
549                             display_mspectra('DISPLAY 5 NM WEIGHTS', weights, 1, 3);
550                           '1':
551                             display_mspectra('DISPLAY 10 NM WEIGHTS', weights, 4, 3);
552                           '2':
553                             display_mspectra('DISPLAY 20 NM WEIGHTS', weights, 7, 3);
554                           '3':
555                             display_mspectra('DISPLAY CMF', observer, 1, 3);
556                           '4':
557                             display_mspectra('DISPLAY ILLUMINANT', illuminant, 1, 1);
558                         END;
559                       'P':
560                         CASE q1 OF
561                           '0':
562                             plotrg_mspectra('PLOT 5 NM WEIGHTS', weights, 1, 3);
563                           '1':
564                             plotrg_mspectra('PLOT 10 NM WEIGHTS', weights, 4, 3);
565                           '2':
566                             plotrg_mspectra('PLOT 20 NM WEIGHTS', weights, 7, 3);
567                           '3':
568                             plotrg_mspectra('PLOT CMF', observer, 1, 3);
569                           '4':
570                             plotrg_mspectra('PLOT ILLUMINANT', illuminant, 1, 1);
571                         END;
572                     END;
573                   END;
574                 'S':
575                   BEGIN
576                   gotoxy(0, 14);
577                   erase(eoscreen);
578                   writec('SELECT NEW ILLUMINANT AND OBSERVER');
579                   jump(1);
580                   writec('Selection of 2 or 10 degree observer');
581                   IF observer_name = '02DEG' THEN
582                     q1 := 'Y'
583                   ELSE
584                     q1 := 'N';
585                   getchar('use the 2 degree observer ? (Y,N)', q1, use_default, ['Y', 'N']);
586                   IF q1 = 'Y' THEN
587                     observer_name := '02DEG'
588                   ELSE
589                     observer_name := '10DEG';
590                   writec('select one of the following illuminants:');
591                   getinf('clr:illuminants.inf', illuminant_name, extra_info, use_default);
592                   strip_trailing_blanks(illuminant_name);
593                   prompt(6);
594                   load_weights_cmf_illuminants(data);
595                   END;
596                 'Q': ;
597               END;
```

```
598             UNTIL q = 'Q';
599             END;
600         END;
601
602
603       PROCEDURE samples_menu(VAR data: program_data);
604     {
605      WHERE
606          data = main program data record
607      RESULTS
608          samples arrays may be manipulated
609     }
610
611         VAR
612           q, q1: char;
613           i, j, k, num, deg: integer;
614           out: text;
615           temp_source: spectral_data;
616           temp1, temp2, source_coordinates: coordinate_data;
617           make_report, coordinates_entered, error: boolean;
618           ss: STRING [78];
619
620
621         PROCEDURE get_source_coordinates;
622
623           VAR
624             i: integer;
625
626           BEGIN
627             WITH data DO
628               BEGIN
629               { find coordinate of illuminant or source used by first getting coordinates of weights, then a source or illumi
630               IF use_illuminant_for_color_calc THEN
631                 BEGIN
632                 calc_color_xyz_weights(weights, error);
633                 IF NOT error THEN
634                   BEGIN
635                   i := 1 - 3;
636                   REPEAT
637                     i := i + 3;
638                   UNTIL (weights[i].start > 100) OR (i = 7);
639                   error := weights[i].start <= 100;
640                   IF NOT error THEN
641                     source_coordinates := weights[i].coordinates;
642                   END;
643                 IF error THEN
644                   BEGIN
645                   calc_color_xyz_source(illuminant[1], observer, error);
646                   IF NOT error THEN
647                     source_coordinates := illuminant[1].coordinates;
648                   END;
649                 END
650               ELSE
651                 BEGIN
```

```
652                 IF source[1].start = 1 THEN
653                   source_coordinates := source[1].coordinates
654                 ELSE
655                   BEGIN
656                   calc_color_xyz_source(source[1], observer, error);
657                   IF NOT error THEN
658                     source_coordinates := source[1].coordinates;
659                   END;
660                 END;
661               END;
662             IF NOT error THEN
663               calc_color_chromaticities(source_coordinates, error);
664           END;
665
666
667         PROCEDURE get_sample_coordinates(VAR sample: spectral_data;
668                                          VAR error: boolean);
669       {
670        WHERE
671            sample = spectral data record for a sample
672        RESULTS
673            sample = color coordiantes are calculated for a sample if they were not entered by hand
674            error = signals an error condition
675       }
676
677           VAR
678             i: integer;
679
680           BEGIN
681             error := false;
682             WITH data DO
683               BEGIN
684               IF sample.start <> 1 THEN { make sure they weren't entered by hand }
685                 BEGIN
686                 { find coordinates of a sample }
687                 IF use_illuminant_for_color_calc THEN
688                   BEGIN
689                   calc_color_xyz_weights_sample(weights, sample, error);
690                   IF error THEN
691                     calc_color_xyz_source_sample(illuminant[1], sample, observer, error);
692                   END
693                 ELSE
694                   BEGIN
695                   error := true;
696                   IF source[1].start <> 1 THEN { if they've not been entered by hand, try to calculate the tristimulus values
697                     calc_color_xyz_source_sample(source[1], sample, observer, error);
698                   END;
699                 END;
700               END;
701           END;
702
703
704         PROCEDURE calculate_color(VAR sample: spectral_data);
705       {
```

```
706         WHERE
707             sample = spectral data record of a sample
708         RESULTS
709             sample = color coordinates are calculated for the sample if possible
710        }
711
712             BEGIN
713               get_sample_coordinates(sample, error);
714               IF error THEN
715                 writec('ERROR calculating tristimulus values')
716               ELSE
717                 BEGIN
718                 IF sample.start <> 1 THEN
719                   calc_color_chromaticities(sample.coordinates, error);
720                 IF error THEN
721                   writec('ERROR calculating chromaticities')
722                 ELSE
723                   BEGIN
724                   calc_color_cielab_cieluv(source_coordinates, sample.coordinates, error);
725                   IF error THEN
726                     writec('ERROR calculating CIELAB or CIELUV coordinates')
727                   END;
728                 END;
729               IF NOT error THEN
730                 calc_color_dominant_wavelength(source_coordinates, sample.coordinates, data.cmf);
731             END;
732
733
734           BEGIN
735             WITH data DO
736               BEGIN
737               IF observer_name = '02DEG' THEN
738                 deg := 2
739               ELSE
740                 deg := 10;
741               get_source_coordinates;
742               REPEAT
743                 show_descrips('S A M P L E S   M E N U', samples);
744                 cursor(up, 1);
745                 IF use_illuminant_for_color_calc THEN
746                   writeln('ILLUMINANT :  ', illuminant_name)
747                 ELSE
748                   BEGIN
749                   write('SOURCE :  ');
750                   display_descrip(output, source[1], 68, true, true, true);
751                   writeln;
752                   END;
753                 file_io_option_line(file_io_set);
754                 writec('-----------------------------------------------');
755                 writec('1: select source for color calc  2: enter coordinates of samples ');
756                 writec('3: calculate color of samples    4: calculate color differences  ');
757                 writec('5: calculate dominant wavelength 6: change display of coordinates');
758                 jump(1);
759                 getchar('select one of the above -or- Q(uit to the main menu', q, no_default, file_io_set + ['Q', '1'..'6']);
```

```
760               IF (q IN file_io_set) THEN
761                 file_io('FROM SAMPLES MENU', q, file_io_set, samples)
762               ELSE
763                 BEGIN
764                 CASE q OF
765                   '1':
766                     BEGIN
767                     writeln(' ': 10, 'Illuminant selected in weights menu :  ', illuminant_name);
768                     jump(2);
769                     q1 := 'I';
770                     getchar('use the I(lluminant, select a S(ource, or E(nter coordinates by hand', q1, use_default, ['I', 'S
771                     CASE q1 OF
772                       'S':
773                         BEGIN
774                         temp_source := source[1];
775                         get_spectra('SELECT SOURCE FOR COLOR CALCULATIONS', temp_source, 0, use_default);
776                         use_illuminant_for_color_calc := false;
777                         source[1] := temp_source;
778                         END;
779                       'I':
780                         use_illuminant_for_color_calc := true;
781                       'E':
782                         BEGIN
783                         use_illuminant_for_color_calc := false;
784                         enter_source_coordinates('ENTER COORDINATES OF SOURCE FOR COLOR CALCULATIONS', source[1].coordinates,
785                                                  source[1].start, use_default);
786                         prompt(6);
787                         readst('enter a long description for the source', source[1].descrip, no_default, no_null);
788                         readst('enter a short description for the source', source[1].name, no_default, no_null);
789                         jump(2);
790                         writec('NOTICE:  Since you have selected to enter source coordinates (not spectra),');
791                         writec('         color calculations can only be performed on samples where their   ');
792                         writec('         coordinates were also entered by hand                             ');
793                         return(2);
794                         END;
795                       END;
796                     coordinates_entered := false;
797                     FOR i := 1 TO 10 DO
798                       coordinates_entered := coordinates_entered OR (samples[i].start = 1);
799                     IF coordinates_entered THEN
800                       BEGIN
801                       prompt(10);
802                       writec('The sample array contains color coordinates entered by hand');
803                       jump(1);
804                       writec('Selecting a new source or illuminant may make');
805                       writec('use of these coordinates invalid');
806                       jump(1);
807                       q1 := 'Y';
808                       getchar('zero any hand entered coordinates ? (Y,N)', q1, use_default, ['Y', 'N']);
809                       IF q1 = 'Y' THEN
810                         FOR i := 1 TO 10 DO
811                           IF samples[i].start = 1 THEN
812                             zero_spectra(samples[i]);
813                       END;
```

```
814                     get_source_coordinates;
815                     END;
816                   '2':
817                     BEGIN
818                     j := 0;
819                     getin('enter sample number to enter coordinates by hand', j, use_default, 0, 10);
820                     IF j > 0 THEN
821                       BEGIN
822                       k := samples[j].start;
823                       enter_sample_coordinates('ENTER COORDINATES OF A SAMPLE FOR COLOR CALCULATIONS', source_coordinates,
824                                                samples[j].coordinates, samples[j].start, use_default);
825                       IF k = 1 THEN
826                         BEGIN
827                         readst('enter a long description for these coordinates', samples[j].descrip, use_default, no_null);
828                         readst('enter a short description for these coordinates', samples[j].name, use_default, no_null);
829                         END
830                       ELSE
831                         BEGIN
832                         readst('enter a long description for these coordinates', samples[j].descrip, no_default, no_null);
833                         readst('enter a short description for these coordinates', samples[j].name, no_default, no_null);
834                         END;
835                       END;
836                     END;
837                   '3':
838                     BEGIN
839                     IF source_coordinates[t_x] < - 998 THEN
840                       BEGIN
841                       prompt(10);
842                       writec('Source or Illuminant tristimulus values are not valid');
843                       jump(1);
844                       writec('Please select a valid source or illuminant and try again');
845                       return(2);
846                       END
847                     ELSE
848                       BEGIN
849                       j := 0;
850                       getin('enter first sample number for color calculation', j, use_default, 0, 10);
851                       IF j > 0 THEN
852                         BEGIN
853                         num := 1;
854                         getin('enter number of samples', num, use_default, 0, 10 - j + 1);
855                         IF num > 0 THEN
856                           ask_to_make_report('N', out, default_output_filename, make_report)
857                         ELSE
858                           make_report := false;
859                         IF make_report THEN
860                           BEGIN
861                           prompt(2);
862                           END;
863                         FOR i := j TO j - 1 + num DO
864                           BEGIN
865                           IF NOT make_report THEN
866                             prompt(2);
867                           calculate_color(samples[i]);
```

```
868                         IF error THEN
869                           BEGIN
870                           IF NOT make_report THEN
871                             jump(8);
872                           write('Sample ', i: 0, '  ');
873                           display_descrip(output, samples[i], 67, true, true, true);
874                           writeln;
875                           IF NOT make_report THEN
876                             writeln;
877                           writec('Color coordinates could not be calculated for this sample');
878                           END
879                         ELSE
880                           BEGIN
881                           IF make_report THEN
882                             BEGIN
883                             writeln(out, '-------------------------------------------------------------------------------');
884                             writeln(out);
885                             IF use_illuminant_for_color_calc THEN
886                               writeln(out, 'ILLUMINANT :  ', illuminant_name)
887                             ELSE
888                               BEGIN
889                               write(out, 'SOURCE :  ');
890                               display_descrip(out, source[1], 68, true, true, true);
891                               writeln(out);
892                               END;
893                             write(out, 'Sample ', i: 0, '  ');
894                             display_descrip(out, samples[i], 67, true, true, true);
895                             writeln(out);
896                             show_sample_calc(out, samples[i].coordinates, show_color_set[1]);
897                             writeln(out);
898                             writeln(out, '-------------------------------------------------------------------------------');
899                             END
900                           ELSE
901                             BEGIN
902                             write('Sample ', i: 0, '  ');
903                             display_descrip(output, samples[i], 67, true, true, true);
904                             writeln;
905                             show_sample_calc(output, samples[i].coordinates, show_color_set[1]);
906                             END;
907                           END;
908                         IF NOT make_report THEN
909                           return(2);
910                         END;
911                       IF make_report THEN
912                         BEGIN
913                         close(out);
914                         jump(1);
915                         writec('Report Generated');
916                         return(2);
917                         END;
918                       END;
919                     END;
920                   END;
921               '4':
```

```
922                     BEGIN
923                     IF source_coordinates[t_x] < - 998 THEN
924                       BEGIN
925                       prompt(10);
926                       writec('Source or Illuminant tristimulus values are not valid');
927                       jump(1);
928                       writec('Please select a valid source or illuminant and try again');
929                       return(2);
930                       END
931                     ELSE
932                       BEGIN
933                       j := 0;
934                       getin('select the STANDARD sample that others will be compared to', j, use_default, 0, 10);
935                       IF j > 0 THEN
936                         BEGIN
937                         IF samples[j].start = 0 THEN
938                           BEGIN
939                           prompt(10);
940                           writec('The STANDARD sample is empty');
941                           jump(1);
942                           writec('Please select another STANDARD and try again');
943                           return(2);
944                           END
945                         ELSE
946                           BEGIN
947                           calculate_color(samples[j]);
948                           IF error THEN
949                             BEGIN
950                             prompt(10);
951                             writec('ERROR calculating STANDARD sample color');
952                             jump(1);
953                             writec('Please select another STANDARD and try again');
954                             return(2);
955                             END
956                           ELSE
957                             BEGIN
958                             k := 0;
959                             getin('enter first sample number for color differences', k, use_default, 0, 10);
960                             IF k > 0 THEN
961                               BEGIN
962                               num := 1;
963                               getin('enter number of samples', num, use_default, 0, 10 - k + 1);
964                               ask_to_make_report('N', out, default_output_filename, make_report);
965                               IF make_report THEN
966                                 prompt(2);
967                               FOR i := k TO k - 1 + num DO
968                                 BEGIN
969                                 IF i <> j THEN
970                                   BEGIN
971                                   IF NOT make_report THEN
972                                     prompt(2);
973                                   calculate_color(samples[i]);
974                                   IF error THEN
975                                     BEGIN
```

```
                                    IF NOT make_report THEN
                                      jump(8);
                                    write('Sample ', i: 0, '  ');
                                    display_descrip(output, samples[i], 67, true, true, true);
                                    writeln;
                                    IF NOT make_report THEN
                                      jump(1);
                                    writec('Color coordinates could not be calculated');
                                    IF NOT make_report THEN
                                      return(2);
                                    END
                                  ELSE
                                    BEGIN
                                    IF NOT make_report THEN
                                      BEGIN
                                      show_difference_calc(output, samples[j], samples[i], show_color_set[2], deg, true);
                                      return(1);
                                      END
                                    ELSE
                                      BEGIN
                                      writeln(out, '-------------------------------------------------------------------------------
                                      writeln(out);
                                      IF use_illuminant_for_color_calc THEN
                                        writeln(out, 'ILLUMINANT :  ', illuminant_name)
                                      ELSE
                                        BEGIN
                                        write(out, 'SOURCE :  ');
                                        display_descrip(out, source[1], 68, true, true, true);
                                        writeln(out);
                                        END;
                                      show_difference_calc(out, samples[j], samples[i], show_color_set[2], deg, false);
                                      writeln(out);
                                      writeln(out, '-------------------------------------------------------------------------------
                                      END;
                                    END;
                                  END;
                                IF make_report THEN
                                  BEGIN
                                  close(out);
                                  jump(1);
                                  writec('Report Generated');
                                  return(2);
                                  END;
                                END;
                              END;
                            END;
                          END;
                        END;
                      END;
                    '5':
                      BEGIN
                      k := 1;
                      gotoxy(0, 10 + 3);
```

```
1030                    erase(eoscreen);
1031                    writec('Calculate Oominant Wavelength and Excitation Purity');
1032                    jump(1);
1033                    q1 := 'S';
1034                    readch('E(nter coordinates by hand, or S(elect a sample', q1, use_default, null_ok);
1035                    IF q1 = 'E' THEN
1036                      BEGIN
1037                      q1 := 'U';
1038                      readch('E(nter source coordinates by hand, or U(se values in memory', q1, use_default, null_ok);
1039                      IF q1 = 'E' THEN
1040                        BEGIN
1041                        writeln('Chromaticity Coordinates for the SOURCE :');
1042                        readre('enter x', temp1[c_x], use_default);
1043                        readre('enter y', temp1[c_y], use_default);
1044                        END
1045                      ELSE
1046                        temp1 := source_coordinates;
1047                      writeln('Chromaticity Coordinates for the SAMPLE :');
1048                      readre('enter x', temp2[c_x], use_default);
1049                      readre('enter y', temp2[c_y], use_default);
1050                      calc_color_dominant_wavelength(temp1, temp2, cmf);
1051                      prompt(4);
1052                      writec('Calculate Oominant Wavelength and Excitation Purity');
1053                      END
1054                    ELSE
1055                      BEGIN
1056                      i := 0;
1057                      getin('select a sample number', i, use_default, 0, 10);
1058                      k := i;
1059                      IF k > 0 THEN
1060                        BEGIN
1061                        calculate_color(samples[i]);
1062                        calc_color_chromaticities(samples[i].coordinates, error);
1063                        prompt(4);
1064                        writec('Calculate Oominant Wavelength and Excitation Purity');
1065                        jump(1);
1066                        write('Sample ', i: 0, '  ');
1067                        display_descrip(output, samples[i], 67, true, true, true);
1068                        writeln;
1069                        temp1 := source_coordinates;
1070                        temp2 := samples[i].coordinates;
1071                        END;
1072                      END;
1073                    IF k > 0 THEN
1074                      BEGIN
1075                      jump(1);
1076                      writeln('Source Chromaticity Coordinates :');
1077                      show_calc(output, temp1, [c_x..c_y], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_w
1078                                4, 1, 5, 7, 4);
1079
1080                      jump(1);
1081                      writeln('Sample Chromaticity Coordinates :');
1082                      show_calc(output, temp2, [c_x..c_y], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_w
1083                                4, 1, 5, 7, 4);
```

```
1084
1085                        jump(1);
1086                        writeln('Dominant Wavelength and Excitation Purity :');
1087                        show_calc(output, samples[i].coordinates, [dominant_wavelength],
1088                                  [show_calc_label, show_calc_colon, show_calc_value], 4, 1, 5, 7, 2);
1089
1090                        show_calc(output, samples[i].coordinates, [excitation_purity],
1091                                  [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 4, 1, 5, 7, 4);
1092
1093                        IF regis_terminal THEN
1094                          BEGIN
1095                          q1 := 'Y';
1096                          jump(1);
1097                          readch('display a plot of the spectrum locus showing these values ? (Y,N)', q1, use_default, null_ok)
1098                          IF q1 <> 'N' THEN
1099                            plotrg_locus('SPECTRUM LOCUS PLOT', cmf, observer_name, temp1, temp2);
1100                          END
1101                        ELSE
1102                          return(1);
1103                        END;
1104                      END;
1105                    '6':
1106                      select_show_coordinates(show_color_set[1], show_color_set[2], deg);
1107                    'Q': ;
1108                    END;
1109                  END;
1110              UNTIL q = 'Q';
1111              END;
1112          END;
1113
1114
1115        PROCEDURE sources_menu(VAR data: program_data);
1116      {
1117       WHERE
1118           data = main program data record
1119       RESULTS
1120           sources arrays may be manipulated
1121      }
1122
1123          VAR
1124            q, q1: char;
1125            i, j: integer;
1126            cct, temp_r: real;
1127            temp: spectral_data;
1128            weights_temp: ARRAY [1..3] OF spectral_data;
1129            out: text;
1130            make_report, error, error1: boolean;
1131            ss: STRING [78];
1132
1133
1134          BEGIN
1135            WITH data DO
1136              BEGIN
1137              REPEAT
```

```
1138                show_descrips('L I G H T   S O U R C E S   M E N U', sources);
1139                file_io_option_line(file_io_set);
1140                writec('-------------------------------------------------');
1141                writec('1: make a daylight illuminant    2: make a blackbody illuminant');
1142                writec('3: calculate OT of a source      4: calculate CCT of a source  ');
1143                writec('5: calculte CRI of a source      6: calculate color of a source');
1144                writec('7: calculate ASTM weights                                      ');
1145                jump(1);
1146                getchar('select one of the above -or- Q(uit to the main menu', q, no_default, file_io_set + ['Q', '1'..'7']);
1147                IF (q IN file_io_set) THEN
1148                  file_io('FROM SOURCES MENU', q, file_io_set, sources)
1149                ELSE
1150                  BEGIN
1151                  IF q <> 'Q' THEN
1152                    BEGIN
1153                    gotoxy(0, 10 + 3);
1154                    erase(eoscreen);
1155                    CASE q OF
1156                      '1':
1157                        writec('Make a Daylight Illuminant Spectra');
1158                      '2':
1159                        writec('Make a Blackbody Illuminant Spectra');
1160                      '3':
1161                        writec('Calculate Distribution Temperature of a Source');
1162                      '4':
1163                        writec('Calculate Correlated Color Temperatureof a Source');
1164                      '5':
1165                        writec('Calculate Color Rendering Index of a Source');
1166                      '6':
1167                        writec('Calculate Color of a Source');
1168                      '7':
1169                        writec('Calculate Weights of a Source');
1170                    END;
1171                    jump(1);
1172                    i := 0;
1173                    getin('select spectra number', i, use_default, 0, 10);
1174                    IF i <> 0 THEN
1175                      BEGIN
1176                      CASE q OF
1177                        '1':
1178                          BEGIN
1179                          temp := make_daylight_spectra(0, error);
1180                          IF error THEN
1181                            BEGIN
1182                            writec('ERROR in calcuating.... spectra not saved');
1183                            return(1);
1184                            END
1185                          ELSE
1186                            sources[i] := temp;
1187                          END;
1188                        '2':
1189                          BEGIN
1190                          temp := make_blackbody_spectra(0, error);
1191                          IF error THEN
```

```
1192                          BEGIN
1193                          writec('ERROR in calcuating.... spectra not saved');
1194                          return(1);
1195                          END
1196                        ELSE
1197                          sources[i] := temp;
1198                        END;
1199                      '3':
1200                        temp_r := calc_dt_spectra(sources[i], error, true);
1201                      '4':
1202                        BEGIN
1203                        calc_color_xyz_source(sources[i], observer, error);
1204                        calc_color_chromaticities(sources[i].coordinates, error1);
1205                        prompt(10);
1206                        IF error OR error1 THEN
1207                          BEGIN
1208                          writec('ERROR calculating color coordinates');
1209                          writec('CCT of source not calculated');
1210                          writec('please correct data and try again');
1211                          return(2);
1212                          END
1213                        ELSE
1214                          BEGIN
1215                          write('Source ', i: 0, '  ');
1216                          display_descrip(output, sources[i], 67, true, true, true);
1217                          writeln;
1218                          jump(1);
1219                          cct := calc_cct(sources[i].coordinates[u], sources[i].coordinates[v]);
1220                          writeln(' ': 10, 'Correlated Color Temperature :  ', round(cct): 7, ' K');
1221                          IF regis_terminal THEN
1222                            BEGIN
1223                            q1 := 'Y';
1224                            writeln;
1225                            writeln('Your source can be compared to a ', round(cct): 7, ' K blackbody spectra');
1226                            writeln;
1227                            readch('Plot the spectra for comparison on your terminal ?', q1, use_default, null_ok);
1228                            IF q1 = 'Y' THEN
1229                              BEGIN
1230                              { setup the initial conditions }
1231                              { normalize the spectra to a value of 100 at 560 nm if possible }
1232                              error1 := false;
1233                              IF sources[i].val[(560 - sources[i].start) DIV sources[i].inc + 1] <> 0 THEN
1234                                weights_temp[1] := math_constant(sources[i],
1235                                                                 100 / sources[i].val[(560 - sources[i].start) DIV sources[i]
1236                                                                 math_constant_multiply, error1)
1237                              ELSE
1238                                weights_temp[1] := sources[i];
1239                              IF error1 THEN
1240                                weights_temp[1] := sources[i];
1241                              weights_temp[2] := make_blackbody_spectra(cct, error1);
1242                              IF error1 THEN
1243                                BEGIN
1244                                writeln;
1245                                writec('ERROR calculating blackbody spectra');
```

```
1246                                return(2);
1247                                END
1248                              ELSE
1249                                plotrg_mspectra('SOURCE COMPARED TO BLACKBODY PLOT', weights_temp, 1, 2);
1250                              END;
1251                            END
1252                          ELSE
1253                            return(2);
1254                          END;
1255                        END;
1256                      '5':
1257                        BEGIN
1258                        ask_to_make_report('N', out, default_output_filename, make_report);
1259                        IF make_report THEN
1260                          BEGIN
1261                          calc_cri(sources[i], observer, out, make_report);
1262                          close(out);
1263                          jump(1);
1264                          writec('Report Generated');
1265                          return(2);
1266                          END
1267                        ELSE
1268                          calc_cri(sources[i], observer, output, make_report);
1269                        END;
1270                      '6':
1271                        BEGIN
1272                        calc_color_xyz_source(sources[i], observer, error);
1273                        prompt(4);
1274                        IF error THEN
1275                          BEGIN
1276                          writec('ERROR calculating tristimulus values');
1277                          writec('please correct data and try again');
1278                          END
1279                        ELSE
1280                          BEGIN
1281                          write('Source ', i: 0, '  ');
1282                          display_descrip(output, sources[i], 67, true, true, true);
1283                          writeln;
1284                          jump(1);
1285                          writeln('Tristimulus Values :');
1286                          show_calc(output, sources[i].coordinates, [t_x..t_z],
1287                                    [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 4, 1, 5, 7,
1288                          jump(1);
1289                          calc_color_chromaticities(sources[i].coordinates, error);
1290                          IF error THEN
1291                            writec('ERROR calculating chromaticities')
1292                          ELSE
1293                            BEGIN
1294                            writeln('Chromaticity Coordinates :');
1295                            show_calc(output, sources[i].coordinates, [c_x..c_z],
1296                                      [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 4, 1, 5, 7
1297                            show_calc(output, sources[i].coordinates, [u, v, v_prime],
1298                                      [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 4, 1, 5, 7
1299                            END;
```

```
1300                        END;
1301                      return(2);
1302                      END;
1303                    '7':
1304                      BEGIN
1305                      IF sources[i].start < starting_wavelength THEN
1306                        BEGIN
1307                        writec('A source must be loaded into memory to calculate weights');
1308                        return(2);
1309                        END
1310                      ELSE
1311                        BEGIN
1312                        temp.inc := 10;
1313                        temp.start := starting_wavelength;
1314                        temp.stop := ending_wavelength;
1315                        REPEAT
1316                          gotoxy(0, 10 + 3);
1317                          erase(eoscreen);
1318                          IF i < 10 THEN
1319                            ss := 'Calculate weights of spectra number ' + chr(48 + i)
1320                          ELSE
1321                            ss := 'Calculate weights of spectra number ' + chr(48 + (i DIV 10)) + chr(48 + (i MOD 10));
1322                          writec(ss);
1323                          jump(1);
1324                          writec('valid increments   0 = 5 nm,  1 = 10 nm,  2 = 20 nm');
1325                          CASE temp.inc OF
1326                            5:
1327                              j := 0;
1328                            10:
1329                              j := 1;
1330                            20:
1331                              j := 2;
1332                            END;
1333                          getin('select an increment for the weights', j, use_default, 0, 2);
1334                          CASE j OF
1335                            0:
1336                              temp.inc := 5;
1337                            1:
1338                              temp.inc := 10;
1339                            2:
1340                              temp.inc := 20;
1341                            END;
1342                          getin('select a starting wavelength', temp.start, use_default, starting_wavelength,
1343                                ending_wavelength - temp.inc);
1344                          getin('select an ending wavelength', temp.stop, use_default, temp.start + temp.inc, ending_wavele
1345                          error := (temp.stop - temp.start) MOD temp.inc <> 0;
1346                          IF error THEN
1347                            BEGIN
1348                            jump(1);
1349                            writec('ERROR in wavelength parameters, check and re-enter');
1350                            return(1);
1351                            END;
1352                        UNTIL NOT error;
1353                        writec('Calculated weights will be loaded into three spectral arrays');
```

```
1354                          getin('select first spectra number to contain new weights', j, no_default, 0, 8);
1355                          IF j <> 0 THEN
1356                            BEGIN
1357                            gotoxy(0, 10 + 3);
1358                            erase(eoscreen);
1359                            IF i < 10 THEN
1360                              ss := 'Calculate weights of spectra number ' + chr(48 + i)
1361                            ELSE
1362                              ss := 'Calculate weights of spectra number ' + chr(48 + (i DIV 10)) + chr(48 + (i MOD 10));
1363                            writec(ss);
1364                            jump(1);
1365                            make_weights(sources[i], cmf, sources, j, temp.start, temp.stop, temp.inc, error);
1366                            END;
1367                          END;
1368                        END;
1369                      END;
1370                    END;
1371                  END;
1372                END;
1373              UNTIL q = 'Q';
1374              END;
1375          END;
1376
1377
1378        PROCEDURE file_math_menu(VAR data: program_data);
1379      {
1380       WHERE
1381           data = main program data record
1382       RESULTS
1383           all spectral arrays may be manipulated
1384      }
1385
1386          VAR
1387            q: char;
1388            i: integer;
1389
1390          BEGIN
1391            q := 'T';
1392            REPEAT
1393              WITH data DO
1394                BEGIN
1395                prompt(2);
1396                writec('F I L E   M A T H');
1397                jump(1);
1398                writec('File Math provides a tool to manipulate spectra using');
1399                writec('several mathematical functions and operations available');
1400                writec('and can be performed on selected spectra already loaded');
1401                writec('in memory or in disk files');
1402                jump(2);
1403                writeln(' ': 10, 'W :  File Math using WEIGHTS spectra');
1404                writeln(' ': 10, 'I :  File Math using ILLUMINANT spectra');
1405                writeln(' ': 10, 'C :  File Math using SOURCE spectra');
1406                writeln(' ': 10, 'O :  File Math using OBSERVER spectra');
1407                writeln(' ': 10, 'S :  File Math using SAMPLES spectra');
```

```
1408              writeln(' ': 10, 'L :  File Math using LIGHT SOURCES spectra');
1409              writeln(' ': 10, 'T :  File Math using TEMPORARY spectra');
1410              jump(2);
1411              getchar('     select one of the above -or- Q(uit to main menu', q, use_default, ['W', 'I', 'C', 'O', 'S', 'L',
1412              CASE q OF
1413                'W':
1414                  file_math('File Math using WEIGHTS spectra', weights, file_io_set);
1415                'I':
1416                  file_math('File Math using ILLUMINANT spectra', illuminant, file_io_set);
1417                'C':
1418                  file_math('File Math using SOURCE spectra', illuminant, file_io_set);
1419                'O':
1420                  file_math('File Math using OBSERVER spectra', observer, file_io_set);
1421                'S':
1422                  file_math('File Math using SAMPLES spectra', samples, file_io_set);
1423                'L':
1424                  file_math('File Math using LIGHT SOURCES spectra', sources, file_io_set);
1425                'T':
1426                  file_math('File Math using TEMPORARY spectra', file_math_temps, file_io_set);
1427                'Q': ;
1428                END;
1429              END;
1430          UNTIL q = 'Q';
1431        END;
1432
1433
1434      PROCEDURE configuration_menu(VAR data: program_data);
1435    {
1436     WHERE
1437        data = main program data record
1438     RESULTS
1439        main program data record may be loaded from disk or written to disk
1440    }
1441
1442        VAR
1443          q: char;
1444          filename: fn;
1445
1446        BEGIN
1447          REPEAT
1448            prompt(2);
1449            writec('C O N F I G U R A T I O N   M E N U');
1450            jump(1);
1451            writec('The following options provide a mechanism to save, restore,');
1452            writec('many program variables and features');
1453            jump(2);
1454            writeln(' ': 10, 'R :  Read program varables from a disk file');
1455            writeln(' ': 10, 'S :  Save program varaibles to a disk file');
1456            writeln(' ': 10, 'Z :  Zero program variables');
1457            writeln(' ': 10, 'C :  Change default report filename');
1458            jump(2);
1459            getchar('     select one of the above -or- Q(uit to main menu', q, no_default, ['R', 'S', 'Z', 'Q', 'C']);
1460            CASE q OF
1461              'R':
```

```
1462                BEGIN
1463                prompt(6);
1464                writec('READ PROGRAM VARAIBLES FROM A DISK FILE');
1465                jump(2);
1466                writec('The default disk file name is COLOR_PROGRAM_SAVED_VARIABLES.DAT.');
1467                writec('This file is written each time this program terminates,');
1468                writec('and read each time the program starts');
1469                jump(2);
1470                stass(filename, 'COLOR_PROGRAM_SAVED_VARIABLES.DAT');
1471                readst('enter filename', filename, use_default, no_null);
1472                read_data(filename, data, false);
1473                END;
1474              'S':
1475                BEGIN
1476                prompt(6);
1477                writec('SAVE PROGRAM VARAIBLES TO A DISK FILE');
1478                jump(2);
1479                writec('The default disk file name is COLOR_PROGRAM_SAVED_VARIABLES.DAT.');
1480                writec('This file is written each time this program terminates,');
1481                writec('and read each time the program starts');
1482                jump(2);
1483                stass(filename, 'COLOR_PROGRAM_SAVED_VARIABLES.DAT');
1484                readst('enter filename', filename, use_default, no_null);
1485                save_data(filename, data);
1486                END;
1487              'C':
1488                BEGIN
1489                prompt(6);
1490                writec('CHANGE DEFAULT REPORT FILENAME');
1491                jump(2);
1492                readst('enter filename', data.default_output_filename, use_default, no_null);
1493                END;
1494              'Z':
1495                init_data(data);
1496              'Q': ;
1497              END;
1498          UNTIL q = 'Q';
1499        END;
1500
1501
1502      PROCEDURE main_menu;
1503    {
1504     RESULTS
1505         main menu options are displayed and executed when selected
1506    }
1507
1508        CONST
1509          menu_margin = 30;
1510
1511        BEGIN
1512          REPEAT
1513            prompt(5);
1514            writeln(' ': menu_margin, 'W:  Weights Menu');
1515            writeln;
```

```
1516              writeln(' ': menu_margin, 'S:  Samples Menu');
1517              writeln;
1518              writeln(' ': menu_margin, 'L:  Light Sources Menu');
1519              writeln;
1520              writeln(' ': menu_margin, 'F:  File Math Menu');
1521              writeln;
1522              writeln(' ': menu_margin, 'C:  Configuration Menu');
1523              writeln;
1524              writeln(' ': menu_margin, 'H:  Help Menu');
1525
1526              gotoxy(0, 22);
1527
1528              setscn([blinking]);
1529              write('**');
1530              setscn([not blinking, negative]);
1531              write('  REMINDER: type a "?" in response to any prompt to return to this menu  ');
1532              setscn([positive, blinking]);
1533              writeln('**');
1534              setscn([not_blinking]);
1535
1536              gotoxy(0, 18);
1537              getchar('     select one of the above -or- Q(uit', data.main_q, use_default, ['W', 'S', 'L', 'F', 'C', 'H', 'Q'])
1538
1539              CASE data.main_q OF
1540                'W':
1541                  weights_menu(data);
1542                'S':
1543                  samples_menu(data);
1544                'L':
1545                  sources_menu(data);
1546                'F':
1547                  file_math_menu(data);
1548                'C':
1549                  configuration_menu(data);
1550                'H':
1551                  call_help('dua0:[users.user1.rmm.color.spectra.clr_data]COLOR_PROGRAM.HLB');
1552                'Q': ;
1553                END;
1554            UNTIL data.main_q = 'Q';
1555          END;
1556
1557
1558        BEGIN
1559
1560          start;
1561
1562          read_data('  ', data, true);
1563
1564          start_spectra(data.file_io_set);
1565
1566          data.main_q := 'S';
1567
1568        1: {abort location}
1569
```

```
1570          main_menu;
1571
1572          save_data('  ', data);
1573
1574          prompt(10);
1575
1576          writeln('Bye...');
1577
1578          jump(1);
1579
1580        END.
```

Invocation line:
COLOR.PASF/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM COLOR;
    2        ($nolist)
   16           ($list)
   17           { include the spectra handling routines }
   18           %INCLUDE 'spec:spectra.pas';

spec:spectra.pas

2   1       { SPECTRA.PAS }
2   2
2   3       VAR
2   4            set_spectra_debug,
2   5
2   6            { spectra debug mode displays every data item read from spectra data
2   7              files so that bad data or bad interpretation of data can be found }
2   8
2   9            set_spectra_debug_pause: boolean;
2  10
2  11            { spectra debug pause mode displays a message and waits for the user
2  12              to press RETURN after each debug message }
2  13
2  14
2  15       %include 'spec:spectra.hdr';

spec:spectra.hdr

3   1       { SPECTRA.HDR  contains definitions of procedures that load, display, and save spectral data
3   2
3   3
3   4          written by R. Mitchell Miller }
3   5
3   6       CONST
3   7         starting_wavelength = 360; { minimum wavelength for calculations }
3   8         ending_wavelength = 780; { maximum wavelength for calculations }
3   9         minimum_increment = 5;
3  10         number_of_wavelengths = 85; { maximum number of wavelengths with a minimum wavelength increment of 5 nm }
3  11         stringmax = 40; { length of character strings }
3  12         fn_length = 70; { length of filename variables }
3  13
3  14         X = 1;
3  15         Y = 2;
3  16         Z = 3;
3  17
3  18       TYPE
3  19         {[F-]}
3  20
3  21            lstring = packed array [1..stringmax] of char;
3  22            string5 = packed array [1..5] of char;
3  23            sstring = packed array [1..10] of char;
3  24            fn = packed array [1..fn_length] of char;
3  25
3  26            coordinate_type = ( first,
3  27                                t_x,
3  28                                t_y,
3  29                                t_z,
```

```
3  30                               c_x,
3  31                               c_y,
3  32                               c_z,
3  33                               dominant_wavelength,
3  34                               excitation_purity,
3  35                               u,
3  36                               v,
3  37                               u_prime,
3  38                               v_prime,
3  39                               l_star,
3  40                               a_star,
3  41                               b_star,
3  42                               c_star_ab,
3  43                               h_ab,
3  44                               E_ab,
3  45                               u_star,
3  46                               v_star,
3  47                               c_star_uv,
3  48                               s_uv,
3  49                               h_uv,
3  50                               E_uv,
3  51                               last );
3  52
3  53          coordinate_type_names = array [coordinate_type] of string5;
3  54
3  55          coordinate_type_set = set of coordinate_type;
3  56
3  57          show_coordinate_type = (
3  58                               Tristimulus,
3  59                               Chromaticity_1,
3  60                               Chromaticity_2,
3  61                               Domwave_purity,
3  62                               CIELAB_1,
3  63                               CIELAB_2,
3  64                               CIELUV_1,
3  65                               CIELUV_2,
3  66                               SPECIAL_1,
3  67                               SPECIAL_2,
3  68                               SPECIAL_3,
3  69                               SPECIAL_4,
3  70                               SPECIAL_5
3  71                                   );
3  72
3  73          show_coordinate_data = array [show_coordinate_type] of integer;
3  74
3  75
3  76          coordinate_sets = ( XYZ,
3  77                               xyY,
3  78                               LAB,
3  79                               LUV );
3  80
3  81          coordinate_data = array [coordinate_type] of real;
3  82
3  83
```

```
3  84
3  85          math_options = ( math_add,
3  86                           math_subtract_1_from_2,
3  87                           math_subtract_2_from_1,
3  88                           math_multiply,
3  89                           math_divide_1_by_2,
3  90                           math_divide_2_by_1);
3  91
3  92          math_constant_options = ( math_constant_add,
3  93                                    math_constant_multiply,
3  94                                    math_constant_log10,
3  95                                    math_constant_unlog10,
3  96                                    math_constant_raise,
3  97                                    math_constant_k_over_s );
3  98
3  99          show_calc_options = ( show_calc_label,
3 100                                show_calc_colon,
3 101                                show_calc_delta,
3 102                                show_calc_value,
3 103                                show_calc_nl,
3 104                                show_calc_nl_when_done);
3 105
3 106          show_calc_options_set = set of show_calc_options;
3 107
3 108          file_io_set_type = set of char;
3 109          getchar_charset = set of char;
3 110
3 111          spectral_data=record                    ( data type of color structure )
3 112            filename : fn;
3 113            descrip: lstring;
3 114            name: sstring;
3 115            start,stop,inc: integer;
3 116            val: array[1..number_of_wavelengths] of real;
3 117            coordinates: coordinate_data;
3 118          end;
3 119
3 120          big_spectral_data=record                    ( data type of color structure )
3 121            filename : fn;
3 122            descrip: lstring;
3 123            val: array [360..830] of real;
3 124            coordinates: coordinate_data;
3 125          end;
3 126
3 127      {     chrom=array[0..50] of real;)   ( ???)
3 128      {     r30=array[1..30] of real;  )   ( ???)
3 129
3 130      CONST
3 131          coordinate_names = coordinate_type_names(
3 132                              'FIRST',
3 133                              'X    ',
3 134                              'Y    ',
3 135                              'Z    ',
3 136                              'x    ',
3 137                              'y    ',
```

```
3 138                                   'z    ',
3 139                                   'DW   ',
3 140                                   'Pe   ',
3 141                                   'u    ',
3 142                                   'v    ',
3 143                                   'u''   ',
3 144                                   'v''   ',
3 145                                   'L*   ',
3 146                                   'a*   ',
3 147                                   'b*   ',
3 148                                   'c*_ab',
3 149                                   'h_ab ',
3 150                                   'E_ab ',
3 151                                   'u*   ',
3 152                                   'v*   ',
3 153                                   'c*_uv',
3 154                                   's_uv ',
3 155                                   'h_uv ',
3 156                                   'E_uv ',
3 157                                   'LAST '
3 158                                    );
3 159
3 160       {[F+]}
3 161
3 162         { SPLINE.HDR contains definitions for cubic spline and interpolation routines }
3 163       {$NOLIST}
3 165       {$LIST}
3 166
3 167         { OPSYS.HDR contains definitions for calling some VAX/VMS system services and library routines }
3 168       {$NOLIST}
3 170       {$LIST}
3 171
3 172
3 173       FUNCTION spectra_debug: boolean;
3 174         EXTERNAL;
3 175       {
3 176        RESULTS
3 177            a value of TRUE if debugging statements should be generated for the spectra routines
3 178       }
3 179
3 180
3 181       FUNCTION spectra_debug_pause: boolean;
3 182         EXTERNAL;
3 183       {
3 184        RESULTS
3 185            returns a value of TRUE if the user should be prompted to "Press RETURN to Continue" after each spectra
3 186            debugging statement is executed
3 187       }
3 188
3 189
3 190       FUNCTION big_math(VAR spec1, spec2: big_spectral_data;
3 191                         option: math_options;
3 192                         VAR error: boolean): big_spectral_data;
3 193         EXTERNAL;
```

```
3 194       {
3 195        WHERE
3 196            spec1 = big spectral data record number 1
3 197            spec2 = big spectral data record number 2
3 198            option = type of operation to perform
3 199            error = signals error in parameters
3 200        RESULTS
3 201            a big spectral data record containing the result of the math operation
3 202       }
3 203
3 204
3 205       FUNCTION calc_cct(u, v: real): real;
3 206         EXTERNAL;
3 207       {
3 208        WHERE
3 209            u = 1960 UCS color coordinate u of a test source
3 210            v = 1960 UCS color coordinate v of a test source
3 211        RESULTS
3 212            the correlated color temperature of a source having coordinates u,v
3 213       }
3 214
3 215
3 216       PROCEDURE calc_cri(source: spectral_data;
3 217                          observer: ARRAY [low..high: integer] OF spectral_data;
3 218                          VAR out: text;
3 219                          make_report: boolean);
3 220         EXTERNAL;
3 221       {
3 222        WHERE
3 223            source = spectral data for a test source
3 224            observer = array [1..3] of spectral_data
3 225            out = text file to write results to
3 226            make_report = TRUE for writing to file, FALSE when writing to the terminal screen
3 227        RESULTS
3 228            the color rendering index of a source is calculated and displayed
3 229       }
3 230
3 231
3 232       FUNCTION calc_difference(sample1, sample2: coordinate_data): coordinate_data;
3 233         EXTERNAL;
3 234       {
3 235        WHERE
3 236            sample1 = coordinate data of sample 1
3 237            sample2 = coordinate data of sample 2
3 238        RESULTS
3 239            the differences of color coordinates (and color differences) are calculated and returned
3 240       }
3 241
3 242
3 243       FUNCTION calc_dt_spectra(source: spectral_data;
3 244                                VAR error: boolean;
3 245                                show_calc: boolean): real;
3 246         EXTERNAL;
3 247       {
```

```
3 248       WHERE
3 249           source = spectra of light source to calculate distribution temperature from
3 250           error = signals error
3 251        RESULTS
3 252           the distribution temperature of the source
3 253       }
3 254
3 255
3 256       PROCEDURE calc_lagrange(VAR bspec: big_spectral_data;
3 257                               spec: spectral_data;
3 258                               VAR error: boolean);
3 259         EXTERNAL;
3 260       {
3 261        WHERE
3 262           bspec = a spectral data record to hold the 1 nm data
3 263           spec = a spectral data record containing the data to interpolate
3 264           error = signals an error in the routine
3 265        RESULTS
3 266           bspec is loaded with 1 nm interpolated data from spec
3 267       }
3 268
3 269
3 270       FUNCTION calc_color_bad_value(val: real): boolean;
3 271         EXTERNAL;
3 272       {
3 273        WHERE
3 274           val = a color coordinates value
3 275        RESULTS
3 276           a value of TRUE is returned if val has a value very close to the value of -999 that signals an error condition
3 277       }
3 278
3 279
3 280       PROCEDURE calc_color_chromaticities(VAR coordinates: coordinate_data;
3 281                                           VAR error: boolean);
3 282         EXTERNAL;
3 283       {
3 284        WHERE
3 285           coordinates = an array already containing the tristimulus values X,Y, & Z of a source or sample
3 286         RESULTS
3 287           the following chromaticity coordinates are calculated and returned:
3 288             1931 chromaticity coordinates x,y, & z
3 289             and UCS coordinates u,v,u',& v'
3 290       }
3 291
3 292
3 293       PROCEDURE calc_color_CIELAB_CIELUV(VAR weights, sample: coordinate_data;
3 294                                          VAR error: boolean);
3 295         EXTERNAL;
3 296       {
3 297        WHERE
3 298           weights = color coordinates of the source
3 299           sample = color coordinates of the sample
3 300           error = signals an error
3 301         RESULTS
```

```
3 302           CIELAB & CIELUV color coordinates for the sample are calculated and returned
3 303       }
3 304
3 305
3 306       PROCEDURE calc_color_dominant_wavelength(VAR source, sample: coordinate_data;
3 307                                                VAR cmf: ARRAY [low..high: integer] OF big_spectral_data);
3 308         EXTERNAL;
3 309       {
3 310        WHERE
3 311           source =  color coordinates of the light source (only chromaticities x & y are used)
3 312           sample =  color coordinates of the sample (only chromaticities x & y are used)
3 313           cmf = color matching functions where array index 4 & 5 point to spectral chromaticities of the spectrum locus
3 314         RESULTS
3 315           Dominant wavemength and excitation purity are calculated and returned in the SAMPLE array
3 316       }
3 317
3 318
3 319       PROCEDURE calc_color_xyz_source(VAR source: spectral_data;
3 320                                       observer: ARRAY [low..high: integer] OF spectral_data;
3 321                                       VAR error: boolean);
3 322         EXTERNAL;
3 323       {
3 324        WHERE
3 325           source = spectral data for source or illuminant
3 326           observer = array [1..3] of spectral_data
3 327           error = signals error
3 328        RESULTS
3 329           the tristimulus values X,Y,& Z of the light source are calculated and returned
3 330       }
3 331
3 332
3 333       PROCEDURE calc_color_xyz_source_sample(source: spectral_data;
3 334                                              VAR sample: spectral_data;
3 335                                              observer: ARRAY [low..high: integer] OF spectral_data;
3 336                                              VAR error: boolean);
3 337         EXTERNAL;
3 338       {
3 339        WHERE
3 340           source = spectral data for source or illuminant
3 341           sample = spectral data for a sample
3 342           observer = array [1..3] of spectral_data
3 343           error = signals error
3 344        RESULTS
3 345           the tristimulus values X,Y,& Z of the sample are calculated and returned
3 346       }
3 347
3 348
3 349       PROCEDURE calc_color_xyz_weights(VAR weights: ARRAY [low..high: integer] OF spectral_data;
3 350                                        VAR error: boolean);
3 351         EXTERNAL;
3 352       {
3 353        WHERE
3 354           weights = array [1..9] of spectral_data
3 355        RESULTS
```

```
3 356           the tristimulus values X,Y,& Z of the illuminant are calculated and returned
3 357       }
3 358
3 359
3 360       PROCEDURE calc_color_xyz_weights_sample(weights: ARRAY [low..high: integer] OF spectral_data;
3 361                                               VAR sample: spectral_data;
3 362                                               VAR error: boolean);
3 363         EXTERNAL;
3 364       {
3 365        WHERE
3 366           weights = array [1..9] of spectral data
3 367           sample = spectral data
3 368           error = signals error
3 369       RESULTS
3 370           the tristimulus values X,Y,& Z of the sample are calculated and returned
3 371       }
3 372
3 373
3 374       PROCEDURE call_help(filename: PACKED ARRAY [low..high: integer] OF char);
3 375         EXTERNAL;
3 376       {
3 377        WHERE
3 378           filename = a string containing the filename of a help library file specification
3 379         RESULTS
3 380           help information as requested by the user for the specified help library
3 381       }
3 382
3 383
3 384       PROCEDURE check_spectral_file(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 385                                     VAR descrip: lstring;
3 386                                     VAR start, stop, inc, number_of_spectra_in_file: integer;
3 387                                     VAR error_code: integer);
3 388         EXTERNAL;
3 389       {
3 390        WHERE
3 391           s = name of data file
3 392           descrip = description line from file
3 393           start = starting wavelength
3 394           stop = ending wavelength
3 395           inc = wavelength increment
3 396           number_of_spectra_in_file = number of spectra in file
3 397           error_code = error code that is passed back to calling routine
3 398        RESULTS
3 399           an error_code is returned indicating any errors in reading all spectra from the file
3 400       }
3 401
3 402
3 403       PROCEDURE correct_spectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 404                                 VAR spec: spectral_data);
3 405         EXTERNAL;
3 406       {
3 407        WHERE
3 408           s = string to display
3 409           spec = spectral data record to correct
```

```
3 410         RESULTS
3 411            a spectral data record of manually corrected data
3 412       )
3 413
3 414
3 415       PROCEDURE display_descrip(VAR out: text;
3 416                                 spec: spectral_data;
3 417                                 length: integer;
3 418                                 typ, long, short: boolean);
3 419         EXTERNAL;
3 420       {
3 421        WHERE
3 422            out = file variable for output location
3 423            spec =  spectra data record
3 424            length =  length of line that should be written
3 425            typ = TRUE to display coordinate typ in front of description
3 426            long = TRUE to display the DESCRIPTION of the spectral data
3 427            short = TRUE to display the NAME of the spectral data
3 428        RESULTS
3 429            the description is written to the desired file
3 430       )
3 431
3 432
3 433       PROCEDURE display_spectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 434                                 VAR spec: spectral_data);
3 435         EXTERNAL;
3 436
3 437       {
3 438        WHERE
3 439            s = string to display
3 440            spec = spectral data record
3 441        RESULTS
3 442            the spectra is displayed
3 443       )
3 444
3 445
3 446       PROCEDURE display_mspectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 447                                  VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 448                                  starting_spectra_array_index, number_of_spectra: integer);
3 449         EXTERNAL;
3 450
3 451       {
3 452        WHERE
3 453            s = string to display
3 454            spec = spectral data record
3 455            starting_spectra_array_index = element of array to start displaying  (range is low..high)
3 456            number_of_spectra = number of spectra to be displayed
3 457        RESULTS
3 458            the selected spectra are displayed
3 459       )
3 460
3 461
3 462       PROCEOURE enter_sample_coordinates(s: PACKEO ARRAY [low..high: integer] OF char;
3 463                                          source: coordinate_data;
```

```
3 464                                       VAR sample_coord: coordinate_data;
3 465                                       VAR typ: integer;
3 466                                       default: default_type);
3 467       EXTERNAL;
3 468      {
3 469       WHERE
3 470           s = string to display
3 471           source = coordinates of light source
3 472           sample_coord = coordinates array for a sample
3 473           typ = T when coordinates have already been loaded into the array ( <>1 when the array is empty or contains coordinat
3 474                                                                                 calculated from spectral data )
3 475                   (NOTE:  the variable START of record spectral_data can be passed as this parameter)
3 476       RESULTS
3 477           chromaticity coordinates and tristimulus values are returned in the array
3 478      }
3 479
3 480
3 481      PROCEDURE enter_source_coordinates(s: PACKED ARRAY [low..high: integer] OF char;
3 482                                       VAR source_coord: coordinate_data;
3 483                                       VAR typ: integer;
3 484                                       default: default_type);
3 485       EXTERNAL;
3 486      {
3 487       WHERE
3 488           s = string to display
3 489           source = coordinates array for a source
3 490           typ = 1 when coordinates have already been loaded into the array ( <>1 when the array is empty or contains coordinat
3 491                                                                                 calculated from spectral data )
3 492                   (NOTE:  the variable START of record spectral_data can be passed as this parameter)
3 493       RESULTS
3 494           chromaticity coordinates and tristimulus values are returned in the array
3 495      }
3 496
3 497
3 498      PROCEDURE enter_spectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 499                              VAR spec: spectral_data);
3 500       EXTERNAL;
3 501      {
3 502       WHERE
3 503           s = string to display
3 504           spec = spectral data record
3 505       RESULTS
3 506           a spectral data record full of manually entered data
3 507      }
3 508
3 509
3 510      PROCEDURE file_io(s: PACKED ARRAY [lows..highs: integer] OF char;
3 511                        option: char;
3 512                        file_io_set: file_io_set_type;
3 513                        VAR spec: ARRAY [low..high: integer] OF spectral_data);
3 514       EXTERNAL;
3 515      {
3 516       WHERE
3 517           s = string to display
```

```
3 518           option = character defining the desired operations
3 519           spec = an array of spectral data to operate on
3 520           file_io_set_type = set defines permitted operations by this routine
3 521         RESULTS
3 522           spectra modified by various routines
3 523       )
3 524
3 525
3 526       PROCEDURE file_io_option_line(file_io_set: file_io_set_type);
3 527         EXTERNAL;
3 528       {
3 529        WHERE
3 530           file_io_set = set of characters that define the options available from the file_io procedure
3 531         RESULTS
3 532           an option line on the user's terminal that shows the procedure file_io options available
3 533       )
3 534
3 535
3 536       PROCEDURE file_io_set_init(VAR file_io_set: file_io_set_type);
3 537         EXTERNAL;
3 538       {
3 539        WHERE
3 540           file_io_set = is a variable of type set of char
3 541         RESULTS
3 542           a set of acceptable characters for the file_io routine's operations
3 543       )
3 544
3 545
3 546       PROCEDURE file_math(s: PACKED ARRAY [lows..highs: integer] OF char;
3 547                           VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 548                           file_io_set: file_io_set_type);
3 549         EXTERNAL;
3 550       {
3 551        WHERE
3 552           s = string to display
3 553           spec = an array of spectra data to operate on
3 554           file_io_set = character set of valid file_io options
3 555         RESULTS
3 556           spectra modified by various routines
3 557       )
3 558
3 559
3 560       FUNCTION fold_1_like_2(spec1, spec2: spectral_data;
3 561                              VAR error: boolean): spectral_data;
3 562         EXTERNAL;
3 563
3 564       {
3 565        WHERE
3 566             spec1 = spectral data record number 1
3 567             spec2 = spectral data record number 2
3 568             error = signals error in parameters
3 569         RESULTS
3 570             fold_1_like_2  = spectra 1 folded to the limits of spectra 2
3 571       )
```

```
3 572
3 573
3 574       PROCEDURE getinf(fn: PACKED ARRAY [low..high: integer] OF char;
3 575                        VAR info: PACKED ARRAY [lowd..highd: integer] OF char;
3 576                        VAR extra_info: PACKED ARRAY [lowx..highx: integer] OF char;
3 577                        default: default_type);
3 578         EXTERNAL;
3 579       {
3 580        WHERE
3 581            fn  = filename of getinf data file
3 582            info  = line of information returned from the file
3 583            extra_info  = extra information on the line after info
3 584            default     = permits use of a default value
3 585         RESULTS
3 586            info and extra_info selected from a data file or entered by hand
3 587       }
3 588
3 589
3 590       PROCEDURE getchar(s: PACKED ARRAY [low..high: integer] OF char;
3 591                         VAR value: char;
3 592                         default: default_type;
3 593                         valid_response: getchar_charset);
3 594         EXTERNAL;
3 595       {
3 596        WHERE
3 597            s = string to display in prompt
3 598            value = character variable to contain response
3 599            default = permits use of default value of variable value
3 600            valid_response = set of valid characters
3 601         RESULTS
3 602            a character from the valid_response set
3 603       }
3 604
3 605
3 606       PROCEDURE get_spectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 607                             VAR spec: spectral_data;
3 608                             starting_spectra_in_file_to_read: integer;
3 609                             default: default_type);
3 610         EXTERNAL;
3 611       {
3 612        WHERE
3 613            s = string to display
3 614            spec = spectral data record
3 615            starting_spectra_in_file_to_read = starting spectra number
3 616            default = (use_default,no_default for filenames)
3 617        RESULTS
3 618            a spectra selected by filename is read from disk and returned
3 619       }
3 620
3 621
3 622       PROCEDURE get_mspectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 623                              VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 624                              starting_spectra_array_index, starting_spectra_in_file_to_read: integer;
3 625                              VAR number_of_spectra: integer;
```

```
3 626                                 default: default_type);
3 627       EXTERNAL;
3 628      {
3 629       WHERE
3 630           s = string to display
3 631           spec = spectral data record
3 632           starting_spectra_array_index = element of array to start reading into  (range is low..high)
3 633           starting_spectra_in_file_to_read = starting spectra number
3 634           number_of_spectra = number of spectra to be loaded into the array
3 635           default = (use_default,no_default for filenames)
3 636       RESULTS
3 637           one or more spectra are selected by filename and read from disk and returned
3 638      }
3 639
3 640
3 641      PROCEDURE helper(VAR filename: descriptor_block);
3 642        NONPASCAL;
3 643      {
3 644       WHERE
3 645           filename = a string descriptor pointing to the address of a string containing the filename
3 646                      of a help library file specification
3 647        RESULTS
3 648           help information as requested by the user for the specified help library
3 649      }
3 650
3 651
3 652      FUNCTION make_blackbody_spectra(t: real;
3 653                                      VAR error: boolean): spectral_data;
3 654        EXTERNAL;
3 655      {
3 656       WHERE
3 657           t = color temperature of a blackbody to create (in automatic mode)
3 658                                                    or a value of <= 0 causes the user to be prompted for a value of tempera
3 659           spec = spectral data record
3 660        RESULTS
3 661           make_blackbody_spectra = spectra of blackbody radiator having the color temperature specified
3 662      }
3 663
3 664
3 665      FUNCTION make_daylight_spectra(t: real;
3 666                                     VAR error: boolean): spectral_data;
3 667        EXTERNAL;
3 668      {
3 669       WHERE
3 670           t = correlated color temperature of illuminant to create (in automatic mode)
3 671                or a value of <= 0 causes the user to be prompted for a value of temperature
3 672           spec = spectral data record
3 673       RESULTS
3 674           make_daylight_spectra = spectra of daylight illuminant having the CCT specified
3 675      }
3 676
3 677
3 678      PROCEDURE make_spectra_from_big_spectra(VAR bspec: big_spectral_data;
3 679                                              VAR spec: spectral_data;
```

```
3 680                                           start, stop, inc: integer;
3 681                                           VAR error: boolean);
3 682       EXTERNAL;
3 683     {
3 684      WHERE
3 685          bspec = a spectral data record to containing 1 nm data
3 686          spec = a spectral data record to contain 5, 10 or 20 nm data
3 687          error = signals an error in the routine
3 688      RESULTS
3 689          a spectral data record is created from a big spectral data record
3 690     }
3 691
3 692
3 693     PROCEDURE make_weights(spec: spectral_data;
3 694                            cmf: ARRAY [clow..chigh: integer] OF big_spectral_data;
3 695                            VAR wspec: ARRAY [low..high: integer] OF spectral_data;
3 696                            starting_array_index, start, stop, inc: integer;
3 697                            VAR error: boolean);
3 698       EXTERNAL;
3 699     {
3 700      WHERE
3 701          spec = spectral data of light source to calculate weights from
3 702          cmf = 1 nm color matching functions
3 703          wspec = array of spectral data to contain the calculated weights
3 704          starting_array_index = index for writing first weight spectra to wspec
3 705          start = starting wavelength for weights returned
3 706          stop = ending wavelength for weights returned
3 707          inc = increment for weights returned
3 708          error = signals an error condition
3 709      RESULTS
3 710          ASTM like weights are loaded into an array (to be used or saved)
3 711     }
3 712
3 713
3 714     FUNCTION math(spec1, spec2: spectral_data;
3 715                   option: math_options;
3 716                   VAR error: boolean): spectral_data;
3 717       EXTERNAL;
3 718     {
3 719      WHERE
3 720          spec1                = spectral data record number 1
3 721          spec2                = spectral data record number 2
3 722          option               = type of operation to perform
3 723          error                = signals error in parameters
3 724      RESULTS
3 725          a spectral data record is loaded with the selected math operation
3 726     }
3 727
3 728
3 729     FUNCTION math_constant(spec: spectral_data;
3 730                            r: real;
3 731                            option: math_constant_options;
3 732                            VAR error: boolean): spectral_data;
3 733       EXTERNAL;
```

```
3 734
3 735       {
3 736        WHERE
3 737            spec = spectral data record
3 738            r = constant value for operation
3 739            option = type of operation to perform
3 740            error = signals error in parameters
3 741        RESULTS
3 742            a spectral data record is loaded with the selected math operation
3 743       }
3 744
3 745
3 746       PROCEDURE plotrg_locus(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 747                              VAR cmf: ARRAY [low..high: integer] OF big_spectral_data;
3 748                              observer_name: PACKED ARRAY [olow..ohigh: integer] OF char;
3 749                              source, sample: coordinate_data);
3 750         EXTERNAL;
3 751       {
3 752        WHERE
3 753            s = string to display
3 754            cmf = 1 nm color matching functions
3 755            observer_name = 02OEG or 10OEG for indicating the observer
3 756            source = color coordinates of the light source
3 757            sample = color coordinates of the sample
3 758        RESULTS
3 759            a 1931 (or 1964) CIE chromaticity diagram is drawn on the screeen
3 760       }
3 761
3 762
3 763       PROCEDURE plotrg_spectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 764                                VAR spec: spectral_data);
3 765         EXTERNAL;
3 766
3 767       {
3 768        WHERE
3 769            s = string to display
3 770            spec = spectral data record
3 771        RESULTS
3 772            a spectral data record is plotted on a REGIS terminal
3 773       }
3 774
3 775
3 776       PROCEDURE plotrg_mspectra(s: PACKED ARRAY [slow..shigh: integer] OF char;
3 777                                 VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 778                                 starting_spectra_array_index, number_of_spectra: integer);
3 779         EXTERNAL;
3 780
3 781       {
3 782        where
3 783            s = string to display
3 784            spec = array of spectral data record
3 785            starting_spectra_array_index = element of array to start displaying  (range is low..high)
3 786            number_of_spectra = number of spectra to be displayed
3 787        RESULTS
```

```
3 788           one or more spectral data records are plotted on a REGIS terminal
3 789       }
3 790
3 791
3 792       PROCEDURE report_mspectra(out_fn: PACKED ARRAY [olow..ohigh: integer] OF char;
3 793                                 s: PACKED ARRAY [slow..shigh: integer] OF char;
3 794                                 VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 795                                 starting_spectra_array_index, number_of_spectra, width, ext, line_length: integer;
3 796                                 VAR error_code: integer);
3 797         EXTERNAL;
3 798       {
3 799        WHERE
3 800           out_fn = filename that report should be written to
3 801           s = string to include at top of report
3 802           spec = spectral data records
3 803           starting_spectra_array_index = element of array to start displaying  (range is low..high)
3 804           number_of_spectra = number of spectra to be displayed
3 805           width = filed width for writing real numbers  (values < 2 are set to the default of 12)
3 806           ext = number of digits to display past the decimal point (valid range is 0 to width-2, -1 signals scientific notatio
3 807           line_length = line length of display device
3 808           error_code = signals an error condition
3 809        RESULTS
3 810           a report is generated for the spectral data record(s)
3 811       }
3 812
3 813
3 814       PROCEDURE report_spectra(out_fn: PACKED ARRAY [olow..ohigh: integer] OF char;
3 815                                s: PACKED ARRAY [slow..shigh: integer] OF char;
3 816                                VAR spec: spectral_data;
3 817                                width, ext, line_length: integer;
3 818                                VAR error_code: integer);
3 819         EXTERNAL;
3 820       {
3 821        WHERE
3 822           out_fn = filename that report should be written to
3 823           s = string to include at top of report
3 824           spec = spectral data record
3 825           error_code = signals an error condition
3 826        RESULTS
3 827           a report is generated for the spectral data record
3 828       }
3 829
3 830
3 831       FUNCTION regis_terminal: boolean;
3 832         EXTERNAL;
3 833       { RESULTS
3 834           a boolean value of TRUE if the process terminal has the REGIS characteristic for plotting
3 835       }
3 836
3 837
3 838       PROCEDURE retrieve_big_mspectra(VAR spec: ARRAY [low..high: integer] OF big_spectral_data;
3 839                                       VAR error_code: integer);
3 840         EXTERNAL;
3 841       {
```

```
3 842       WHERE
3 843           spec = spectral data record
3 844           error_code = error code that is passed back to calling routine
3 845       RESULTS
3 846           one or more big spectral data records are loaded from a values in a data file
3 847       }
3 848
3 849
3 850       PROCEDURE retrieve_mspectra(VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 851                                   starting_spectra_array_index, starting_spectra_in_file_to_read, number_of_spectra: integer;
3 852                                   VAR error_code: integer);
3 853         EXTERNAL;
3 854       {
3 855       WHERE
3 856           spec = spectral data record
3 857           starting_spectra_array_index = element of array to start reading into  (range is low..high)
3 858           starting_spectra_in_file_to_read = identifies which column of data in file to read (range is 1..what's there)
3 859           number_of_spectra = number of spectra to be loaded into the array
3 860           error_code = error code that is passed back to calling routine
3 861       RESULTS
3 862           one or more spectral data records are loaded from a values in a data file
3 863       }
3 864
3 865
3 866       PROCEDURE retrieve_spectra(VAR spec: spectral_data;
3 867                                  starting_spectra_in_file_to_read: integer;
3 868                                  VAR error_code: integer);
3 869         EXTERNAL;
3 870       {
3 871       WHERE
3 872           spec = spectral data record
3 873           number_of_spectra = number of spectra to be loaded into the array
3 874           error_code = error code that is passed back to calling routine
3 875       RESULTS
3 876           one spectral data records is loaded from a values in a data file
3 877       }
3 878
3 879
3 880       PROCEDURE save_mspectra(VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 881                               starting_spectra_array_index, number_of_spectra: integer;
3 882                               VAR error_code: integer);
3 883         EXTERNAL;
3 884       {
3 885       WHERE
3 886           spec                          = array of spectral data record s
3 887           starting_spectra_array_index  = element of array to start saving
3 888           number_of_spectra             = number of spectra to be saved
3 889           error_code                    = signals error
3 890       RESULTS
3 891           one or more spectral data records are written to a data file
3 892       }
3 893
3 894
3 895       PROCEDURE save_spectra(VAR spec: spectral_data;
```

```
3 896                                   VAR error_code: integer);
3 897         EXTERNAL;
3 898
3 899       {
3 900        WHERE
3 901            spec = spectral data record
3 902            error_code = signals error
3 903        RESULTS
3 904            one spectral data record is written to a data file
3 905       }
3 906
3 907
3 908       PROCEDURE save_mspectra_as_xydata(VAR spec: ARRAY [low..high: integer] OF spectral_data;
3 909                                         starting_spectra_array_index, number_of_spectra: integer;
3 910                                         VAR error_code: integer);
3 911         EXTERNAL;
3 912
3 913       {
3 914        WHERE
3 915            spec = spectral data record
3 916            starting_spectra_array_index = element of array to start saving
3 917            number_of_spectra = number of spectra to be saved
3 918            error_code = signals error
3 919        RESULTS
3 920            one or more spectral data records are written to a data file in X,Y1,Y2,Y3.... format
3 921       }
3 922
3 923
3 924       PROCEDURE save_spectra_as_xydata(VAR spec: spectral_data;
3 925                                        VAR error_code: integer);
3 926         EXTERNAL;
3 927       {
3 928        WHERE
3 929            spec = spectral data record
3 930            error_code = signals an error
3 931        RESULTS
3 932            one spectral data record is written to a data file in X,Y format
3 933       }
3 934
3 935
3 936       PROCEDURE say_error(s: PACKED ARRAY [low..high: integer] OF char;
3 937                           error, pause: boolean);
3 938         EXTERNAL;
3 939       {
3 940        WHERE
3 941            s = message to be displayed if there is an error
3 942            error = signals an error
3 943            pause = signals that the user should be prompted to press RETURN to continue
3 944        RESULTS
3 945            the user is notified upon ERROR being TRUE
3 946       }
3 947
3 948
3 949       PROCEDURE say_error_code(s: PACKED ARRAY [low..high: integer] OF char;
```

```
3 950                                 error_code: integer;
3 951                                 pause: boolean);
3 952         EXTERNAL;
3 953       {
3 954        WHERE
3 955            s =  message to be displayed if there is an error
3 956            error_code =  integer returned by a spectra routine to signal errors
3 957            pause =  signals that the user should be prompted to press RETURN to continue
3 958        RESULTS
3 959            the user is notified upon an error signalled by error_code having a non-zero value
3 960       }
3 961
3 962
3 963       PROCEDURE select_show_coordinates(VAR color, difference: show_coordinate_data;
3 964                                         degree: integer);
3 965         EXTERNAL;
3 966       {
3 967        WHERE
3 968            color = array that selects color coordinates to be displayed or reported
3 969            difference = array that selects color coordinate differences to be displayed or reported
3 970        RESULTS
3 971            values are returned in the above two arrays
3 972       }
3 973
3 974
3 975       PROCEDURE show_calc(VAR out: text;
3 976                           VAR coordinates: coordinate_data;
3 977                           to_show: coordinate_type_set;
3 978                           options: show_calc_options_set;
3 979                           space_before, space_between, space_after, field_width, extension: integer);
3 980         EXTERNAL;
3 981       {
3 982        WHERE
3 983            out = file for output
3 984            coordinates = coordinate data to display
3 985            to_show:
3 986            options = options to use when displaying value
3 987            space_before = number of spaces to write before a value or label
3 988            space_between = number of spaces to write between a value and label
3 989            space_after = number of spaces to write after a value
3 990            field_width = width of space to write value in
3 991            extension = number of digits to display in each value past the decimal place
3 992        RESULTS
3 993            a value is displayed using the options selected
3 994       }
3 995
3 996
3 997       PROCEDURE show_descrips(s: PACKED ARRAY [lows..highs: integer] OF char;
3 998                               VAR spec: ARRAY [low..high: integer] OF spectral_data);
3 999         EXTERNAL;
31000       {
31001        WHERE
31002            s = string to display
31003            spec = array of spectra_data to display descriptions of
```

```
31004       RESULTS
31005           descriptions (short & long) are displayed on the users' terminal
31006       }
31007
31008
31009       PROCEDURE show_difference_calc(VAR out: text;
31010                                      VAR sample1, sample2: spectral_data;
31011                                      color: show_coordinate_data;
31012                                      degree: integer;
31013                                      user_viewing: boolean);
31014         EXTERNAL;
31015       {
31016        WHERE
31017            out = text file
31018            sample 1 = STANDARD spectra for difference calculations
31019            sample 2 = SAMPLE spectra for difference calculations
31020            color = selection of differences to show
31021            degree = 2 for two degree observer, 10 for ten degree observer
31022            user_viewing = use RETURN(1) to pause just prior to overfilling the screen
31023        RESULTS
31024            differences are written to file variable OUT
31025       }
31026
31027
31028       PROCEDURE show_sample_calc(VAR out: text;
31029                                  sample: coordinate_data;
31030                                  color: show_coordinate_data);
31031         EXTERNAL;
31032       {
31033        WHERE
31034            out = text f le
31035            sample = coordinates to be displayed
31036            color = selection of coordinates to show
31037        RESULTS
31038            sample color calculations are displayed on the terminal
31039       }
31040
31041
31042       PROCEDURE special(VAR coord1, coord2, coord3: coordinate_data;
31043                         VAR lname, name: descriptor_block;
31044                         VAR degree, number: integer);
31045         NONPASCAL;
31046       {
31047        WHERE
31048            coord1 & coord2 = array of color coordinates of two samples
31049            coord3 = array of color coordinates of color differences calculated between samples 1 and 2
31050            lname = descriptor block for 40 character name of the special calculation
31051            name = descriptor block for 5 character name of the special calcuation
31052            degree = 2 if coordinates were selected using the two degree observer, 10 for ten degree observer
31053            number = special calculation number to use  ( >0 indicates :
31054                                                              calclations indicated by number should be performed
31055                                                              name and lname for the calculation should be returned
31056                                                         <0 indicated :
31057                                                              name and lname for the calculation should be returned for abs(num
```

```
31058
31059       RESULTS
31060           coord3[last] contains the result (or -999 if calculation was not possible)
31061           number = 0 for successful completion
31062                   -1 for invalid special calculation number
31063       }
31064
31065
31066       PROCEDURE spectra_debug_pause_execution;
31067         EXTERNAL;
31068       {
31069        RESULTS
31070           a vale of TRUE indicates program execution should be stopped until a return is pressed after encountered an error
31071       }
31072
31073
31074       FUNCTION spectra_ymin(spec: spectral_data): real;
31075         EXTERNAL;
31076
31077       {
31078        WHERE
31079           spec = spectral data record
31080        RESULTS
31081           the minimum value in the spectral data record is returned
31082       }
31083
31084
31085       FUNCTION spectra_ymax(spec: spectral_data): real;
31086         EXTERNAL;
31087
31088       {
31089        WHERE
31090           spec = spectral data record
31091        RESULTS
31092           the maximum value in the spectral data record is returned
31093       }
31094
31095
31096       FUNCTION spectral_summation(spec: spectral_data;
31097                                   starting_wavelength, ending_wavelength: integer;
31098                                   VAR error: boolean): real;
31099         EXTERNAL;
31100
31101       {
31102        WHERE
31103           spec                = spectral data record
31104           starting_wavelength = starting wavelength of summation
31105           ending_wavelength   = ending wavelength of summation
31106           error               = signals error in parameters
31107        RESULTS
31108           the summation of the spectral data record is returned
31109       }
31110
31111
```

spec:spectra.hdr

```
31112       PROCEDURE strass(VAR t: PACKED ARRAY [tlow..thigh: integer] OF char;
31113                        VAR s: PACKED ARRAY [slow..shigh: integer] OF char);
31114         EXTERNAL;
31115       {
31116        WHERE
31117            t = target string to hold a passable copy of the source conformant array string
31118            s = source string which is not passable by value since it a conformant array
31119        RESULTS
31120            the source string is copied to the target string
31121       }
31122
31123
31124       PROCEDURE stass(VAR t: PACKED ARRAY [tlow..thigh: integer] OF char;
31125                       s: PACKED ARRAY [slow..shigh: integer] OF char);
31126         EXTERNAL;
31127       {
31128        WHERE
31129            t = target string to hold a passable copy of the source conformant array string
31130            s = source string which is not passable by value since it a conformant array
31131        RESULTS
31132            the source string is copied to the target string
31133       }
31134
31135
31136       FUNCTION truncate_1_like_2(spec1, spec2: spectral_data;
31137                                  VAR error: boolean): spectral_data;
31138         EXTERNAL;
31139
31140       {
31141        WHERE
31142            spec1 = spectral data record number 1
31143            spec2 = spectral data record number 2
31144            error = signals error in parameters
31145        RESULTS
31146            a spectral data record truncated to the wavelength limits of the second
31147       }
31148
31149
31150       PROCEDURE zero_spectra(VAR spec: spectral_data);
31151         EXTERNAL;
31152       {
31153        WHERE
31154            spec = spectra data record
31155        RESULTS
31156            a zeroed spectral data record
31157       }
```

spec:spectra.pas

```
2  16
2  17       FUNCTION spectra_debug(: boolean);
2  18       {
2  19        RESULTS
2  20            a value of TRUE if debugging statements should be generated for the spectra routines
```

```
2  21      }
2  22      begin
2  23        spectra_debug := set_spectra_debug;
2  24      end;
2  25
2  26      FUNCTION spectra_debug_pause(: boolean);
2  27      {
2  28       RESULTS
2  29           returns a value of TRUE if the user should be prompted to "Press RETURN to Continue" after each spectra
2  30           debugging statement is executed
2  31      }
2  32      begin
2  33        spectra_debug_pause := set_spectra_debug_pause;
2  34      end;
2  35
2  36      procedure start_spectra ( var file_io_set: file_io_set_type );
2  37      begin
2  38
2  39        { initially turn off these features }
2  40        set_spectra_debug := false;
2  41        set_spectra_debug_pause := false;
2  42
2  43        { define a set of rountine character responses for the file_io procedure }
2  44        file_io_set_init(file_io_set);
2  45
2  46      end;

COLOR.PASF

   19          {$nolist}

Invocation line:
COLDR.PASF/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM BIG_MATH;
 ($nolist)
 ($list)


  FUNCTION BIG_MATH ((VAR spec1, spec2: big_spectral_data; option: math_options; VAR error: boolean): big_spectral_data)
{
 WHERE
     spec1 = big spectral data record number 1
     spec2 = big spectral data record number 2
     option = type of operation to perform
     error = signals error in parameters
 RESULTS
     a big spectral data record containing the result of the math operation
}

    VAR
      i: integer;
      temp: big_spectral_data;

    BEGIN
      error := false;
      { no error checking needed now since wavelengths are hard coded }
      IF NOT error THEN
        BEGIN
        temp := spec1; { temp assumes many characteristics of the first set of spectral data }
        temp.filename := '                                                                    ';
        FOR i := 360 TO 830 DO
          BEGIN
          CASE option OF

            math_add:
              temp.val[i] := spec1.val[i] + spec2.val[i];

            math_subtract_1_from_2:
              temp.val[i] := spec2.val[i] - spec1.val[i];

            math_subtract_2_from_1:
              temp.val[i] := spec1.val[i] - spec2.val[i];

            math_multiply:
              temp.val[i] := spec1.val[i] * spec2.val[i];

            math_divide_1_by_2:
              IF abs(spec2.val[i]) > 1E-30 THEN
                temp.val[i] := spec1.val[i] / spec2.val[i]
              ELSE
                temp.val[i] := - 999;

            math_divide_2_by_1:
              IF abs(spec1.val[i]) > 1E-30 THEN
                temp.val[i] := spec2.val[i] / spec1.val[i]
              ELSE
                temp.val[i] := - 999;
            END;
```

```
60                END;
61              END;
62            BIG_MATH := temp;
63          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]BIG_MATH.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_CCT;
 {$nolist}
 {$list}


  FUNCTION CALC_CCT ((u, v: real): real) ;
{
 WHERE
     u = 1960 UCS color coordinate u of a test source
     v = 1960 UCS color coordinate v of a test source
 RESULTS
     the correlated color temperature of a source having coordinates u,v
}

    VAR
      temp_value, u_value, v_value, slope_value, cct_value: ARRAY [1..30] OF real;
      i: integer;
      cct_temp: real;

{   Procedures for Correlated color temperature calculations
   Adapted from Fortran routines written by A.R. Robertson             }


    BEGIN

      { load values from A.R. Robertson's journal article :  u values }

      u_value[1] := 0.18065;
      u_value[2] := 0.18132;
      u_value[3] := 0.18208;
      u_value[4] := 0.18293;
      u_value[5] := 0.18388;
      u_value[6] := 0.18494;
      u_value[7] := 0.18611;
      u_value[8] := 0.18739;
      u_value[9] := 0.18879;
      u_value[10] := 0.19031;
      u_value[11] := 0.19461;
      u_value[12] := 0.19960;
      u_value[13] := 0.20523;
      u_value[14] := 0.21140;
      u_value[15] := 0.21804;
      u_value[16] := 0.22507;
      u_value[17] := 0.23243;
      u_value[18] := 0.24005;
      u_value[19] := 0.24787;
      u_value[20] := 0.25585;
      u_value[21] := 0.26394;
      u_value[22] := 0.27210;
      u_value[23] := 0.28032;
      u_value[24] := 0.28854;
      u_value[25] := 0.29676;
      u_value[26] := 0.30496;
      u_value[27] := 0.31310;
      u_value[28] := 0.32119;
```

```
            u_value[29] := 0.32920;
            u_value[30] := 0.33713;

            { load values from A.R. Robertson's journal article :  v values }

            v_value[1] := 0.26589;
            v_value[2] := 0.26845;
            v_value[3] := 0.27118;
            v_value[4] := 0.27407;
            v_value[5] := 0.27708;
            v_value[6] := 0.28020;
            v_value[7] := 0.28340;
            v_value[8] := 0.28666;
            v_value[9] := 0.28995;
            v_value[10] := 0.29325;
            v_value[11] := 0.30139;
            v_value[12] := 0.30918;
            v_value[13] := 0.31645;
            v_value[14] := 0.32309;
            v_value[15] := 0.32906;
            v_value[16] := 0.33436;
            v_value[17] := 0.33901;
            v_value[18] := 0.34305;
            v_value[19] := 0.34653;
            v_value[20] := 0.34948;
            v_value[21] := 0.35198;
            v_value[22] := 0.35405;
            v_value[23] := 0.35575;
            v_value[24] := 0.35713;
            v_value[25] := 0.35822;
            v_value[26] := 0.35906;
            v_value[27] := 0.35968;
            v_value[28] := 0.36011;
            v_value[29] := 0.36038;
            v_value[30] := 0.36051;

            { load values from A.R. Robertson's journal article :  slope values }

            slope_value[1] := - 0.2548;
            slope_value[2] := - 0.2687;
            slope_value[3] := - 0.2854;
            slope_value[4] := - 0.3047;
            slope_value[5] := - 0.3267;
            slope_value[6] := - 0.3515;
            slope_value[7] := - 0.3790;
            slope_value[8] := - 0.4094;
            slope_value[9] := - 0.4426;
            slope_value[10] := - 0.4787;
            slope_value[11] := - 0.5517;
            slope_value[12] := - 0.7043;
            slope_value[13] := - 0.8484;
            slope_value[14] := - 1.017;
            slope_value[15] := - 1.216;
            slope_value[16] := - 1.450;
```

```
114             slope_value[17] := - 1.728;
115             slope_value[18] := - 2.016;
116             slope_value[19] := - 2.465;
117             slope_value[20] := - 2.96;
118             slope_value[21] := - 3.576;
119             slope_value[22] := - 4.355;
120             slope_value[23] := - 5.365;
121             slope_value[24] := - 6.711;
122             slope_value[25] := - 8.572;
123             slope_value[26] := - 11.29;
124             slope_value[27] := - 15.56;
125             slope_value[28] := - 23.20;
126             slope_value[29] := - 40.41;
127             slope_value[30] := - 113.8;
128
129             { load values from A.R. Robertson's journal article :  slope values }
130
131             cct_value[1] := 100000.0;
132             cct_value[2] := 50000.0;
133             cct_value[3] := 33333.0;
134             cct_value[4] := 25000.0;
135             cct_value[5] := 20000.0;
136             cct_value[6] := 16667.0;
137             cct_value[7] := 14285.0;
138             cct_value[8] := 12500.0;
139             cct_value[9] := 11111.0;
140             cct_value[10] := 10000.0;
141             cct_value[11] := 8000.0;
142             cct_value[12] := 6667.0;
143             cct_value[13] := 5714.0;
144             cct_value[14] := 5000.0;
145             cct_value[15] := 4444.0;
146             cct_value[16] := 4000.0;
147             cct_value[17] := 3636.0;
148             cct_value[18] := 3333.0;
149             cct_value[19] := 3077.0;
150             cct_value[20] := 2857.0;
151             cct_value[21] := 2667.0;
152             cct_value[22] := 2500.0;
153             cct_value[23] := 2353.0;
154             cct_value[24] := 2222.0;
155             cct_value[25] := 2105.0;
156             cct_value[26] := 2000.0;
157             cct_value[27] := 1905.0;
158             cct_value[28] := 1818.0;
159             cct_value[29] := 1739.0;
160             cct_value[30] := 1667.0;
161
162             FOR i := 1 TO 30 DO
163               temp_value[i] := (v - v_value[i] - (slope_value[i] * (u - u_value[i]))) / sqrt(1 + sqr(slope_value[i]));
164             i := 0;
165             REPEAT
166               i := i + 1;
167               cct_temp := temp_value[i] / temp_value[i + 1];
```

```
  168             UNTIL (i = 29) OR (cct_temp < 0.0);
  169
  170             cct_temp := 1 / cct_value[i] + (temp_value[i] / (temp_value[i] - temp_value[i + 1])) * (1 / cct_value[i + 1] - 1 /
  171                         cct_value[i]);
  172             cct_temp := 1 / cct_temp + 0.05;
  173             CALC_CCT := cct_temp;
  174           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_CCT.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM CALC_COLOR_BAD_VALUE;
    2        ($nolist)
    7        ($list)
    8
    9
   10         FUNCTION CALC_COLOR_BAD_VALUE ((val: real): boolean) ;
   11       {
   12        WHERE
   13            val = a color coordinates value
   14        RESULTS
   15            a value of TRUE is returned if val has a value very close to the value of -999 that signals an error condition
   16       }
   17
   18
   19           BEGIN
   20             CALC_COLOR_BAD_VALUE := (val > - 999.5) AND (val < - 998.5);
   21           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_BAD_VALUE.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM CALC_COLOR_CHROMATICITIES;
2        ($nolist)
7        ($list)
8
9
10        PROCEDURE CALC_COLOR_CHROMATICITIES ((VAR coordinates: coordinate_data; VAR error: boolean)) ;
11      {
12       WHERE
13          coordinates = an array already containing the tristimulus values X,Y, & Z of a source or sample
14       RESULTS
15          the following chromaticity coordinates are calculated and returned:
16            1931 chromaticity coordinates x,y, & z
17            and UCS coordinates u,v,u',& v'
18      }
19
20          VAR
21            temp: real;
22            coord: coordinate_type;
23
24          BEGIN
25
26            { calculate 1931 chromaticity coordinates }
27
28            temp := 0;
29            FOR coord := t_x TO t_z DO
30              temp := temp + coordinates[coord];
31            error := temp = 0;
32            IF NOT error THEN
33              FOR coord := c_x TO c_z DO
34                coordinates[coord] := coordinates[loophole(coordinate_type, ord(t_x) + ord(coord) - ord(c_x))] / temp
35            ELSE
36            { on error set all values to -999 }
37              FOR coord := c_x TO c_z DO
38                coordinates[coord] := - 999;
39
40            { calculate UCS chromaticity coordinates }
41
42            temp := coordinates[t_x] + 15 * coordinates[t_y] + 3 * coordinates[t_z];
43            error := error OR (temp = 0);
44            IF NOT error THEN
45              IF (temp <> 0) AND (coordinates[c_x] > - 998) THEN
46                BEGIN
47                coordinates[u] := 4 * coordinates[t_x] / temp;
48                coordinates[v] := 6 * coordinates[t_y] / temp;
49                coordinates[u_prime] := coordinates[u];
50                coordinates[v_prime] := 9 * coordinates[t_y] / temp;
51                END
52              ELSE
53              { on error set all values to -999 }
54                FOR coord := u TO v_prime DO
55                  coordinates[coord] := - 999;
56
57            { set all other color coordinates to a value of -999 that will later be calculated from the results of this routine
58            FOR coord := l_star TO pred(last) DO
59              coordinates[coord] := - 999;
```

```
   60
   61          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_CHROMATICITIES.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_COLOR_CIELAB_CIELUV;
 {$nolist}
 {$list}


  PROCEDURE CALC_COLOR_CIELAB_CIELUV ((VAR weights, sample: coordinate_data; VAR error: boolean)) ;
{
 WHERE
     weights = color coordinates of the source
     sample = color coordinates of the sample
     error = signals an error
  RESULTS
     CIELAB & CIELUV color coordinates for the sample are calculated and returned
}


    FUNCTION raise(y, x: real): real;

      VAR
        temp: real;

      BEGIN
        IF y > 0 THEN
          temp := exp(x * ln(y))
        ELSE IF y < 0 THEN
          BEGIN
          temp := - 999; ( signal error on negative y values )
          error := true;
          END
        ELSE
          temp := 0; ( assume 0^x = 0 )
        raise := temp;
      END;


    FUNCTION cube_root(val: real): real;

      BEGIN
        IF val < 0 THEN
          cube_root := - raise( - val, 1 / 3)
        ELSE
          cube_root := raise(val, 1 / 3);
      END;


    FUNCTION fn(val: real): real;

      BEGIN
        IF val < 0.008856 THEN
          fn := 7.787 * val + 16 / 116
        ELSE
          fn := cube_root(val);
      END;
```

```
FUNCTION hue_angle(a, b: real): real;

  CONST
    pi = 3.1415927;

  VAR
    temp: real;

  BEGIN
    IF a <> 0 THEN
      temp := arctan(b / a) / pi * 180
    ELSE IF b > 0 THEN
      temp := 90
    ELSE IF b < 0 THEN
      temp := - 90
    ELSE
      temp := - 999;

    IF a < 0 THEN
      temp := temp + 180
    ELSE IF b < 0 THEN
      temp := temp + 360;

    hue_angle := temp;
  END;


BEGIN
  { there is currently no error checking in this procedure }
  error := false;

  { calculate CIELAB values }

  sample[l_star] := 116 * fn(sample[t_y] / weights[t_y]) - 16;
  sample[a_star] := 500 * (fn(sample[t_x] / weights[t_x]) - fn(sample[t_y] / weights[t_y]));
  sample[b_star] := 200 * (fn(sample[t_y] / weights[t_y]) - fn(sample[t_z] / weights[t_z]));
  sample[c_star_ab] := sqrt(sqr(sample[a_star]) + sqr(sample[b_star]));
  sample[h_ab] := hue_angle(sample[a_star], sample[b_star]);

  { calculate CIELUV values }

  sample[u_star] := 13 * sample[l_star] * (sample[u_prime] - weights[u_prime]);
  sample[v_star] := 13 * sample[l_star] * (sample[v_prime] - weights[v_prime]);

  sample[c_star_uv] := sqrt(sqr(sample[u_star]) + sqr(sample[v_star]));
  sample[h_uv] := hue_angle(sample[u_star], sample[v_star]);
  sample[s_uv] := sample[c_star_uv] / sample[l_star];

END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_CIELAB_CIELUV.PASF;4/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
1       PROGRAM CALC_COLOR_OOMINANT_WAVELENGTH;
2        ($nolist)
7        ($list)
8
9
10        PROCEOURE CALC_COLOR_OOMINANT_WAVELENGTH ((VAR source, sample: coordinate_data; VAR cmf: ARRAY [low..high: integer] OF
11                                                  big_spectral_data)) ;
12      {
13       WHERE
14           source =  color coordinates of the light source (only chromaticities x & y are used)
15           sample =  color coordinates of the sample (only chromaticities x & y are used)
16           cmf = color matching functions where array index 4 & 5 point to spectral chromaticities of the spectrum locus
17        RESULTS
18           Dominant wavemength and excitation purity are calculated and returned in the SAMPLE array
19      }
20
21          VAR
22            i, j, wavelength: integer;
23            x, y, xn, yn, xl, yl, m1, m2, sample_angle, first_angle, last_angle, test_angle, test_angle2, ratio, distance1, dis
24             shift, domwave: real;
25            error, complementary: boolean;
26
27
28          FUNCTION rotate(angle, degrees: real): real;
29           { this function rotates an angle by the specified number of degrees (to avoid error checking near angles of 0 and 36
30
31            BEGIN
32              angle := angle + degrees;
33              IF angle < 0 THEN
34                angle := angle + 360
35              ELSE IF angle > 360 THEN
36                angle := angle - 360;
37              rotate := angle;
38            ENO;
39
40
41          FUNCTION angle(x, y: real): real;
42
43            CONST
44              pi = 3.1415927;
45
46            VAR
47              dx, dy, temp: real;
48
49            BEGIN
50              dx := x - xn;
51              dy := y - yn;
52              IF dx <> 0 THEN
53                temp := arctan(dy / dx) / pi * 180
54              ELSE IF dy > 0 THEN
55                temp := 90
56              ELSE IF dy < 0 THEN
57                temp := - 90
58              ELSE
59                BEGIN
```

```
60              {       writeln('x= ',x,' y = ',y,' xn= ',xn,' yn = ',yn); }
61              error := true;
62              temp := - 999;
63              END;
64
65            IF NOT error THEN
66              BEGIN
67              IF dx < 0 THEN
68                temp := temp + 180
69              ELSE IF dy < 0 THEN
70                temp := temp + 360;
71              END;
72
73            angle := temp;
74          END;
75
76
77        BEGIN
78          error := false;
79          complementary := false;
80          x := sample[c_x];
81          y := sample[c_y];
82          xn := source[c_x];
83          yn := source[c_y];
84          sample_angle := angle(x, y);
85          first_angle := angle(cmf[4].val[360], cmf[5].val[360]);
86          last_angle := angle(cmf[4].val[830], cmf[5].val[830]);
87          shift := - angle(cmf[4].val[550], cmf[5].val[550]);
88
89          IF (rotate(sample_angle, shift) > rotate(first_angle, shift)) AND (rotate(sample_angle, shift) < rotate(last_angle,
90             shift)) THEN
91            BEGIN
92            complementary := true;
93            sample_angle := rotate(sample_angle, 180);
94            END;
95          { rotate the coordinates so the search never crosses the 360 degree boundary }
96          shift := - angle((cmf[4].val[360] + cmf[4].val[830]) / 2, (cmf[5].val[360] + cmf[5].val[830]) / 2);
97          sample_angle := rotate(sample_angle, shift);
98          wavelength := 360;
99          REPEAT
100           wavelength := wavelength + 10;
101           test_angle := rotate(angle(cmf[4].val[wavelength], cmf[5].val[wavelength]), shift);
102   {     writeln('wave = ',wavelength:0,' x= ',cmf[4].val[wavelength]:8:5,' y= ',cmf[5].val[wavelength]:8:5,' test angle = ',
103            test_angle:10:4); }
104         UNTIL (test_angle <= sample_angle) OR (wavelength = 830);
105         {  writeln('test_angle = ',test_angle:10:4,' sample_angle = ',sample_angle:10:4,' TEST > = error = ',error); }
106         error := test_angle > sample_angle;
107         {  writeln('error = ',error); }
108         IF NOT error THEN
109           BEGIN
110           wavelength := wavelength - 11;
111           REPEAT
112             wavelength := wavelength + 1;
113             test_angle := rotate(angle(cmf[4].val[wavelength], cmf[5].val[wavelength]), shift);
```

```
114               {      writeln('wavelength = ',wavelength:0); }
115             UNTIL (test_angle <= sample_angle) OR (wavelength = 830);
116             IF wavelength = 360 THEN
117               BEGIN
118               domwave := 360;
119               xl := cmf[4].val[360];
120               yl := cmf[5].val[360];
121               END
122             ELSE
123               BEGIN
124               domwave := wavelength;
125               test_angle2 := rotate(angle(cmf[4].val[wavelength - 1], cmf[5].val[wavelength - 1]), shift);
126               ratio := 0;
127               IF test_angle2 - test_angle > 1E-30 THEN
128                 BEGIN
129                 ratio := (sample_angle - test_angle) / (test_angle2 - test_angle);
130                 domwave := domwave - ratio;
131                 END;
132               IF NOT complementary THEN
133                 BEGIN
134                 xl := cmf[4].val[wavelength] - (cmf[4].val[wavelength - 1] - cmf[4].val[wavelength]) * ratio;
135                 yl := cmf[5].val[wavelength] - (cmf[5].val[wavelength - 1] - cmf[5].val[wavelength]) * ratio;
136                 END
137               ELSE
138                 BEGIN
139                 m2 := (cmf[5].val[830] - cmf[5].val[360]) / (cmf[4].val[830] - cmf[4].val[360]);
140                 IF x - xn = 0 THEN
141                   xl := x
142                 ELSE
143                   BEGIN
144                   m1 := (y - yn) / (x - xn);
145                   xl := (m1 * x - m2 * cmf[4].val[360] - y + cmf[5].val[360]) / (m1 - m2);
146                   END;
147                 yl := m2 * (xl - cmf[4].val[830]) + cmf[5].val[830];
148                 END;
149               END;
150
151             { calculate excitation purity }
152
153             distance1 := sqrt(sqr(x - xn) + sqr(y - yn));
154             distance2 := sqrt(sqr(xl - xn) + sqr(yl - yn));
155
156             IF distance2 < 1E-30 THEN
157               sample[excitation_purity] := - 999
158             ELSE
159               sample[excitation_purity] := distance1 / distance2;
160             sample[dominant_wavelength] := domwave;
161             IF complementary THEN
162               sample[dominant_wavelength] := - 1 * sample[dominant_wavelength];
163             END
164           ELSE
165             BEGIN
166             sample[excitation_purity] := - 999;
167             sample[dominant_wavelength] := - 999;
```

```
  168              END;
  169          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_DOMINANT_WAVELENGTH.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_COLOR_XYZ_SOURCE;
 {$nolist}
 {$list}


  PROCEDURE CALC_COLOR_XYZ_SOURCE ((VAR source: spectral_data; observer: ARRAY [low..high: integer] OF spectral_data; VAR
                                    boolean)) ;
{
 WHERE
     source = spectral data for source or illuminant
     observer = array [1..3] of spectral_data
     error = signals error
 RESULTS
     the tristimulus values X,Y,& Z of the light source are calculated and returned
}

    VAR
      coord: coordinate_type;
      temp1: spectral_data;
      temp_error: boolean;
      r: real;

    BEGIN
      error := false;
      error := error OR (observer[low].start <= 100) OR (source.start <= 100);

      IF NOT error THEN
        BEGIN
        { truncate then fold the source to match the observer }
        temp1 := truncate_1_like_2(source, observer[low], temp_error);
        error := error OR temp_error;
        temp1 := fold_1_like_2(temp1, observer[low], temp_error);
        error := error OR temp_error;
        END;

      IF NOT error THEN
        BEGIN
        { calculate the tristimulus values of the source }
        FOR coord := t_x TO t_z DO
          BEGIN
          observer[ord(coord) - ord(t_x) + low] := math(temp1, observer[ord(coord) - ord(t_x) + low], math_multiply, temp
          error := error OR temp_error;
          source.coordinates[coord] := spectral_summation(observer[ord(coord) - ord(t_x) + low],
                                                          observer[ord(coord) - ord(t_x) + low].start,
                                                          observer[ord(coord) - ord(t_x) + low].stop, temp_error);
          error := error OR temp_error;
          END;
        error := error OR (source.coordinates[t_y] = 0);
        IF NOT error THEN
          BEGIN
          r := 100 / source.coordinates[t_y];
          FOR coord := t_x TO t_z DO
            source.coordinates[coord] := source.coordinates[coord] * r;
          END;
        END;
```

```
6D
61          { if an error was encountered then set all parameters calculated by this routine to a value of -999 }
62          IF error THEN
63            FOR coord := t_x TO t_z DO
64              source.coordinates[coord] := - 999;
65
66          { set all other color coordinates to a value of -999 that will later be calculated from the results of this routine
67          FOR coord := c_x TO pred(last) DO
68            source.coordinates[coord] := - 999;
69
7D        END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_XYZ_SOURCE.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_COLOR_XYZ_SOURCE_SAMPLE;
 {$nolist}
 {$list}


  PROCEDURE CALC_COLOR_XYZ_SOURCE_SAMPLE ((source: spectral_data; VAR sample: spectral_data; observer: ARRAY [low..high:
                                          spectral_data;   R error: boolean)} ;
{
 WHERE
     source = spectral data for source or illuminant
     sample = spectral data for a sample
     observer = array [1..3] of spectral_data
     error = signals error
 RESULTS
     the tristimulus values X,Y,& Z of the sample are calculated and returned
}

    VAR
      coord: coordinate_type;
      temp1: spectral_data;
      temp_error: boolean;
      r: real;

    BEGIN
      error := false;
      error := error OR (observer[low].start <= 100) OR (source.start <= 100) OR (sample.start <= 100);

      IF NOT error THEN
        BEGIN
        { truncate then fold the source to match the observer }
        source := truncate_1_like_2(source, observer[low], temp_error);
        error := error OR temp_error;
        source := fold_1_like_2(source, observer[low], temp_error);
        error := error OR temp_error;
        END;

      IF NOT error THEN
        BEGIN
        { calculate the normalizing constant for the source }
        temp1 := math(source, observer[low + 1], math_multiply, temp_error);
        error := error OR temp_error;
        r := spectral_summation(temp1, temp1.start, temp1.stop, temp_error);
        error := error OR temp_error;
        error := error OR (r = 0);
        IF NOT error THEN
          r := 100 / r / source.inc;
        END;

      IF NOT error THEN
        BEGIN
        { truncate then fold the sample to match the observer }
        temp1 := truncate_1_like_2(sample, observer[low], temp_error);
        error := error OR temp_error;
        temp1 := fold_1_like_2(temp1, observer[low], temp_error);
        error := error OR temp_error;
```

```
60              END;
61
62            IF NOT error THEN
63              BEGIN
64              { calculate the tristimulus values of the  sample}
65              FOR coord := t_x TO t_z DO
66                BEGIN
67                observer[ord(coord) - ord(t_x) + low] := math(source, observer[ord(coord) - ord(t_x) + low], math_multiply, tem
68                error := error OR temp_error;
69                observer[ord(coord) - ord(t_x) + low] := math(temp1, observer[ord(coord) - ord(t_x) + low], math_multiply, temp
70                error := error OR temp_error;
71                sample.coordinates[coord] := spectral_summation(observer[ord(coord) - ord(t_x) + low],
72                                                                observer[ord(coord) - ord(t_x) + low].start,
73                                                                observer[ord(coord) - ord(t_x) + low].stop, temp_error);
74                error := error OR temp_error;
75                END;
76              IF NOT error THEN
77                FOR coord := t_x TO t_z DO
78                  sample.coordinates[coord] := sample.coordinates[coord] * r * sample.inc;
79              END;
80
81            { if an error was encountered then set all parameters calculated by this routine to a value of -999 }
82            IF error THEN
83              FOR coord := t_x TO t_z DO
84                sample.coordinates[coord] := - 999;
85
86            { set all other color coordinates to a value of -999 that will later be calculated from the results of this routine
87            FOR coord := c_x TO pred(last) DO
88              sample.coordinates[coord] := - 999;
89
90          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_XYZ_SOURCE_SAMPLE.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_COLOR_XYZ_WEIGHTS;
 {$nolist}
 {$list}


  PROCEDURE CALC_COLOR_XYZ_WEIGHTS ((VAR weights: ARRAY [low..high: integer] OF spectral_data; VAR error: boolean)) ;
{
 WHERE
     weights = array [1..9] of spectral_data
 RESULTS
     the tristimulus values X,Y,& Z of the illuminant are calculated and returned
}

    VAR
      i_wts: integer;
      coord: coordinate_type;
      temp_error: boolean;


    BEGIN
      error := false;

      { clear the tristimulus values of all weights to start with }
      FOR coord := t_x TO t_z DO
        FOR i_wts := low TO high DO
          weights[i_wts].coordinates[coord] := - 999;

      { first try 5, then 10, then 20 nm weights }
      i_wts := 1 - 3;
      REPEAT
        i_wts := i_wts + 3;
      UNTIL (weights[i_wts].start > 100) OR (i_wts = 7);

      error := error OR (weights[i_wts].start <= 100);

      IF NOT error THEN
        BEGIN
        { calculate the tristimulus values of the source }

        FOR coord := t_x TO t_z DO
          BEGIN
          weights[i_wts].coordinates[coord] := spectral_summation(weights[ord(coord) - ord(t_x) + i_wts],
                                                                  weights[ord(coord) - ord(t_x) + i_wts].start,
                                                                  weights[ord(coord) - ord(t_x) + i_wts].stop, temp_error
          error := error OR temp_error;
          END;
        END;

      { if an error was encountered then set all parameters calculated by this routine to a value of -999 }
      IF error THEN
        FOR coord := t_x TO t_z DO
          weights[i_wts].coordinates[coord] := - 999;

      { set all other color coordinates to a value of -999 that will later be calculated from the results of this routine
      FOR coord := c_x TO pred(last) DO
```

```
   60              FOR i_wts := low TO high DO
   61                weights[i_wts].coordinates[coord] := - 999;
   62
   63          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_XYZ_WEIGHTS.PASF;4/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM CALC_COLOR_XYZ_WEIGHTS_SAMPLE;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE CALC_COLOR_XYZ_WEIGHTS_SAMPLE ((weights: ARRAY [low..high: integer] OF spectral_data; VAR sample: spectral_da
11                                                  error: boolean)} ;
12      {
13       WHERE
14           weights = array [1..9] of spectral data
15           sample = spectral data
16           error = signals error
17      RESULTS
18           the tristimulus values X,Y,& Z of the sample are calculated and returned
19      }
20
21          VAR
22            coord: coordinate_type;
23            temp1: spectral_data;
24            temp_error: boolean;
25            i, i_wts: integer;
26
27          BEGIN
28
29            error := false;
30
31            error := error OR (sample.start <= 100) OR NOT (sample.inc IN [5, 10, 20]);
32
33            IF NOT error THEN
34              BEGIN
35              CASE sample.inc OF
36                5:
37                  BEGIN
38                  i_wts := 1;
39                  { sample.calc_type := WTS_5; }
40                  END;
41                10:
42                  BEGIN
43                  i_wts := 4;
44                  { sample.calc_type := WTS_10; }
45                  END;
46                20:
47                  BEGIN
48                  i_wts := 7;
49                  { sample.calc_type := WTS_20; }
50                  END;
51                OTHERWISE
52                  error := true
53                END;
54              error := error OR (weights[i_wts].start <= 100) OR ((sample.start - weights[i_wts].start) MOD sample.inc <> 0);
55              END;
56
57            IF NOT error THEN
58              BEGIN
59              { truncate then fold the spectra to match the weights }
```

```
60              temp1 := truncate_1_like_2(sample, weights[i_wts], temp_error);
61              error := error OR temp_error;
62              temp1 := fold_1_like_2(temp1, weights[i_wts], temp_error);
63              error := error OR temp_error;
64              END;
65
66            IF NOT error THEN
67              BEGIN
68              { calculate the tristimulus values of the sample }
69              FOR coord := t_x TO t_z DO
70                BEGIN
71                weights[ord(coord) - ord(t_x) + i_wts] := math(weights[ord(coord) - ord(t_x) + i_wts], temp1, math_multiply, te
72                error := error OR temp_error;
73                sample.coordinates[coord] := spectral_summation(weights[ord(coord) - ord(t_x) + i_wts],
74                                                                weights[ord(coord) - ord(t_x) + i_wts].start,
75                                                                weights[ord(coord) - ord(t_x) + i_wts].stop, temp_error);
76                error := error OR temp_error;
77                END;
78              END;
79
80            { if an error was encountered then set all parameters calculated by this routine to a value of -999 }
81            IF error THEN
82              BEGIN
83              { sample.calc_type := NO_COLOR_CALC; }
84              FOR coord := t_x TO t_z DO
85                sample.coordinates[coord] := - 999;
86              END;
87
88            { set all other color coordinates to a value of -999 that will later be calculated from the results of this routine
89            FOR coord := c_x TO pred(last) DO
90              sample.coordinates[coord] := - 999;
91
92          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_COLOR_XYZ_WEIGHTS_SAMPLE.PASF;5/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM CALC_CRI;
 ($nolist)
 ($list)


  PROCEDURE CALC_CRI ((source: spectral_data; observer: ARRAY [low..high: integer] OF spectral_data; VAR out: text; make_
                       boolean)) ;
{
 WHERE
     source = spectral data for a test source
     observer = array [1..3] of spectral_data
     out = text file to write results to
     make_report = TRUE for writing to file, FALSE when writing to the terminal screen
 RESULTS
     the color rendering index of a source is calculated and displayed
}

    VAR
      coord: coordinate_type;
      illuminant: spectral_data;
      observer_fn: ARRAY [1..3] OF spectral_data;
      samples: ARRAY [1..14] OF spectral_data;
      coords: ARRAY [1..2, 0..14] OF coordinate_data; { to hold the color coordinates of each patch under both sources th
                                                        of coordinates are for the sources }
      i, j, error_code: integer;
      error, temp_error: boolean;
      denom, cct: real;
      q: char;


    FUNCTION raise(y, x: real): real;

      VAR
        temp: real;

      BEGIN
        IF y > 0 THEN
          temp := exp(x * ln(y))
        ELSE IF y < 0 THEN
          BEGIN
          temp := - 999; { signal error on negative y values }
          error := true;
          END
        ELSE
          temp := 0; { assume 0^x = 0 }
        raise := temp;
      END;


    FUNCTION cube_root(val: real): real;

      BEGIN
        IF val < 0 THEN
          cube_root := - raise( - val, 1 / 3)
        ELSE
```

```
60                cube_root := raise(val, 1 / 3);
61              END;
62
63
64            BEGIN
65              error := false;
66              FOR i := 1 TO 3 DO
67                observer_fn[i] := observer[i];
68              prompt(10);
69              calc_color_xyz_source(source, observer_fn, error);
70              IF NOT error THEN
71                calc_color_chromaticities(source.coordinates, error);
72              IF NOT error THEN
73                cct := calc_cct(source.coordinates[u], source.coordinates[v]);
74              error := error OR NOT ((cct > 500) AND (cct < 25000));
75              IF error THEN
76                BEGIN
77                writec('ERROR calculating color and correlated color temperature of test source');
78                writec('please correct the test source data and try again');
79                END
80              ELSE
81                BEGIN
82                IF cct < 5000 THEN
83                  illuminant := make_blackbody_spectra(cct, error)
84                ELSE
85                  illuminant := make_daylight_spectra(cct, error);
86                IF error THEN
87                  BEGIN
88                  writec('ERROR creating reference illuminant');
89                  writeln(' ': 10, ' Correlated color temperature of test source :  ', cct: 9: 2);
90                  writec('please correct the test source data and try again');
91                  END
92                ELSE
93                  BEGIN
94                  IF regis_terminal AND NOT make_report THEN
95                    BEGIN
96                    q := 'Y';
97                    readch('plot the source and reference illuminant on the screen (Y,N) ?', q, use_default, null_ok);
98                    IF q <> 'N' THEN
99                      BEGIN
100                     temp_error := false;
101                     { normalize the spectra to a value of 100 at 560 nm if possible }
102                     IF source.val[(560 - source.start) DIV source.inc + 1] <> 0 THEN
103                       samples[1] := math_constant(source, 100 / source.val[(560 - source.start) DIV source.inc + 1],
104                                                   math_constant_multiply, temp_error)
105                     ELSE
106                       samples[1] := source;
107                     IF temp_error THEN
108                       samples[1] := source;
109                     samples[2] := illuminant;
110                     samples[2].descrip := 'REFERENCE ILLUMINANT                    ';
111                     samples[2].name := '          ';
112                     plotrg_mspectra('SOURCE and REFERENCE ILLUMINANT', samples, 1, 2);
113                     END;
```

```
                prompt(10);
                END;

              stass(samples[1].filename, 'CLR:MUNSELL.DAT');
              retrieve_mspectra(samples, 1, 1, 14, error_code);
              say_error_code('ERROR reading CLR:MUNSELL.DAT spectral files', error_code, true);
              error := error_code <> 0;
              END;
            END;
          IF NOT error THEN
            BEGIN
            calc_color_xyz_source(illuminant, observer_fn, error);
            IF NOT error THEN
              calc_color_chromaticities(illuminant.coordinates, error);
            IF error THEN
              BEGIN
              writec('ERROR in calculating color coordinates of reference illuminant');
              writec('please correct the test source data and try again');
              END;
            END;
          IF NOT error THEN
            BEGIN
        { start loading coords array ===>
          format = [1 = test source or 2 = reference illuminant, 0 = coords of source   1 - 14 = coords of samples]   }
            coords[1, 0] := source.coordinates;
            coords[2, 0] := illuminant.coordinates;
            FOR i := 1 TO 14 DO
              BEGIN
              { calculate color of samples under test source }
              calc_color_xyz_source_sample(source, samples[i], observer_fn, temp_error);
              IF NOT temp_error THEN
                calc_color_chromaticities(samples[i].coordinates, temp_error);
              error := error OR temp_error;
              coords[1, i] := samples[i].coordinates;

              { calculate color of samples under reference illuminant }
              calc_color_xyz_source_sample(illuminant, samples[i], observer_fn, temp_error);
              IF NOT temp_error THEN
                calc_color_chromaticities(samples[i].coordinates, temp_error);
              error := error OR temp_error;
              coords[2, i] := samples[i].coordinates;
              END;
            IF error THEN
              BEGIN
              writec('ERROR in calculating color coordinates of Munsell Patches');
              writec('please try again');
              END;
            END;
          IF NOT error THEN
            BEGIN
            { load calculations of "c" and "d" in array indexes a_star and b_star respectively }
            { for the samples viewed under the test source }
            FOR j := 1 TO 14 DO
              BEGIN
```

```
168               coords[1, j][a_star] := (4 - coords[1, j][u] - 10 * coords[1, j][v]) / coords[1, j][v];
169               coords[1, j][b_star] := (1.708 * coords[1, j][v] + 0.404 - 1.481 * coords[1, j][u]) / coords[1, j][v];
170               END;
171
172             { calculate values of "c" and "d" for the reference and illuminant }
173             FOR i := 1 TO 2 DO
174               BEGIN
175               coords[i, 0][a_star] := (4 - coords[i, 0][u] - 10 * coords[i, 0][v]) / coords[i, 0][v];
176               coords[i, 0][b_star] := (1.708 * coords[i, 0][v] + 0.404 - 1.481 * coords[i, 0][u]) / coords[i, 0][v];
177               END;
178
179             { and ratio the source values to c/d since they will always be used in this way }
180             coords[1, 0][a_star] := coords[2, 0][a_star] / coords[1, 0][a_star];
181             coords[1, 0][b_star] := coords[2, 0][b_star] / coords[1, 0][b_star];
182
183           { now calculate the chromaticity coordinates u' and v' for the samples viewed under the test source considering the
184             adaptive color shift due to chromaticity differences between the test source and reference illuminant }
185
186             FOR i := 1 TO 14 DO
187               BEGIN
188               denom := 16.518 + 1.481 * coords[1, 0][a_star] * coords[1, i][a_star] - coords[1, 0][b_star] * coords[1, i][b_s
189
190               coords[1, i][u] := (10.872 + 0.404 * coords[1, 0][a_star] * coords[1, i][a_star] - 4 * coords[1, 0][b_star] *
191                                   coords[1, i][b_star]) / denom;
192
193               coords[1, i][v] := 5.520 / denom;
194               END;
195
196           { calculate W* u* and v* for each sample viewed under the test source and reference illuminant
197               (using array indexes of L_start, u_star, and v_star respectively ) }
198             FOR i := 1 TO 2 DO
199               FOR j := 1 TO 14 DO
200                 BEGIN
201                 coords[i, j][l_star] := 25 * cube_root(coords[i, j][t_y]) - 17;
202                 coords[i, j][u_star] := 13 * coords[i, j][l_star] * (coords[i, j][u] - coords[2, 0][u]);
203                 coords[i, j][v_star] := 13 * coords[i, j][l_star] * (coords[i, j][v] - coords[2, 0][v]);
204                 END;
205
206             { calculate the color differences and place the result in index E_uv }
207             FOR j := 1 TO 14 DO
208               coords[1, j][E_uv] := sqrt(sqr(coords[2, j][l_star] - coords[1, j][l_star]) + sqr(coords[2, j][u_star] -
209                                                                                                 coords[1, j][u_star]) +
210                                                 sqr(coords[2, j][v_star] - coords[1, j][v_star]));
211
212             { calculate the special color rendering indices and place the result in index E_ab }
213             FOR j := 1 TO 14 DO
214               coords[1, j][E_ab] := 100 - 4.6 * coords[1, j][E_uv];
215
216             denom := 0;
217             FOR j := 1 TO 8 DO
218               denom := denom + round(coords[1, j][E_ab]);
219             denom := denom / 8;
220
221             IF make_report THEN
```

```
222              prompt(2);
223            writeln(out, ' ': 24, 'SPECIAL COLOR RENDERING INDICES');
224            write(out, 'TEST SOURCE :  ');
225            display_descrip(out, source, 63, false, true, true);
226            writeln(out);
227            writeln(out);
228            writeln(out, '         Sample    Munsell        Color Appearance         Special CRI');
229            writeln(out, '         Number   Notation         Under Daylight                    ');
230            writeln(out, '         ------- ----------   --------------------------   -----------');
231            writeln(out, '            1     7.5R 6/4     Light greyish red        ', round(coords[1, 1][E_ab]): 10);
232            writeln(out, '            2      5Y 6/4      Dark greyish yellow      ', round(coords[1, 2][E_ab]): 10);
233            writeln(out, '            3     5GY 6/8      Strong yellow green      ', round(coords[1, 3][E_ab]): 10);
234            writeln(out, '            4     2.5G 6/6     Moderate yellowish green ', round(coords[1, 4][E_ab]): 10);
235            writeln(out, '            5     10BG 6/4     Light bluish green       ', round(coords[1, 5][E_ab]): 10);
236            writeln(out, '            6     5PB 6/8      Light blue               ', round(coords[1, 6][E_ab]): 10);
237            writeln(out, '            7     2.5P 6/8     Light violet             ', round(coords[1, 7][E_ab]): 10);
238            writeln(out, '            8     10P 6/8      Light reddish purple     ', round(coords[1, 8][E_ab]): 10);
239            writeln(out, '            9    6.5R 4/13     Strong red               ', round(coords[1, 9][E_ab]): 10);
240            writeln(out, '           10     5Y 8/10      Strong yellow            ', round(coords[1, 10][E_ab]): 10);
241            writeln(out, '           11     4.5G 5/8     Strong green             ', round(coords[1, 11][E_ab]): 10);
242            writeln(out, '           12     3PB 3/11     Strong blue              ', round(coords[1, 12][E_ab]): 10);
243            writeln(out, '           13     5YR 8/4      Light yellowish pink     ', round(coords[1, 13][E_ab]): 10);
244            writeln(out, '           14     5GY 4/4      Moderate olive green     ', round(coords[1, 14][E_ab]): 10);
245            writeln(out);
246            writeln(out, ' ': 22, ' GENERAL COLOR RENDERING INDEX :  ', round(denom): 10);
247            END;
248          IF NOT make_report THEN
249            return(1);
250        END;
```

```
PROGRAM CALC_DIFFERENCE;
 {$nolist}
 {$list}


  FUNCTION CALC_DIFFERENCE ((sample1, sample2: coordinate_data): coordinate_data) ;
{
 WHERE
     sample1 = coordinate data of sample 1
     sample2 = coordinate data of sample 2
 RESULTS
     the differences of color coordinates (and color differences) are calculated and returned
}

    VAR
      coord, coord1: coordinate_type;
      coords: coordinate_data;
      error: boolean;


    BEGIN
      FOR coord := succ(first) TO pred(last) DO
        BEGIN
        coords[coord] := - 999;
        CASE coord OF
          E_ab:
            BEGIN
            error := false;
            FOR coord1 := l_star TO b_star DO
              error := error OR (calc_color_bad_value(sample1[coord1]) OR calc_color_bad_value(sample2[coord1]));
            IF NOT error THEN
              BEGIN
              coords[E_ab] := 0;
              FOR coord1 := l_star TO b_star DO
                coords[E_ab] := coords[E_ab] + sqr(sample1[coord1] - sample2[coord1]);
              coords[E_ab] := sqrt(coords[E_ab]);
              END;
            END;
          E_uv:
            BEGIN
            error := (calc_color_bad_value(sample1[l_star]) OR calc_color_bad_value(sample2[l_star]));
            FOR coord1 := u_star TO v_star DO
              error := error OR (calc_color_bad_value(sample1[coord1]) OR calc_color_bad_value(sample2[coord1]));
            IF NOT error THEN
              BEGIN
              coords[E_uv] := sqr(sample1[l_star] - sample2[l_star]);
              FOR coord1 := u_star TO v_star DO
                coords[E_uv] := coords[E_uv] + sqr(sample1[coord1] - sample2[coord1]);
              coords[E_uv] := sqrt(coords[E_uv]);
              END;
            END;
          h_ab: ; { don't calculate this for now }
          h_uv: ; { don't calculate this for now }
          OTHERWISE
            IF NOT (calc_color_bad_value(sample1[coord]) OR calc_color_bad_value(sample2[coord])) THEN
```

```
   60                    coords[coord] := sample1[coord] - sample2[coord]
   61                END;
   62              END;
   63
   64            { return calculated values }
   65            CALC_DIFFERENCE := coords;
   66          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_DIFFERENCE.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM CALC_DT_SPECTRA;
2        ($nolist)
7        ($list)
8
9
10        FUNCTION CALC_DT_SPECTRA ((source: spectral_data; VAR error: boolean; show_calc: boolean): real) ;
11      {
12       WHERE
13           source = spectra of light source to calculate distribution temperature from
14           error = signals error
15        RESULTS
16           the distribution temperature of the source
17      }
18
19          CONST
20            max_number_of_tries = 200;
21
22          VAR
23            t, t_inc, old_sum, very_old_sum, sum, sum1, sum2: real;
24            sources: ARRAY [1..4] OF spectral_data; { 1 = user's source, 2 = Planckian radiator with the same DT as user's sour
25                                                    ratio of the first two sources for intermediate calculations 4 = square of
26                                                    the first two sources for intermediate calculations }
27
28            number_of_tries, i, j: integer;
29            test, testsm: ARRAY [1..max_number_of_tries] OF real;
30            blackbody_error: boolean;
31            q: char;
32
33
34          BEGIN
35
36            { setup the initial conditions }
37            { normalize the spectra to a value of 100 at 560 nm if possible }
38            blackbody_error := false;
39            IF source.val[(560 - source.start) DIV source.inc + 1] <> 0 THEN
40              sources[1] := math_constant(source, 100 / source.val[(560 - source.start) DIV source.inc + 1], math_constant_mult
41                                          blackbody_error)
42            ELSE
43              sources[1] := source;
44            IF blackbody_error THEN
45              sources[1] := source;
46
47            t := 25000.0;
48            number_of_tries := 0;
49            t_inc := 1000.0;
50            old_sum := 1.0E8;
51            sum := 1.0E7;
52            error := false;
53            blackbody_error := false;
54
55            j := 0;
56            REPEAT
57              REPEAT
58                j := j + 1;
59                test[j] := t;
```

```
60                number_of_tries := number_of_tries + 1;
61                sum1 := 0;
62                sum2 := 0;
63
64                { make the blackbody for this temperature }
65                sources[2] := make_blackbody_spectra(t, blackbody_error);
66                say_error('ERROR CREATING A BLACKBODY', blackbody_error, true);
67
68                sources[3] := math(sources[1], sources[2], math_divide_1_by_2, blackbody_error);
69                say_error('ERROR DIVIDING SPECTRA', blackbody_error, true);
70
71                sum1 := spectral_summation(sources[3], sources[3].start, sources[3].stop, blackbody_error);
72                say_error('ERROR SUMMING RATIOS', blackbody_error, true);
73
74                sources[4] := math_constant(sources[3], 2, math_constant_raise, blackbody_error);
75                say_error('ERROR SQUARING RATIOS', blackbody_error, true);
76
77                sum2 := spectral_summation(sources[4], sources[4].start, sources[4].stop, blackbody_error);
78                say_error('ERROR SUMMING SQUARED RATIOS', blackbody_error, true);
79
80                sum1 := sum1 / sum2;
81                very_old_sum := old_sum;
82                old_sum := sum;
83                sum := 0;
84
85                WITH sources[3] DO
86                  FOR i := 1 TO (stop - start) DIV inc + 1 DO
87                    sum := sum + sqr(1 - val[i] * sum1);
88
89                IF show_calc THEN
90                  writeln(' ': 10, 'T = ', t: 10: 4, 'K    RMS Error = ', sum);
91                t := t - t_inc;
92                testsm[j] := sum;
93              UNTIL (sum > old_sum) OR ((sum = old_sum) AND (sum = very_old_sum)) OR (sum < 2.0E-7);
94              IF sum > old_sum THEN
95                BEGIN
96                IF j > 3 THEN
97                  BEGIN
98                  t := test[j - 2];
99                  sum := testsm[j - 2];
100                 old_sum := testsm[j - 3];
101                 END
102               ELSE
103                 BEGIN
104                 t := test[1];
105                 sum := testsm[1];
106                 old_sum := 1.0E9;
107                 END;
108               j := 1;
109               test[1] := t;
110               testsm[1] := sum;
111               t_inc := t_inc / 2;
112               t := t - t_inc;
113               END
```

```
114             ELSE
115               t := t + t_inc;
116           UNTIL (number_of_tries = max_number_of_tries) OR (t_inc < 0.1) OR ((sum = old_sum) AND (sum = very_old_sum)) OR
117                 (sum < 2.0E-7);
118           IF show_calc THEN
119             BEGIN
120             writeln;
121             writeln;
122             writeln('Distribution Temperature :  ', round(t): 7, ' K');
123             writeln;
124             writeln;
125             writeln('      Source Description :  ', source.descrip);
126             IF regis_terminal THEN
127               BEGIN
128               q := 'Y';
129               writeln;
130               writeln('Your source can be compared to a ', round(t): 7, ' K blackbody spectra');
131               writeln;
132               readch('Plot the spectra for comparison on your terminal ?', q, use_default, null_ok);
133               IF q = 'Y' THEN
134                 plotrg_mspectra('SOURCE COMPARED TO BLACKBODY PLOT', sources, 1, 2);
135               END
136             ELSE
137               BEGIN
138               writeln;
139               return(1);
140               END;
141             END;
142           CALC_DT_SPECTRA := t;
143         END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_DT_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM CALC_LAGRANGE;
 {$nolist}
 {$list}


  PROCEDURE CALC_LAGRANGE ((VAR bspec: big_spectral_data; spec: spectral_data; VAR error: boolean)} ;
{
 WHERE
     bspec = a spectral data record to hold the 1 nm data
     spec = a spectral data record containing the data to interpolate
     error = signals an error in the routine
 RESULTS
     bspec is loaded with 1 nm interpolated data from spec
}

{ this procedure calculates the Lagrange coefficeints for the ASTM weight
  calculations using a reference by Hugh Fairman, "The Calculation of
  Weight Factors for Tristimulus Integration", CR&A, Vol. 10, No. 4, pp.
  199 - 203 }

    TYPE
      coef_3pt = ARRAY [0..2, 1..19] OF real;
      coef_4pt = ARRAY [0..3, 1..19] OF real;

    VAR
      pt3: coef_3pt;
      pt4: coef_4pt;
      i, j: integer;
      r: real;


    PROCEDURE calc_values(p0, typ: integer);

      VAR
        i, j: integer;

      BEGIN
        { this routine is only valid for wavelength increments of 5, 10, and 20 nm }
        IF typ = 1 THEN
          BEGIN
          { use three points }
          FOR i := 1 TO spec.inc - 1 DO
            BEGIN
            bspec.val[p0 + i] := 0;
            FOR j := 0 TO 2 DO
              bspec.val[p0 + i] := bspec.val[p0 + i] + pt3[j, i * 20 DIV spec.inc] * bspec.val[p0 + j * spec.inc];
            END;
          END
        ELSE IF typ = 3 THEN
          BEGIN
          { use three points }
          FOR i := 1 TO spec.inc - 1 DO
            BEGIN
            bspec.val[p0 + spec.inc + i] := D;
            FOR j := 0 TO 2 DO
```

```
60                    bspec.val[p0 + spec.inc + i] := bspec.val[p0 + spec.inc + i] + pt3[2 - j, (spec.inc - i) * 20 DIV spec.inc]
61                                                    bspec.val[p0 + j * spec.inc];
62                  END;
63                END
64              ELSE
65                BEGIN
66                { use four points }
67                FOR i := 1 TO spec.inc - 1 DO
68                  BEGIN
69                  bspec.val[p0 + spec.inc + i] := 0;
70                  FOR j := 0 TO 3 DO
71                    bspec.val[p0 + spec.inc + i] := bspec.val[p0 + spec.inc + i] + pt4[j, i * 20 DIV spec.inc] * bspec.val[p0 +
72                                                    spec.inc];
73                  END;
74                END;
75            END;
76
77
78          BEGIN
79            { zero out the bspec record }
80            FOR i := starting_wavelength TO ending_wavelength DO
81              bspec.val[i] := 0;
82            stass(bspec.filename, '  ');
83            stass(bspec.descrip, '  ');
84            error := NOT (spec.inc IN [5, 10, 20]);
85            error := error OR (spec.start < starting_wavelength) OR (spec.stop > ending_wavelength);
86            IF NOT error THEN
87              BEGIN
88              { calculate the coefficeints for the first interval }
89              FOR i := 1 TO 19 DO
90                BEGIN
91                r := i / 20;
92                pt3[0, i] := (r - 1) * (r - 2) / 2;
93                pt3[1, i] := - r * (r - 2);
94                pt3[2, i] := r * (r - 1) / 2;
95                END;
96
97              { calculate the coefficeints for intermediate intervals }
98              FOR i := 1 TO 19 DO
99                BEGIN
100               r := 1 + i / 20;
101               pt4[0, i] := - (r - 1) * (r - 2) * (r - 3) / 6;
102               pt4[1, i] := r * (r - 2) * (r - 3) / 2;
103               pt4[2, i] := - r * (r - 1) * (r - 3) / 2;
104               pt4[3, i] := r * (r - 1) * (r - 2) / 6;
105               END;
106
107             { load data from spec into bspec }
108             FOR i := 1 TO (spec.stop - spec.start) DIV spec.inc + 1 DO
109               bspec.val[spec.start + (i - 1) * spec.inc] := spec.val[i];
110
111             calc_values(spec.start, 1); { first wavelength to interpolate }
112             calc_values(spec.stop - 2 * spec.inc, 3); { last wavelength to interpolate }
113
```

```
114             { remainder of wavelengths to interpolate }
115             i := spec.start;
116             REPEAT
117               calc_values(i, 2);
118               i := i + spec.inc;
119             UNTIL i >= spec.stop - 2 * spec.inc;
120             { fold the longer wavelengths }
121             FOR j := spec.stop + 1 TO 830 DO
122               bspec.val[j] := bspec.val[i - spec.inc];
123             { fold the shorter wavelengths }
124             FOR j := starting_wavelength TO spec.start - 1 DO
125               bspec.val[j] := spec.val[1];
126             bspec.descrip := spec.descrip;
127             bspec.filename := spec.filename;
128             END
129           ELSE
130             BEGIN
131             jump(1);
132             writec('ERROR in CALC_LAGRANGE');
133             writec('spectra empty or wavelength limits or increment not correct');
134             return(1);
135             END;
136         END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALC_LAGRANGE.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
Pascal-2 VAX/VMS 2.2B    6-Aug-89   7:29 PM    Site #1-1    Page 1-1
Oregon Software, Inc.  6915 SW Macadam Ave.  Suite # 200  Portland  OR  97219  USA
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALL_HELP.PASF;3

     1      PROGRAM CALL_HELP;
     2       {$nolist}
     7       {$list}
     8
     9
    10        PROCEDURE CALL_HELP ((filename: PACKED ARRAY [low..high: integer] OF char)) ;
    11      {
    12       WHERE
    13           filename = a string containing the filename of a help library file specification
    14        RESULTS
    15           help information as requested by the user for the specified help library
    16      }
    17
    18          VAR
    19            descrip: descriptor_block;
    20            ss: PACKED ARRAY [1..70] OF char;
    21
    22          BEGIN
    23            strass(ss, filename);
    24            descrip := make_descriptor_block(ss, 70);
    25            gotoxy(0, 0);
    26            erase(eoscreen);
    27            break(output);
    28            helper(descrip);
    29          END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CALL_HELP.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
1       PROGRAM CHECK_SPECTRAL_FILE;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE CHECK_SPECTRAL_FILE ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR descrip: lstring; VAR start, stop,
11                                        number_of_spectra_in_file: integer; VAR error_code: integer)} ;
12      {
13       WHERE
14           s = name of data file
15           descrip = description line from file
16           start = starting wavelength
17           stop = ending wavelength
18           inc = wavelength increment
19           number_of_spectra_in_file = number of spectra in file
20           error_code = error code that is passed back to calling routine
21       RESULTS
22           an error_code is returned indicating any errors in reading all spectra from the file
23      }
24
25          CONST
26            max_reals_converted_per_line = 100;
27
28          VAR
29            number_of_data_lines, chr_ptr, index, getsub_code, rval_code, i, len, num, skip_before, skip_after: integer;
30            inp: text;
31            delim: char;
32            temp: ARRAY [1..max_reals_converted_per_line] OF real; { temp holds real numbers converted from a line read from a
33            line, sub_line: STRING [255];
34            temp_name: sstring;
35            error, eos: boolean;
36            ss: STRING [255];
37
38
39          PROCEDURE write_string(s: PACKED ARRAY [slow..shigh: integer] OF char);
40
41            VAR
42              i: integer;
43
44            BEGIN
45              IF slow = 0 THEN
46                FOR i := 1 TO ord(s[0]) DO
47                  write(s[i])
48              ELSE IF slow = 1 THEN
49                FOR i := slow TO shigh DO
50                  write(s[i])
51              ELSE
52                write('WRITE_STRING-ERROR Low string array index not in [0,1]');
53            END;
54
55
56          BEGIN
57
58            strass(ss, s);
59
```

```
60              error_code := 0; { everything's ok to start with }
61              start := 0;
62              stop := 0;
63              inc := 0;
64              number_of_spectra_in_file := 0;
65
66              IF spectra_debug THEN
67                BEGIN
68                write('%CHECK_SPECTRA_FILE-INFO filename = ');
69                write_string(ss);
70                writeln;
71                spectra_debug_pause_execution;
72                END;
73
74              IF error_code = 0 THEN
75                BEGIN
76                len := 0;
77                reset(inp, ss, '.OAT', len); { open the file }
78                IF ioerror(inp) THEN
79                  error_code := - 4;
80                END;
81
82              IF error_code = 0 THEN
83                BEGIN
84                noioerror(inp);
85                readln(inp, descrip); { read description of data }
86                IF spectra_debug THEN
87                  BEGIN
88                  writeln('%CHECK_SPECTRA_FILE-INFO descrip =', descrip);
89                  spectra_debug_pause_execution;
90                  END;
91                IF ioerror(inp) THEN
92                  error_code := - 5;
93                END;
94
95              IF error_code = 0 THEN
96                BEGIN
97                readln(inp, line); { read the line that should contain start, stop, inc, & number of spectra }
98                IF spectra_debug THEN
99                  BEGIN
100                 writeln('%CHECK_SPECTRA_FILE-INFO wavelength line=', line);
101                 spectra_debug_pause_execution;
102                 END;
103               IF ioerror(inp) THEN
104                 error_code := - 6
105               ELSE
106                 BEGIN
107                 i := 0;
108                 chr_ptr := 1;
109                 eos := false;
110                 error := false;
111                 WHILE NOT eos AND NOT error AND (i <= max_reals_converted_per_line) DO
112                   BEGIN
113                   i := i + 1;
```

```
114                   getsub(line, [' ', ','], chr_ptr, sub_line, delim, getsub_code, eos);
115
116                   IF spectra_debug THEN
117                     BEGIN
118                     writeln('%CHECK_SPECTRA_FILE-INFO getsub sub line=', sub_line);
119                     spectra_debug_pause_execution;
120                     END;
121
122                   error := error OR (getsub_code <> 0);
123                   rval(sub_line, temp[i], rval_code);
124                   error := error OR (rval_code <> 0);
125                   IF error THEN
126                     error_code := - 6;
127                   END;
128                 IF error_code = 0 THEN
129                   BEGIN
130                   IF i = 3 THEN
131                     temp[4] := 1; { if number_of_spectra not specified assume 1 }
132                   IF NOT (i IN [3, 4]) THEN
133                     error_code := - 7;
134                   END;
135
136                 IF error_code = 0 THEN
137                   BEGIN
138                   start := round(temp[1]);
139                   stop := round(temp[2]);
140                   inc := round(temp[3]);
141                   number_of_spectra_in_file := round(temp[4]);
142                   IF (start < 200) OR (stop > 2000) OR (stop <= start) OR (abs(start - starting_wavelength) MOD inc <> 0) OR
143                      (abs(stop - ending_wavelength) MOD inc <> 0) OR NOT (inc IN [5, 10, 20]) THEN
144                     error_code := - 6;
145
146                   IF error_code = 0 THEN
147                     BEGIN
148                     number_of_data_lines := (stop - start) DIV inc + 1;
149                     IF start < starting_wavelength THEN
150                       BEGIN
151                       skip_before := (starting_wavelength - start) DIV inc;
152                       start := starting_wavelength;
153                       END
154                     ELSE
155                       skip_before := 0;
156                     IF stop > ending_wavelength THEN
157                       BEGIN
158                       skip_after := (stop - ending_wavelength) DIV inc;
159                       stop := ending_wavelength;
160                       END
161                     ELSE
162                       skip_after := 0;
163                     IF spectra_debug THEN
164                       BEGIN
165                       writeln('%CHECK_SPECTRA_FILE-INFO start = ', start);
166                       writeln('%CHECK_SPECTRA_FILE-INFO stop = ', stop);
167                       writeln('%CHECK_SPECTRA_FILE-INFO inc = ', inc);
```

```
168                     writeln('%CHECK_SPECTRA_FILE-INFO number of data lines = ', number_of_data_lines);
169                     spectra_debug_pause_execution;
170                     END;
171                   END;
172                 END;
173               END;
174
175             IF error_code = 0 THEN
176               BEGIN
177               readln(inp, line); { read the line containing space delimited descriptions of the columns of data }
178               IF spectra_debug THEN
179                 BEGIN
180                 writeln('%CHECK_SPECTRA_FILE-INFO name line=', line);
181                 spectra_debug_pause_execution;
182                 END;
183               IF ioerror(inp) THEN
184                 error_code := - 10
185               ELSE
186                 BEGIN
187                 i := 0;
188                 chr_ptr := 1;
189                 eos := false;
190                 error := false;
191                 WHILE NOT eos AND NOT error AND (i <= max_reals_converted_per_line) DO
192                   BEGIN
193                   i := i + 1;
194                   getsub(line, [' '], chr_ptr, temp_name, delim, getsub_code, eos);
195
196                   IF spectra_debug THEN
197                     BEGIN
198                     writeln('%CHECK_SPECTRA_FILE-INFO sub line=', temp_name);
199                     spectra_debug_pause_execution;
200                     END;
201                   error := error OR (getsub_code <> 0);
202                   END;
203                 END;
204               END;
205
206             IF error_code = 0 THEN
207               BEGIN
208               num := 0; { num is the line number of data being processed }
209               REPEAT
210                 readln(inp, line); { read each row of data }
211                 IF spectra_debug THEN
212                   BEGIN
213                   writeln('%CHECK_SPECTRA_FILE-INFO data line=', line);
214                   spectra_debug_pause_execution;
215                   END;
216                 IF ioerror(inp) THEN
217                   error_code := - 11
218                 ELSE
219                   BEGIN
220                   num := num + 1;
221
```

```
222                 i := 0;
223                 chr_ptr := 1;
224                 eos := false;
225                 error := false;
226                 WHILE NOT eos AND NOT error AND (i <= number_of_spectra_in_file) DO { process the line reading ALL data }
227                   BEGIN
228                   i := i + 1;
229                   getsub(line, [' ', ','], chr_ptr, sub_line, delim, getsub_code, eos);
230                   IF spectra_debug THEN
231                     BEGIN
232                     writeln('%CHECK_SPECTRA_FILE-INFO sub data line=', sub_line);
233                     spectra_debug_pause_execution;
234                     END;
235                   IF getsub_code <> 0 THEN
236                     error_code := - 13; { data point missing }
237                   rval(sub_line, temp[i], rval_code);
238                   IF rval_code <> 0 THEN
239                     error_code := - 12; { error converting string to real }
240                   error := error_code <> 0;
241                   END;
242                 IF error_code = 0 THEN
243                   IF i <> number_of_spectra_in_file THEN
244                     error_code := - 13;
245                 END;
246               UNTIL (error_code <> 0) OR (num = number_of_data_lines);
247               END;
248           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CHECK_SPECTRAL_FILE.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM CORRECT_SPECTRA;
2        ($nolist)
7        ($list)
8
9
10        PROCEDURE CORRECT_SPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: spectral_data)) ;
11      {
12       WHERE
13           s = string to display
14           spec = spectral data record to correct
15        RESULTS
16           a spectral data record of manually corrected data
17      }
18
19          CONST
20            x_offset = 4; { these offsets are used by CURMAT to position the wavelength values on the screen }
21            y_offset = 5;
22            prompt_line = 21;
23
24          VAR
25            ss: STRING [255];
26            temp: spectral_data;
27            i, j, k, error_code: integer;
28
29            q: char;
30
31
32          BEGIN
33            { work on a copy of spec until the end in case someone aborts before the end }
34            temp := spec;
35            WITH temp DO
36              BEGIN
37              prompt(1);
38              strass(ss, s);
39              writec(ss);
40              jump(3);
41              writec('To leave the descriptions unchanged');
42              writec('press RETURN in response to the following two questions');
43              jump(2);
44              readst('enter a short description of these data (no intermediate spaces)', name, use_default, no_null);
45              jump(1);
46
47              { trim the string after the first space }
48              i := 0;
49              REPEAT
50                i := i + 1;
51              UNTIL (i = 10) OR (name[i] = ' ');
52              IF i < 10 THEN
53                FOR i := i TO 10 DO
54                  name[i] := ' ';
55
56              readst('enter a long description of these data', descrip, use_default, no_null);
57
58              i := 0;
59              REPEAT
```

```
60              prompt(1);
61              writec(ss);
62              write('Description :  ');
63              display_descrip(output, temp, 63, false, true, true);
64              writeln;
65              writeln('Wavelength range (nm) :  ', start: 0, ' to ', stop: 0, ' every ', inc: 0);
66              j := i;
67              REPEAT
68                i := i + 1;
69                curmat(i MOD 46 + i DIV 46, 3, 15, x_offset, y_offset, vertical);
70                writeln(' ': 3, start + (i - 1) * inc: 0, ': ', val[i]: 15);
71              UNTIL (i - j = 45) OR (i = (stop - start) DIV inc + 1);
72
73              REPEAT
74                gotoxy(0, prompt_line);
75                erase(eoscreen);
76                readin('enter wavelength (nm) to correct -or- 0 to finish', k, no_default);
77                gotoxy(0, prompt_line);
78                erase(eoscreen);
79                { check value }
80                IF k <> 0 THEN
81                  BEGIN
82                  IF (k < start) OR (k > stop) OR ((k - start) MOD inc <> 0) THEN
83                    BEGIN
84                    writec('BAD WAVELENGTH VALUE -- Please try again');
85                    return(0);
86                    END
87                  ELSE
88                    BEGIN
89                    writeln('Current Wavelength :  ', k: 0, ' nm');
90                    j := (k - start) DIV inc + 1;
91                    readre('enter correct value', val[j], use_default);
92                    curmat(j MOD 46 + j DIV 46, 3, 15, x_offset, y_offset, vertical);
93                    write(' ': 3, k: 0, ': ', val[j]: 15);
94                    END;
95                  END;
96              UNTIL k = 0;
97            UNTIL (i = (stop - start) DIV inc + 1);
98            gotoxy(0, prompt_line);
99            erase(eoscreen);
100           q := 'N';
101           readch('Save the corrected data to disk ?', q, use_default, null_ok);
102           IF q = 'Y' THEN
103             BEGIN
104             error_code := 0;
105             REPEAT
106               gotoxy(0, prompt_line);
107               erase(eoscreen);
108               say_error_code('ERROR WRITING FILE, TRY AGAIN', error_code, false);
109               readst('enter a filename for these data', filename, use_default, no_null);
110               save_spectra(temp, error_code);
111             UNTIL error_code = 0;
112             END;
113           END;
```

```
  114             spec := temp;
  115           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]CORRECT_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM DISPLAY_DESCRIP;
    2        ($nolist)
    7        ($list)
    8
    9
   10         PROCEDURE DISPLAY_DESCRIP ((VAR out: text; spec: spectral_data; length: integer; typ, long, short: boolean)) ;
   11       {
   12        WHERE
   13            out = file variable for output location
   14            spec =  spectra data record
   15            length =  length of line that should be written
   16            typ = TRUE to display coordinate typ in front of description
   17            long = TRUE to display the DESCRIPTION of the spectral data
   18            short = TRUE to display the NAME of the spectral data
   19        RESULTS
   20            the description is written to the desired file
   21       }
   22
   23           VAR
   24             i, ii, len, used: integer;
   25             s: PACKED ARRAY [1..10] OF char;
   26             null: char;
   27
   28           BEGIN
   29             null := chr(0);
   30             used := 0;
   31             IF typ THEN
   32               BEGIN
   33               CASE spec.start OF
   34                 0:
   35                   s := '  <empty> ';
   36                 1:
   37                   s := 'COORDS  : ';
   38                 OTHERWISE
   39                   s := 'SPECTRA : '
   40                 END;
   41               IF length < 10 THEN
   42                 len := length
   43               ELSE
   44                 len := 10;
   45               FOR i := 1 TO len DO
   46                 write(out, s[i]); { write the description }
   47               used := len;
   48               END;
   49             IF long THEN
   50               BEGIN
   51               i := stringmax;
   52               WHILE (i > 1) AND (spec.descrip[i] IN [null, ' ']) DO
   53                 i := i - 1; { figure out how many trailing blanks are present }
   54               IF spec.descrip[i] IN [null, ' '] THEN
   55                 i := i - 1;
   56               IF i + used > length THEN
   57                 len := length - used
   58               ELSE
   59                 len := i;
```

```
60              FOR i := 1 TO len DO
61                write(out, spec.descrip[i]); { write the description }
62              used := used + len;
63              END;
64            IF short THEN
65              BEGIN
66              IF used < length THEN
67                BEGIN
68                IF used <= length - 2 THEN
69                  BEGIN
70                  write(out, '/ '); { write "/ " to seperate the descrip and name fields }
71                  used := used + 2;
72                  END
73                ELSE
74                  BEGIN
75                  FOR i := used + 1 TO length DO
76                    write(out, ' ');
77                  used := length;
78                  END;
79                END;
80              IF used <> length THEN { only write out the name field if there is enough room }
81                BEGIN
82                ii := 10; { length of name field in spectra_data record }
83                WHILE (ii > 1) AND (spec.name[ii] IN [null, ' ']) DO
84                  ii := ii - 1; { figure out how many trailing blanks are present }
85                IF spec.name[ii] IN [null, ' '] THEN
86                  ii := ii - 1;
87                IF used + ii > length THEN
88                  ii := length - used;
89                FOR i := 1 TO ii DO
90                  write(out, spec.name[i]); { write the name }
91                used := used + ii;
92                END;
93              END;
94            FOR i := used + 1 TO length DO
95              write(out, ' '); { write trailing spaces to fill up the length specified in the call }
96          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]DISPLAY_DESCRIP.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM DISPLAY_MSPECTRA;
 {$nolist}
 {$list}


  PROCEDURE DISPLAY_MSPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: ARRAY [low..high: integer] OF
                               spectral_data; starting_spectra_array_index, number_of_spectra: integer)) ;

{
 WHERE
     s = string to display
     spec = spectral data record
     starting_spectra_array_index = element of array to start displaying  (range is low..high)
     number_of_spectra = number of spectra to be displayed
 RESULTS
     the selected spectra are displayed
}

    VAR
      i, spec_start, spec_stop, display_start, display_stop, num, error_code: integer;
      error: boolean;
      ss: STRING [78];

    BEGIN
      prompt(2);
      strass(ss, s);
      writec(ss);
      jump(2);
      error_code := 0; { everything's ok to start with }

      IF NOT (starting_spectra_array_index IN [low..high]) THEN
        error_code := - 1; { invalid array index }

      IF error_code = 0 THEN
        IF number_of_spectra < 1 THEN
          error_code := - 2; { bad number parameter }

      IF error_code = 0 THEN
        IF NOT (starting_spectra_array_index + number_of_spectra - 1 IN [low..high]) THEN
          error_code := - 2; { invalid array index addressed }

      IF error_code <> 0 THEN
        BEGIN
        writec('DISPLAY_MSPECTRA - parameter error');
        return(2);
        END;

      IF error_code = 0 THEN
        BEGIN
        spec_start := starting_spectra_array_index;
        spec_stop := starting_spectra_array_index + number_of_spectra - 1;

        IF number_of_spectra > 1 THEN
          BEGIN
          error := false;
```

```
60              FOR i := spec_start + 1 TO spec_stop DO
61                BEGIN
62                error := error OR (spec[i].start <> spec[spec_start].start);
63                error := error OR (spec[i].stop <> spec[spec_start].stop);
64                error := error OR (spec[i].inc <> spec[spec_start].inc);
65                END;
66              IF error THEN
67                BEGIN
68                writec('MULTIPLE SPECTRA TO BE DISPLAYED HAVE MISMATCHED');
69                writec('WAVELENGTH PARAMETERS AND WILL BE DISPLAYED INDIVIDUALLY');
70                return(2);
71                error_code := - 100; { spectra have different wavelength parameters, treat them individually }
72                FOR i := spec_start TO spec_stop DO
73                  display_spectra('SPECTRA ARE BEING DISPLAYED INDIVIDUALLY', spec[i]);
74                END;
75              END;
76            END;
77
78          IF error_code = 0 THEN
79            BEGIN
80            IF spec[spec_start].start = 0 THEN
81              BEGIN
82              IF number_of_spectra = 1 THEN
83                writec('This array has not been loaded with the read option')
84              ELSE
85                writec('These arrays have not been loaded with the read option');
86              return(2);
87              END
88            ELSE IF spec[spec_start].start = 1 THEN
89              BEGIN
90              IF number_of_spectra = 1 THEN
91                writec('This array contains only color coordinates, not spectral data to display')
92              ELSE
93                writec('These arrays contain only color coordinates, not spectral data to display');
94              return(2);
95              END
96            ELSE
97              BEGIN
98              write(' ': 5, 'Spectra', spec_start: 3, ' File : ');
99              FOR num := 1 TO 55 DO
100               write(spec[spec_start].filename[num]);
101             writeln;
102             FOR i := spec_start + 1 TO spec_stop DO
103               BEGIN
104               write(' ': 5, 'Spectra', i: 3, ' File : ');
105               FOR num := 1 TO 55 DO
106                 write(spec[i].filename[num]);
107               writeln;
108               END;
109             jump(1);
110             writeln(' ': 5, 'starting wavelength :  ', spec[spec_start].start: 8, ' nm');
111             writeln(' ': 5, 'ending wavelength   :  ', spec[spec_start].stop: 8, ' nm');
112             writeln(' ': 5, 'wavelength increment:  ', spec[spec_start].inc: 8, ' nm');
113             jump(1);
```

```
114                 FOR i := spec_start TO spec_stop DO
115                   BEGIN
116                   write(' ': 5, 'Spectra', i: 3, ' Descrip : ');
117                   display_descrip(output, spec[i], 52, false, true, true);
118                   writeln;
119                   END;
120                 return(2);
121                 display_stop := spec_start - 1;
122                 REPEAT { set up to display up to three spectra at a time }
123                   prompt(2);
124                   display_start := display_stop + 1;
125                   IF spec_stop - display_start > 2 THEN
126                     display_stop := display_start + 2
127                   ELSE
128                     display_stop := spec_stop;
129
130                   write(' ': 13); { display some labels at the top of the first display screen }
131                   FOR num := display_start TO display_stop DO
132                     write(' ': 5 + 2, spec[num].name, ' ': 3);
133                   writeln;
134                   jump(1);
135                   FOR i := 1 TO (spec[spec_start].stop - spec[spec_start].start) DIV spec[spec_start].inc + 1 DO
136                     BEGIN
137                     IF i MOD 18 = 0 THEN
138                       BEGIN
139                       return(1);
140                       prompt(2);
141                       write(' ': 13); { display some labels at the top of the other display screens }
142                       FOR num := display_start TO display_stop DO
143                         write(' ': 5 + 2, spec[num].name, ' ': 3);
144                       writeln;
145                       jump(1);
146                       END;
147                     write((i - 1) * spec[spec_start].inc + spec[spec_start].start: 10, ' nm');
148                     FOR num := display_start TO display_stop DO
149                       write(' ': 5, spec[num].val[i]: 15);
150                     writeln;
151                     END;
152                   return(2);
153                 UNTIL display_stop = spec_stop;
154                 END;
155               END;
156           END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]DISPLAY_MSPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM DISPLAY_SPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   1D         PROCEDURE DISPLAY_SPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: spectral_data)) ;
   11       {
   12        WHERE
   13            s = string to display
   14            spec = spectral data record
   15        RESULTS
   16            the spectra is displayed
   17       }
   18
   19            TYPE
   2D              t_spectral_data = ARRAY [1..1] OF spectral_data;
   21
   22            VAR
   23              temp_spec: t_spectral_data;
   24              ss: STRING [78];
   25
   26            BEGIN
   27              temp_spec := loophole(t_spectral_data, spec);
   28              strass(ss, s);
   29              display_mspectra(ss, temp_spec, 1, 1);
   3D            END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]DISPLAY_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM ENTER_SAMPLE_COORDINATES;
 {$nolist}
 {$list}


  PROCEDURE ENTER_SAMPLE_COORDINATES ((s: PACKED ARRAY [low..high: integer] OF char; source: coordinate_data; VAR sample_
                                      coordinate_data; VAR typ: integer; default: default_type)} ;
{
 WHERE
     s = string to display
     source = coordinates of light source
     sample_coord = coordinates array for a sample
     typ = T when coordinates have already been loaded into the array ( <>1 when the array is empty or contains coordinat
                                                                        calculated from spectral data )
           (NOTE:  the variable START of record spectral_data can be passed as this parameter)
 RESULTS
     chromaticity coordinates and tristimulus values are returned in the array
}

    VAR
      sample: coordinate_data;
      ratio: real;
      q, q1: char;
      bad_data, bad_source: boolean;
      coord: coordinate_type;
      ss: STRING [78];

    BEGIN
      sample := sample_coord;
      bad_data := false;
      FOR coord := t_x TO c_z DO
        bad_data := bad_data OR calc_color_bad_value(sample[coord]);
      FOR coord := u TO v_prime DO
        bad_data := bad_data OR calc_color_bad_value(sample[coord]);
      IF (typ <> 1) OR bad_data THEN
        BEGIN
        FOR coord := succ(first) TO pred(last) DO
          sample[coord] := - 999;
        default := no_default;
        END;
      bad_source := false;
      FOR coord := t_x TO c_z DO
        bad_source := bad_source OR calc_color_bad_value(source[coord]);
      FOR coord := u_prime TO v_prime DO
        bad_source := bad_source OR calc_color_bad_value(source[coord]);
      strass(ss, s);
      q := 'T';
      REPEAT
        prompt(8);
        writec(ss);
        jump(1);
        writec('T :  enter tristimulus values       :  X, Y, & Z   ');
        writec('C :  enter chromaticity coordinates :  x, y, & Y   ');
        IF NOT bad_source THEN
          BEGIN
```

```
60              writec('A :  enter CIELAB coordinates       :  L*, a*, & b*');
61              writec('U :  enter CIELUV coordinates       :  L*, u*, & v*');
62              END;
63            jump(1);
64            IF NOT bad_source THEN
65              getchar('select type of coordinates to enter', q, use_default, ['T', 'C', 'A', 'U'])
66            ELSE
67              getchar('select type of coordinates to enter', q, use_default, ['T', 'C']);
68            prompt(10);
69            CASE q OF
70              'T':
71                BEGIN
72                bad_data := false;
73                FOR coord := t_x TO t_z DO
74                  bad_data := bad_data OR calc_color_bad_value(sample[coord]);
75                IF bad_data THEN
76                  default := no_default
77                ELSE
78                  default := use_default;
79                getre('enter tristimulus value X', sample[t_x], default, 0, 200);
80                getre('enter tristimulus value Y', sample[t_y], default, 0, 200);
81                getre('enter tristimulus value Z', sample[t_z], default, 0, 200);
82                calc_color_chromaticities(sample, bad_data);
83                END;
84              'C':
85                BEGIN
86                bad_data := calc_color_bad_value(sample[t_y]);
87                FOR coord := c_x TO c_y DO
88                  bad_data := bad_data OR calc_color_bad_value(sample[coord]);
89                IF bad_data THEN
90                  default := no_default
91                ELSE
92                  default := use_default;
93                getre('enter chromaticity coordinate x', sample[c_x], default, 1E-10, 1);
94                getre('enter chromaticity coordinate y', sample[c_y], default, 1E-10, 1);
95                getre('enter tristimulus value Y', sample[t_y], default, 0, 200);
96                ratio := sample[t_y] / sample[c_y];
97                sample[t_x] := sample[c_x] * ratio;
98                sample[t_z] := (1 - sample[c_x] - sample[c_y]) * ratio;
99                calc_color_chromaticities(sample, bad_data);
100               END;
101             'A':
102               BEGIN
103               bad_data := false;
104               FOR coord := l_star TO b_star DO
105                 bad_data := bad_data OR calc_color_bad_value(sample[coord]);
106               IF bad_data THEN
107                 default := no_default
108               ELSE
109                 default := use_default;
110               getre('enter CIELAB coordinate L*', sample[l_star], default, 0, 100);
111               getre('enter CIELAB coordinate a*', sample[a_star], default, -300, 300);
112               getre('enter CIELAB coordinate b*', sample[b_star], default, -300, 300);
113               sample[t_y] := sqr((sample[l_star] + 16) / 116) * ((sample[l_star] + 16) / 116) * source[t_y];
```

```
114                 sample[t_x] := sqr(sample[a_star] / 500 + (sample[l_star] + 16) / 116) * (sample[a_star] / 500 + (sample[l_st
115                                 116) * source[t_x];
116                 sample[t_z] := sqr((sample[l_star] + 16) / 116 - sample[b_star] / 200) * ((sample[l_star] + 16) / 116 - sampl
117                                 200) * source[t_z];
118                 calc_color_chromaticities(sample, bad_data);
119                 END;
120               'U':
121                 BEGIN
122                 bad_data := calc_color_bad_value(sample[l_star]);
123                 FOR coord := u_star TO v_star DO
124                   bad_data := bad_data OR calc_color_bad_value(sample[coord]);
125                 IF bad_data THEN
126                   default := no_default
127                 ELSE
128                   default := use_default;
129                 getre('enter CIELUV coordinate L*', sample[l_star], default, 1E-10, 100);
130                 getre('enter CIELUV coordinate u*', sample[u_star], default, - 300, 300);
131                 getre('enter CIELUV coordinate v*', sample[v_star], default, - 300, 300);
132                 sample[t_y] := sqr((sample[l_star] + 16) / 116) * ((sample[l_star] + 16) / 116) * source[t_y];
133                 sample[u_prime] := sample[u_star] / 13 / sample[l_star] + source[u_prime];
134                 sample[v_prime] := sample[v_star] / 13 / sample[l_star] + source[v_prime];
135                 IF (sample[u_prime] = 0) OR (sample[v_prime] = 0) THEN
136                   bad_data := true
137                 ELSE
138                   BEGIN
139                   sample[t_x] := 9 * sample[u_prime] * sample[t_y] / 4 / sample[v_prime];
140                   sample[t_z] := (4 * sample[t_x] / sample[u_prime] - sample[t_x] - 15 * sample[t_y]) / 3;
141                   calc_color_chromaticities(sample, bad_data);
142                   END;
143                 END;
144
145               END;
146             default := use_default;
147             jump(2);
148             IF bad_data THEN
149               BEGIN
150               writec('ERROR calculating color coordinates');
151               writec('please correct values entered');
152               return(2);
153               END
154             ELSE
155               BEGIN
156               q1 := 'N';
157               getchar('change the values entered above ? (Y,N)', q1, use_default, ['Y', 'N']);
158               END;
159           UNTIL NOT bad_data AND (q1 = 'N');
160           { signal that the data returned is coordinates }
161           typ := 1;
162           sample_coord := sample;
163         END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]ENTER_SAMPLE_COORDINATES.PASF;3/NOMAIN/LIST/DOUBLE

  *** No lines with errors detected ***
```

```
PROGRAM ENTER_SOURCE_COORDINATES;
 {$nolist}
 {$list}


  PROCEDURE ENTER_SOURCE_COORDINATES ((s: PACKED ARRAY [low..high: integer] OF char; VAR source_coord: coordinate_data; V
                                      integer; default: default_type)) ;
{
 WHERE
     s = string to display
     source = coordinates array for a source
     typ = 1 when coordinates have already been loaded into the array ( <>1 when the array is empty or contains coordinat
                                                                          calculated from spectral data )
           (NOTE:  the variable START of record spectral_data can be passed as this parameter)
 RESULTS
     chromaticity coordinates and tristimulus values are returned in the array
}

    VAR
      source: coordinate_data;
      ratio: real;
      q, q1: char;
      bad_data: boolean;
      coord: coordinate_type;
      ss: STRING [78];

    BEGIN
      source := source_coord;
      bad_data := false;
      FOR coord := t_x TO c_z DO
        bad_data := bad_data OR calc_color_bad_value(source[coord]);
      FOR coord := u TO v_prime DO
        bad_data := bad_data OR calc_color_bad_value(source[coord]);
      IF (typ <> 1) OR bad_data THEN
        BEGIN
        FOR coord := succ(first) TO pred(last) DO
          source[coord] := - 999;
        default := no_default;
        END;
      strass(ss, s);
      q := 'T';
      REPEAT
        prompt(8);
        writec(ss);
        jump(1);
        writec('T :  enter tristimulus values       :  X, Y, & Z');
        writec('C :  enter chromaticity coordinates :  x, y, & Y');
        jump(1);
        getchar('select type of coordinates to enter', q, use_default, ['T', 'C']);
        jump(2);
        CASE q OF
          'T':
            BEGIN
            getre('enter tristimulus value X', source[t_x], default, 0, 200);
            getre('enter tristimulus value Y', source[t_y], default, 0, 200);
```

```
60                  getre('enter tristimulus value Z', source[t_z], default, 0, 200);
61                  calc_color_chromaticities(source, bad_data);
62                  END;
63                'C':
64                  BEGIN
65                  getre('enter chromaticity coordinate x', source[c_x], default, 1E-10, 1);
66                  getre('enter chromaticity coordinate y', source[c_y], default, 1E-10, 1);
67                  getre('enter tristimulus value Y', source[t_y], default, 0, 200);
68                  ratio := source[t_y] / source[c_y];
69                  source[t_x] := source[c_x] * ratio;
70                  source[t_z] := (1 - source[c_x] - source[c_y]) * ratio;
71                  calc_color_chromaticities(source, bad_data);
72                  END;
73                END;
74              default := use_default;
75              jump(2);
76              IF bad_data THEN
77                BEGIN
78                writec('ERROR calculating color coordinates');
79                writec('please correct values entered');
80                return(2);
81                END
82              ELSE
83                BEGIN
84                q1 := 'N';
85                getchar('change the values entered above ? (Y,N)', q1, use_default, ['Y', 'N']);
86                END;
87            UNTIL NOT bad_data AND (q1 = 'N');
88            { signal that the data returned is coordinates }
89            typ := 1;
90            source_coord := source;
91          END;
```

```
PROGRAM ENTER_SPECTRA;
 {$nolist}
 {$list}


  PROCEDURE ENTER_SPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: spectral_data)) ;
{
 WHERE
     s = string to display
     spec = spectral data record
  RESULTS
     a spectral data record full of manually entered data
}

    CONST
      x_offset = 4; { these offsets are used by CURMAT to position the wavelength values on the screen }
      y_offset = 5;
      prompt_line = 21;

    VAR
      ss: STRING [255];
      temp: spectral_data;
      i, j, error_code: integer;
      q: char;


    BEGIN
      zero_spectra(spec);
      zero_spectra(temp);
      WITH temp DO
        BEGIN
        prompt(1);
        strass(ss, s);
        writec(ss);
        jump(1);
        writec('Spectral values can be entered over the wavelength range of');
        writeln(' ': 10, starting_wavelength: 0, ' nm  to  ', ending_wavelength: 0, ' nm  every 5, 10, or 20 nm');
        jump(1);
        inc := 10;
        REPEAT
          IF NOT (inc IN [5, 10, 20]) THEN
            writeln('INVALID INCREMENT... valid values are  5, 10 , & 20 nm');
          getin('Enter increment', inc, no_default, 5, 20);
        UNTIL (inc IN [5, 10, 20]);
        jump(1);
        writec('Starting and ending wavelengths must be a multiple of the increment');
        jump(1);
        start := starting_wavelength;
        getin('enter the starting wavelength', start, use_default, starting_wavelength, ending_wavelength - inc);
        stop := ending_wavelength;
        getin('enter the ending wavelength', stop, use_default, start + inc, ending_wavelength);
        jump(1);
        readst('Please enter a short description of these data (no intermediate spaces)', name, no_default, no_null);

        { trim the string after the first space }
```

```
60              i := 0;
61              REPEAT
62                i := i + 1;
63              UNTIL (i = 10) OR (name[i] = ' ');
64              IF i < 10 THEN
65                FOR i := i TO 10 DO
66                  name[i] := ' ';
67
68              jump(1);
69              readst('Please enter a long description of these data', descrip, no_default, no_null);
70
71              i := 0;
72              REPEAT
73                prompt(1);
74                writec(ss);
75                write('Description :  ');
76                display_descrip(output, temp, 63, false, true, true);
77                writeln;
78                writeln('Wavelength range (nm) :  ', start: 0, ' to ', stop: 0, ' every ', inc: 0);
79                j := i;
80                REPEAT
81                  i := i + 1;
82                  gotoxy(0, prompt_line);
83                  erase(eoscreen);
84                  writeln('Current Wavelength :  ', start + (i - 1) * inc: 0, ' nm');
85                  readre('enter value', val[i], no_default);
86                  curmat(i MOD 46 + i DIV 46, 3, 15, x_offset, y_offset, vertical);
87                  writeln(' ': 3, start + (i - 1) * inc: 0, ': ', val[i]: 15);
88                UNTIL (i - j = 45) OR (i = (stop - start) DIV inc + 1);
89              UNTIL (i = (stop - start) DIV inc + 1);
90
91              gotoxy(0, prompt_line);
92              erase(eoscreen);
93              q := 'N';
94              readch('Do you wish to make any corrections to these entries ?', q, use_default, null_ok);
95              IF q = 'Y' THEN
96                correct_spectra('CORRECT DATA JUST ENTERED', temp)
97              ELSE
98                BEGIN
99                q := 'N';
100               readch('Do you wish to save this spectra to a file ?', q, use_default, null_ok);
101               IF q = 'Y' THEN
102                 BEGIN
103                 error_code := 0;
104                 REPEAT
105                   gotoxy(0, prompt_line);
106                   erase(eoscreen);
107                   say_error_code('ERROR WRITING FILE, TRY AGAIN', error_code, false);
108                   IF error_code = 0 THEN
109                     readst('enter a filename for these data', filename, no_default, no_null)
110                   ELSE
111                     readst('enter a filename for these data', filename, use_default, no_null);
112                   save_spectra(temp, error_code);
113                 UNTIL error_code = 0;
```

```
  114                 END;
  115               END;
  116             END;
  117           spec := temp;
  118         END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]ENTER_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM FILE_IO;
 {$nolist}
 {$list}


  PROCEDURE FILE_IO ((s: PACKED ARRAY [lows..highs: integer] OF char; option: char; file_io_set: file_io_set_type; VAR sp
                      [low..high: integer] OF spectral_data)} ;
{
 WHERE
     s = string to display
     option = character defining the desired operations
     spec = an array of spectral data to operate on
     file_io_set_type = set defines permitted operations by this routine
  RESULTS
     spectra modified by various routines
}

    CONST
      skip_lines = 3;
      maximum_num = 10;

    VAR
      i, j, max_num, num, error_code, width, ext, line_length: integer;
      opt, q, q1, q2: char;
      ss: STRING [78];
      filename: STRING [70];

    BEGIN
      IF (low <> 1) OR (high < 1) OR (high > maximum_num) THEN
        BEGIN
        prompt(10);
        writec('FILE_IO cannot be used due to an array index mismatch');
        jump(1);
        writeln('  low = ', low: 0, '  high = ', high: 0, '  maximum = ', maximum_num: 0);
        return(2);
        END
      ELSE
        BEGIN
        max_num := high;
        strass(ss, s);
        IF NOT (option IN file_io_set) THEN
          BEGIN
          writec('ERROR in FILE_IO  -- invalid option');
          writeln(' ': 10, 'Option requested :  ', option);
          write(' ': 10, 'Valid Options    :  ');
          FOR opt := 'A' TO 'Z' DO
            IF (opt IN file_io_set) THEN
              write(' ', opt);
          writeln;
          return(2);
          END
        ELSE
          BEGIN
          gotoxy(0, max_num + skip_lines);
          erase(eoscreen);
```

```
60              writec('Select 0 (zero) to abort the selected option and return to the previous menu');
61              jump(1);
62              CASE option OF
63                'R':
64                  BEGIN
65                  writec('R E A O  S P E C T R A  ' + ss);
66                  getin('Select first spectra number to read', i, no_default, 0, max_num);
67                  IF i > 0 THEN
68                    BEGIN
69                    num := 0;
70                    get_mspectra('READ SPECTRA NUMBER ' + chr(i + 48) + '  ' + ss, spec, i, 0, num, use_default);
71                    END;
72                  END;
73                'O':
74                  BEGIN
75                  writec('D I S P L A Y  S P E C T R A  ' + ss);
76                  getin('Select first spectra number to display', i, no_default, 0, max_num);
77                  IF i > 0 THEN
78                    BEGIN
79                    num := 1;
80                    getin('enter number of spectra to display', num, use_default, 0, max_num - i + 1);
81                    IF num > 0 THEN
82                      display_mspectra('OISPLAY SPECTRA NUMBER ' + chr(i + 48) + '  ' + ss, spec, i, num);
83                    END;
84                  END;
85                'L':
86                  BEGIN
87                  writec('L I S T  S P E C T R A  ' + ss);
88                  getin('Select first spectra number to write to a listing file', i, no_default, 0, max_num);
89                  IF i > 0 THEN
90                    BEGIN
91                    num := 1;
92                    getin('enter number of spectra to list', num, use_default, 0, max_num - i + 1);
93                    IF num > 0 THEN
94                      BEGIN
95                      readst('enter filename for listing file', filename, no_default, no_null);
96                      q1 := 'Y';
97                      readch('report spectral values using scientific notation ? (Y,N)', q1, use_default, null_ok);
98                      width := 12;
99                      getin('enter field width for reporting real values', width, use_default, 2, 50);
100                     IF q1 = 'N' THEN
101                       BEGIN
102                       IF width < 5 THEN
103                         ext := width - 2
104                       ELSE
105                         ext := 3;
106                       getin('enter number of digits to report after a decimal point', ext, use_default, 0, width - 2);
107                       END
108                     ELSE
109                       ext := - 1;
110                     line_length := 80;
111                     getin('enter line length of report', line_length, use_default, 25, 250);
112                     report_mspectra(filename, 'LIST SPECTRA ' + ss, spec, i, num, width, ext, line_length, error_code);
113                     say_error_code('ERROR writing listing file', error_code, true);
```

```
                        END;
                      END;
                  END;
                'Z':
                  BEGIN
                  writec('Z E R O  S P E C T R A  ' + ss);
                  getin('Select first spectra number to zero', i, no_default, 0, max_num);
                  IF i > 0 THEN
                    BEGIN
                    num := 1;
                    getin('enter number of spectra to zero', num, use_default, 0, max_num - i + 1);
                    IF num > 0 THEN
                      BEGIN
                      getchar('Are you sure (Y,N) ?', q, no_default, ['Y', 'N']);
                      IF q = 'Y' THEN
                        FOR j := i TO i + num - 1 DO
                          zero_spectra(spec[j]);
                      END;
                    END;
                  END;
                'E':
                  BEGIN
                  writec('E N T E R  S P E C T R A  ' + ss);
                  getin('Select spectra number to enter by hand', i, no_default, 0, max_num);
                  IF i > 0 THEN
                    BEGIN
                    ss := 'E N T E R  S P E C T R A  ' + chr(i + 48) + '  ' + ss;
                    enter_spectra(ss, spec[i]);
                    END;
                  END;
                'C':
                  BEGIN
                  writec('C O R R E C T  S P E C T R A  ' + ss);
                  getin('Select spectra number to enter by hand', i, no_default, 0, max_num);
                  IF i > 0 THEN
                    correct_spectra('C O R R E C T  S P E C T R A  ' + chr(i + 48) + '  ' + ss, spec[i]);
                  END;
                'P':
                  BEGIN
                  writec('P L O T  S P E C T R A  ' + ss);
                  getin('Select first spectra number to plot', i, no_default, 0, max_num);
                  IF i > 0 THEN
                    BEGIN
                    num := 1;
                    getin('enter number of spectra to plot', num, use_default, 0, max_num - i + 1);
                    IF num > 0 THEN
                      plotrg_mspectra('P L O T  S P E C T R A  ' + chr(i + 48) + '  ' + ss, spec, i, num);
                    END;
                  END;
                'S':
                  BEGIN
                  writec('S A V E  S P E C T R A  ' + ss);
                  getin('Select the first spectra number to save', i, no_default, 0, max_num);
                  IF i > 0 THEN
```

```
168                     BEGIN
169                     num := 1;
170                     getin('enter number of spectra to save', num, use_default, 0, max_num - i + 1);
171                     IF num > 0 THEN
172                       BEGIN
173                       error_code := 0;
174                       REPEAT
175                         gotoxy(0, max_num + skip_lines + 6);
176                         erase(eoscreen);
177                         say_error_code('ERROR OPENING FILE, PLEASE TRY AGAIN', error_code, false);
178                         readst('enter filename for spectra ', spec[i].filename, use_default, no_null);
179                         save_mspectra(spec, i, num, error_code);
180                       UNTIL error_code = 0;
181                       END;
182                     END;
183                   END;
184                 END;
185               END;
186             END;
187         END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]FILE_IO.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM FILE_IO_OPTION_LINE;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE FILE_IO_OPTION_LINE ((file_io_set: file_io_set_type)) ;
   11       {
   12        WHERE
   13            file_io_set = set of characters that define the options available from the file_io procedure
   14         RESULTS
   15            an option line on the user's terminal that shows the procedure file_io options available
   16       }
   17
   18           VAR
   19             ss: STRING [78];
   20
   21           BEGIN
   22             ss[0] := chr(0);
   23             IF 'D' IN file_io_set THEN
   24               ss := ss + ' D(isplay';
   25             IF 'E' IN file_io_set THEN
   26               ss := ss + ' E(nter';
   27             IF 'C' IN file_io_set THEN
   28               ss := ss + ' C(orrect';
   29             IF 'R' IN file_io_set THEN
   30               ss := ss + ' R(ead';
   31             IF 'S' IN file_io_set THEN
   32               ss := ss + ' S(ave';
   33             IF 'P' IN file_io_set THEN
   34               ss := ss + ' P(lot';
   35             IF 'L' IN file_io_set THEN
   36               ss := ss + ' L(ist';
   37             IF 'Z' IN file_io_set THEN
   38               ss := ss + ' Z(ero';
   39
   40             IF ord(ss[0]) = 0 THEN
   41               writec('** NO FILE IO OPTIONS ARE AVAILABLE **')
   42             ELSE
   43               BEGIN
   44               ss := ss + '  spectra';
   45               writec(ss);
   46               END;
   47           END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]FILE_IO_OPTION_LINE.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM FILE_IO_SET_INIT;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE FILE_IO_SET_INIT ((VAR file_io_set: file_io_set_type)) ;
   11       {
   12        WHERE
   13            file_io_set = is a variable of type set of char
   14         RESULTS
   15            a set of acceptable characters for the file_io routine's operations
   16       }
   17
   18
   19           BEGIN
   20             { define a set of rountine character responses for the file_io procedure }
   21             file_io_set := ['R', 'D', 'E', 'C', 'P', 'Z', 'S', 'L'];
   22             IF NOT regis_terminal THEN
   23               file_io_set := file_io_set - ['P'];
   24           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]FILE_IO_SET_INIT.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM FILE_MATH;
 ($nolist)
 ($list)


  PROCEDURE FILE_MATH ((s: PACKED ARRAY [lows..highs: integer] OF char; VAR spec: ARRAY [low..high: integer] OF spectral_
                        file_io_set: file_io_set_type)) ;
{
 WHERE
     s = string to display
     spec = an array of spectra data to operate on
     file_io_set = character set of valid file_io options
  RESULTS
     spectra modified by various routines
}

    CONST
      null_type = no_null;
      maximum_num = 10; { must be less than or equal to 10}
      skip_lines = 3;

    VAR
      i, j, k, num, max_num, error_code: integer;
      tempb: big_spectral_data;
      temp: spectral_data;
      r: real;
      q, q1, q2, q3: char;
      need_display_refresh, error: boolean;
      ss: STRING [78];

    BEGIN
      IF (low <> 1) OR (high < 1) OR (high > maximum_num) THEN
        BEGIN
        prompt(10);
        writec('FILE_MATH cannot be used due to an array index mismatch');
        jump(1);
        writeln('  low = ', low: 0, '  high = ', high: 0, '  maximum = ', maximum_num: 0);
        return(2);
        END
      ELSE
        BEGIN
        max_num := high;
        strass(ss, s);
        REPEAT
          show_descrips(ss, spec);
          gotoxy(0, max_num + skip_lines);
          erase(eoscreen);
          writec('O P T I O N S');
          jump(1);
          file_io_option_line(file_io_set);
          writec('-----------------------------------------');
          writec('math functions that operate on O(ne or T(wo spectra');
          jump(1);
          getchar('select one of the above -or- Q(uit', q, no_default, file_io_set + ['O', 'T', 'Q']);
          gotoxy(0, max_num + skip_lines);
```

```
60              erase(eoscreen);
61              IF q IN file_io_set THEN
62                file_io(ss, q, file_io_set, spec)
63              ELSE
64                BEGIN
65                CASE q OF
66                  'T':
67                    BEGIN
68                    need_display_refresh := false;
69                    REPEAT
70                      IF need_display_refresh THEN
71                        show_descrips(ss, spec);
72                      need_display_refresh := true;
73                      gotoxy(0, max_num + skip_lines);
74                      erase(eoscreen);
75                      writec('----------------------');
76                      writec('MATH FUNCTIONS THAT OPERATE ON TWO SPECTRA');
77                      writec('----------------------------------------');
78                      writec('A(dd, S(ubtract, M(ultiply, and D(ivide Spectra');
79                      writec('----------------------------------------');
80                      writec('F(old or T(runcate one spectra to the limits of another spectra');
81                      jump(1);
82                      getchar('select one of the above or Q(uit', q1, no_default, ['A', 'S', 'M', 'D', 'F', 'T', 'Q']);
83
84                      gotoxy(0, max_num + skip_lines);
85                      erase(eoscreen);
86                      writec('Select 0 (zero) to abort the selected option and return to the previous menu');
87                      jump(1);
88                      CASE q1 OF
89                        'A':
90                          BEGIN
91                          writec('A D D   S P E C T R A');
92                          jump(1);
93                          writec('EQUATION FORM ==>  RESULT =  A + B');
94                          jump(1);
95                          getin('Select number of A spectra', i, no_default, 0, max_num);
96                          IF i <> 0 THEN
97                            BEGIN
98                            getin('Select number of B spectra', j, no_default, 0, max_num);
99                            IF j <> 0 THEN
100                             BEGIN
101                             getin('Select number of RESULT spectra', k, no_default, 0, max_num);
102                             IF k <> 0 THEN
103                               BEGIN
104                               spec[k] := math(spec[i], spec[j], math_add, error);
105                               IF error THEN
106                                 writeln('ERROR IN 1+2');
107                               readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
108                               readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
109                               END;
110                             END;
111                           END;
112                         END;
113                       'S':
```

```
                        BEGIN
                        writec('S U B T R A C T   S P E C T R A');
                        jump(1);
                        writec('EQUATION FORM ==>  RESULT =  A - B');
                        jump(1);
                        getin('Select number of A spectra', i, no_default, 0, max_num);
                        IF i <> 0 THEN
                          BEGIN
                          getin('Select number of B spectra', j, no_default, 0, max_num);
                          IF j <> 0 THEN
                            BEGIN
                            getin('Select number of RESULT spectra', k, no_default, 0, max_num);
                            IF k <> 0 THEN
                              BEGIN
                              spec[k] := math(spec[i], spec[j], math_subtract_2_from_1, error);
                              IF error THEN
                                writeln('ERROR IN 2-1');
                              readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                              readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                              END;
                            END;
                          END;
                        END;
                      'D':
                        BEGIN
                        writec('D I V I D E   S P E C T R A');
                        jump(1);
                        writec('EQUATION FORM ==>  RESULT =  A / B');
                        jump(1);
                        getin('Select number of A spectra', i, no_default, 0, max_num);
                        IF i <> 0 THEN
                          BEGIN
                          getin('Select number of B spectra', j, no_default, 0, max_num);
                          IF j <> 0 THEN
                            BEGIN
                            getin('Select number of RESULT spectra', k, no_default, 0, max_num);
                            IF k <> 0 THEN
                              BEGIN
                              spec[k] := math(spec[i], spec[j], math_divide_1_by_2, error);
                              IF error THEN
                                writeln('ERROR IN 1/2');
                              readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                              readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                              END;
                            END;
                          END;
                        END;
                      'M':
                        BEGIN
                        writec('M U L T I P L Y   S P E C T R A');
                        jump(1);
                        writec('EQUATION FORM ==>  RESULT =  A * B');
                        jump(1);
                        getin('Select number of A spectra', i, no_default, 0, max_num);
```

```
168                             IF i <> 0 THEN
169                               BEGIN
170                               getin('Select number of B spectra', j, no_default, 0, max_num);
171                               IF j <> 0 THEN
172                                 BEGIN
173                                 getin('Select number of RESULT spectra', k, no_default, 0, max_num);
174                                 IF k <> 0 THEN
175                                   BEGIN
176                                   spec[k] := math(spec[i], spec[j], math_multiply, error);
177                                   IF error THEN
178                                     writeln('ERROR IN 1*2');
179                                   readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
180                                   readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
181                                   END;
182                                 END;
183                               END;
184                             END;
185                           'F':
186                             BEGIN
187                             writec('F O L D   S P E C T R A');
188                             jump(1);
189                             writec('EQUATION FORM ==>  RESULT =  FOLD A to the wavelength limits of B');
190                             jump(1);
191                             getin('Select number of A spectra', i, no_default, 0, max_num);
192                             IF i <> 0 THEN
193                               BEGIN
194                               getin('Select number of B spectra', j, no_default, 0, max_num);
195                               IF j <> 0 THEN
196                                 BEGIN
197                                 getin('Select number of RESULT spectra', k, no_default, 0, max_num);
198                                 IF k <> 0 THEN
199                                   BEGIN
200                                   spec[k] := fold_1_like_2(spec[i], spec[j], error);
201                                   IF error THEN
202                                     writeln('ERROR FOLDING');
203                                   readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
204                                   readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
205                                   END;
206                                 END;
207                               END;
208                             END;
209                           'T':
210                             BEGIN
211                             writec('T R U N C A T E   S P E C T R A');
212                             jump(1);
213                             writec('EQUATION FORM ==>  RESULT =  TRUNCATE A to the wavelength limits of B');
214                             jump(1);
215                             getin('Select number of A spectra', i, no_default, 0, max_num);
216                             IF i <> 0 THEN
217                               BEGIN
218                               getin('Select number of B spectra', j, no_default, 0, max_num);
219                               IF j <> 0 THEN
220                                 BEGIN
221                                 getin('Select number of RESULT spectra', k, no_default, 0, max_num);
```

```
                                IF k <> 0 THEN
                                  BEGIN
                                  spec[k] := truncate_1_like_2(spec[i], spec[j], error);
                                  IF error THEN
                                    writeln('ERROR TRUNCATING');
                                  readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                                  readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                                  END;
                                END;
                              END;
                            END;
                          'Q': ;
                          END;
                      UNTIL q1 = 'Q';
                      END;

                    '0':
                      BEGIN
                      need_display_refresh := false;
                      REPEAT
                        IF need_display_refresh THEN
                          show_descrips(ss, spec);
                        need_display_refresh := true;
                        gotoxy(0, max_num + skip_lines);
                        erase(eoscreen);
                        writec('----------------------');
                        writec('MATH FUNCTIONS THAT OPERATE ON ONE SPECTRA');
                        writec('-----------------------------------------');
                        writec('A(dd a constant to a spectra, M(ultiply a spectra by a constant');
                        writec('-----------------------------------------');
                        writec('S(um, L(og, U(nlog, K(/S, I(nterpolate or R(aise a spectra to a power');
                        jump(1);
                        getchar('select one of the above or Q(uit', q1, no_default, ['A', 'M', 'S', 'L', 'U', 'K', 'I', 'R', 'Q

                        gotoxy(0, max_num + skip_lines);
                        erase(eoscreen);
                        writec('Select 0 (zero) to abort the selected option and return to the previous menu');
                        jump(1);
                        CASE q1 OF
                          'A':
                            BEGIN
                            writec('ADD A CONSTANT TO A SPECTRA');
                            jump(1);
                            getin('Select number of spectra', i, no_default, 0, max_num);
                            IF i <> 0 THEN
                              BEGIN
                              readre('enter constant value to add', r, no_default);
                              getin('Select number of RESULT spectra', k, no_default, 0, max_num);
                              IF k <> 0 THEN
                                BEGIN
                                spec[k] := math_constant(spec[i], r, math_constant_add, error);
                                IF error THEN
                                  writeln('ERROR TRUNCATING');
                                readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
```

```
276                             readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
277                             END;
278                           END;
279                         END;
280                       'I':
281                         BEGIN
282                         writec('INTERPOLATE A SPECTRA');
283                         jump(1);
284                         getin('Select number of spectra', i, no_default, 0, max_num);
285                         IF i <> 0 THEN
286                           BEGIN
287                           getin('Select number of RESULT spectra', k, no_default, 0, max_num);
288                           IF k <> 0 THEN
289                             BEGIN
290                             calc_lagrange(tempb, spec[i], error);
291                             IF error THEN
292                               writeln('ERROR INTERPOLATION');
293                             temp := spec[i];
294                             writeln('wavelength increment of original data :  ', spec[i].inc: 0, ' nm');
295                             temp.inc := 5;
296                             REPEAT
297                               IF NOT (temp.inc IN [5, 10, 20]) THEN
298                                 writeln('TRY AGAIN.... select 5, 10, or 20 nanometers');
299                               readin('enter wavelength increment for the interpolated data (5,10,20)', temp.inc, use_defaul
300                             UNTIL temp.inc IN [5, 10, 20];
301                             IF (starting_wavelength - temp.start) MOD temp.inc <> 0 THEN
302                               temp.start := starting_wavelength;
303                             REPEAT
304                               IF NOT (starting_wavelength - temp.start) MOD temp.inc = 0 THEN
305                                 writeln('TRY AGAIN.... starting wavelength is not valid');
306                               getin('enter low wavelength (data is folded)', temp.start, use_default, starting_wavelength,
307                                     ending_wavelength - temp.inc);
308                             UNTIL (starting_wavelength - temp.start) MOD temp.inc = 0;
309                             IF (starting_wavelength - temp.stop) MOD temp.inc <> 0 THEN
310                               temp.stop := ending_wavelength;
311                             REPEAT
312                               IF NOT (starting_wavelength - temp.stop) MOD temp.inc = 0 THEN
313                                 writeln('TRY AGAIN.... ending wavelength is not valid');
314                               getin('enter high wavelength (data is folded)', temp.stop, use_default, temp.start + temp.inc
315                                     ending_wavelength);
316                             UNTIL (starting_wavelength - temp.stop) MOD temp.inc = 0;
317                             make_spectra_from_big_spectra(tempb, spec[k], temp.start, temp.stop, temp.inc, error);
318                             IF error THEN
319                               writeln('ERROR INTERPOLATION');
320                             readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
321                             readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
322                             END;
323                           END;
324                         END;
325                       'M':
326                         BEGIN
327                         writec('MULTIPLY A SPECTRA BY A CONSTANT');
328                         jump(1);
329                         getin('Select number of spectra', i, no_default, 0, max_num);
```

```
330                         IF i <> 0 THEN
331                           BEGIN
332                           readre('enter constant value to multiply by', r, no_default);
333                           getin('Select number of RESULT spectra', k, no_default, 0, max_num);
334                           IF k <> 0 THEN
335                             BEGIN
336                             spec[k] := math_constant(spec[i], r, math_constant_multiply, error);
337                             IF error THEN
338                               writeln('ERROR TRUNCATING');
339                             readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
340                             readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
341                             END;
342                           END;
343                         END;
344                       'S':
345                         BEGIN
346                         writec('SUM A SPECTRA');
347                         jump(1);
348                         getin('Select number of spectra', i, no_default, 0, max_num);
349                         IF i <> 0 THEN
350                           BEGIN
351                           IF spec[i].start = 0 THEN
352                             writec('NO SPECTRAL DATA LOADED -- LOAD DATA WITH THE READ OPTION')
353                           ELSE IF spec[i].start = 1 THEN
354                             writec('ONLY COORDINATES LOADED, NO SPECTRAL DATA')
355                           ELSE
356                             BEGIN
357                             j := spec[i].start;
358                             getin('enter low wavelength limit', j, use_default, spec[i].start, spec[i].stop - spec[i].inc);
359                             k := spec[i].stop;
360                             getin('enter high wavelength limit', k, use_default, spec[i].start + spec[i].inc, spec[i].stop)
361                             r := spectral_summation(spec[i], j, k, error);
362                             IF error THEN
363                               writec('ERROR CALCULATING SUMMATION')
364                             ELSE
365                               writeln(' ': 5, 'Summation = ', r: 15, '   (NOT multiplied by delta-lambda)');
366                             return(2);
367                             END;
368                           END;
369                         END;
370                       'L':
371                         BEGIN
372                         writec('TAKE THE LOG (base 10) OF A SPECTRA');
373                         jump(1);
374                         writec('EQUATION FORM ==>  RESULT =  LOG10 ( SPECTRA A VALUES * CONSTANT');
375                         jump(1);
376                         getin('Select number of spectra A', i, no_default, 0, max_num);
377                         IF i <> 0 THEN
378                           BEGIN
379                           r := 1;
380                           readre('enter CONSTANT value', r, use_default);
381                           getin('Select number of RESULT spectra', k, no_default, 0, max_num);
382                           IF k <> 0 THEN
383                             BEGIN
```

```
                                spec[k] := math_constant(spec[i], r, math_constant_log10, error);
                                IF error THEN
                                  writeln('ERROR IN LOG10');
                                readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                                readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                                END;
                            END;
                          END;
                        'R':
                          BEGIN
                          writec('RAISE A SPECTRA TO A POWER');
                          jump(1);
                          getin('Select number of spectra', i, no_default, 0, max_num);
                          IF i <> 0 THEN
                            BEGIN
                            readre('enter power to raise spectral values to', r, no_default);
                            getin('Select number of RESULT spectra', k, no_default, 0, max_num);
                            IF k <> 0 THEN
                              BEGIN
                              spec[k] := math_constant(spec[i], r, math_constant_raise, error);
                              IF error THEN
                                writeln('ERROR RAISEING');
                              readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                              readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                              END;
                            END;
                          END;
                        'U':
                          BEGIN
                          writec('UNLOG (base 10) OF A SPECTRA');
                          jump(1);
                          writec('EQUATION FORM ==>  RESULT VALUES =  10 ^ ( A VALUES * CONSTANT )');
                          jump(1);
                          getin('Select number of spectra A', i, no_default, 0, max_num);
                          IF i <> 0 THEN
                            BEGIN
                            r := - 1;
                            readre('enter CONSTANT value', r, use_default);
                            getin('Select number of RESULT spectra', k, no_default, 0, max_num);
                            IF k <> 0 THEN
                              BEGIN
                              spec[k] := math_constant(spec[i], r, math_constant_unlog10, error);
                              IF error THEN
                                writeln('ERROR IN UNLOG10');
                              readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
                              readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
                              END;
                            END;
                          END;
                        'K':
                          BEGIN
                          writec('K/S OF A SPECTRA');
                          jump(1);
                          writec('EQUATION FORM ==>  RESULT VALUES = SQUARE(1-A VALUES)/2/(A VALUES)');
```

```
438                             jump(1);
439                             getin('Select number of spectra A', i, no_default, 0, max_num);
440                             IF i <> 0 THEN
441                               BEGIN
442                               r := - 1;
443                               q3 := 'N';
444                               getchar('Log (base 10) the result ? (Y,N)', q3, use_default, ['Y', 'N']);
445                               getin('Select number of RESULT spectra', k, no_default, 0, max_num);
446                               IF k <> 0 THEN
447                                 BEGIN
448                                 r := 0;
449                                 spec[k] := math_constant(spec[i], r, math_constant_k_over_s, error);
450                                 IF error THEN
451                                   BEGIN
452                                   writeln('ERROR IN K/S CALCULATION');
453                                   writeln('RESULTS MAY BE QUESTIONABLE');
454                                   return(1);
455                                   END
456                                 ELSE
457                                   BEGIN
458                                   r := 1;
459                                   spec[k] := math_constant(spec[i], r, math_constant_log10, error);
460                                   IF error THEN
461                                     BEGIN
462                                     writeln('ERROR IN LOGGING K/S RESULT');
463                                     writeln('RESULTS MAY BE QUESTIONABLE');
464                                     return(1);
465                                     END
466                                   END;
467                                 readst('enter long description for RESULT spectra', spec[k].descrip, use_default, null_type);
468                                 readst('enter short description for RESULT spectra', spec[k].name, use_default, null_type);
469                                 END;
470                               END;
471                             END;
472                           'Q': ;
473                           END;
474                       UNTIL q1 = 'Q';
475                       END;
476                     'Q':
477                       prompt(10);
478                     END;
479                   END;
480               UNTIL q = 'Q';
481               prompt(10);
482               END;
483           END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]FILE_MATH.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM FOLD_1_LIKE_2;
 {$nolist}
 {$list}


  FUNCTION FOLO_1_LIKE_2 ((spec1, spec2: spectral_data; VAR error: boolean): spectral_data) ;
{
 WHERE
       spec1 = spectral data record number 1
       spec2 = spectral data record number 2
       error = signals error in parameters
  RESULTS
       fold_1_like_2  = spectra 1 folded to the limits of spectra 2
}

    VAR
      i, j: integer;
      temp: spectral_data;

    BEGIN
      error := false;
      IF spec1.start = 0 THEN
        BEGIN
        error := true;
        writec('%FOLD-WARNING !!! data not in memory for sample 1');
        END;
      IF spec2.start = 0 THEN
        BEGIN
        error := true;
        writec('%FOLD-WARNING !!! data not in memory for sample 2');
        END;

      IF NOT error THEN
        BEGIN

        temp := spec1;
        FOR i := 1 TO number_of_wavelengths DO
          temp.val[i] := 0;

        { find start, stop to fold }
        IF spec1.start > spec2.start THEN
          temp.start := spec2.start;
        IF spec1.stop < spec2.stop THEN
          temp.stop := spec2.stop;
        { *** needs more error checking above }

        { fold the lower wavelengths if necessary }
        IF spec1.start <> temp.start THEN
          BEGIN
          i := temp.start;
          REPEAT
            temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[1];
            i := i + temp.inc;
          UNTIL i >= spec1.start;
          END;
```

```
60
61              { leave unfolded  wavelengths alone }
62              i := spec1.start;
63              REPEAT
64                temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1];
65                i := i + temp.inc;
66              UNTIL i > spec1.stop;
67
68              { fold the higher wavelengths if necessary }
69              IF spec1.stop <> temp.stop THEN
70                BEGIN
71                j := i - temp.inc;
72                REPEAT
73                  temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(j - spec1.start) DIV spec1.inc + 1];
74                  i := i + temp.inc;
75                UNTIL i > temp.stop;
76                END;
77              END
78            ELSE
79              zero_spectra(temp);
80
81            FOLD_1_LIKE_2 := temp;
82          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]FOLD_1_LIKE_2.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM GETCHAR;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE GETCHAR ((s: PACKED ARRAY [low..high: integer] OF char; VAR value: char; default: default_type; valid_respons
11                            getchar_charset)) ;
12      {
13       WHERE
14           s = string to display in prompt
15           value = character variable to contain response
16           default = permits use of default value of variable value
17           valid_response = set of valid characters
18        RESULTS
19           a character from the valid_response set
20      }
21
22          VAR
23            message_line, len, i: integer;
24            value_temp: char;
25            first_time, done: boolean;
26            ss: PACKED ARRAY [1..80] OF char;
27
28
29          BEGIN
30            value_temp := value;
31            FOR len := low TO high DO
32              ss[len - low + 1] := s[len];
33            FOR len := high - low + 2 TO 80 DO
34              ss[len] := chr(0);
35            first_time := true;
36            IF default = use_default THEN
37              len := 4
38            ELSE
39              len := 0;
40            IF high - low + 1 + len + 4 + 12 <= 78 THEN
41              message_line := 2
42            ELSE
43              message_line := 3;
44            REPEAT
45              IF NOT first_time THEN
46                BEGIN
47                setscn([bold, blinking]);
48                write(chr(7));
49                END;
50              first_time := false;
51              FOR i := 1 TO message_line DO
52                writeln; { new line 10/20/87 EGT }
53
54              write('                          ENTER AN ACCEPTABLE CHARACTER                          ', chr(13));
55
56              setscn([not_bold, not_blinking]);
57              cursor(up, message_line);
58              value := value_temp;
59
```

```
60              readch(ss, value, default, null_ok);
61
62              done := (value IN valid_response);
63
64              IF NOT done THEN
65                FOR i := 1 TO message_line - 1 DO
66                  BEGIN
67                  cursor(up, 1);
68                  erase(line);
69                  END;
70            UNTIL done;
71            cursor(down, 1);
72            erase(line);
73            cursor(up, 1);
74          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]GETCHAR.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM GETINF;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE GETINF ((fn: PACKED ARRAY [low..high: integer] OF char; VAR info: PACKED ARRAY [lowd..highd: integer] OF char
11                           extra_info: PACKED ARRAY [lowx..highx: integer] OF char; default: default_type)) ;
12      {
13       WHERE
14           fn  = filename of getinf data file
15           info  = line of information returned from the file
16           extra_info  = extra information on the line after info
17           default     = permits use of a default value
18        RESULTS
19           info and extra_info selected from a data file or entered by hand
20      }
21
22      {  Program Written/ 10-31-87/ Mitch Miller
23         Misc cleanup   / 12-15-87/ Eric Townsend
24         end of informational heading }
25
26          CONST
27            max_getinf_defs = 15;
28            max_getinf_len = 78;
29            max_getinf_fn = 40;
30
31          TYPE
32            getinf_lines = PACKED ARRAY [1..max_getinf_len] OF char;
33
34          VAR
35            inp: text;
36            lines: ARRAY [0..max_getinf_defs] OF getinf_lines;
37            filename: PACKED ARRAY [1..max_getinf_fn] OF char;
38            i, j, k, x, default_k: integer;
39            str_equal: boolean;
40            max_k: integer;
41            allow_other_code: integer;
42            allow_other: boolean;
43
44
45          BEGIN
46
47            { === put fn into filename for use in a reset === }
48
49            IF low = 0 THEN
50              j := ord(fn[0])
51            ELSE
52              j := high;
53            IF j > max_getinf_fn THEN
54              j := max_getinf_fn;
55            FOR i := 1 TO j DO
56              filename[i] := fn[i];
57            FOR i := j + 1 TO max_getinf_fn DO
58              filename[i] := ' ';
59
```

```
60            { === reset the file === }
61
62            reset(inp, filename, '.INF', i);
63            IF i = - 1 THEN
64              BEGIN
65              writeln('WARNING...  File: ', filename);
66              writeln('            NOT AVAILABLE');
67              return(2);
68              exitst(4);
69              END;
70
71            { === Read file into array === }
72
73            readln(inp, lines[0]); { 1st line contains descriptive info }
74            readln(inp, default_k, allow_other_code); { 2nd line contains default default and allow other values flag }
75            readln(inp); { Skip past 3rd record }
76            IF allow_other_code <> 0 THEN
77              allow_other := true
78            ELSE
79              allow_other := false;
80
81            i := 0;
82            WHILE NOT eof(inp) AND (i <= max_getinf_defs) DO
83              BEGIN
84              i := i + 1;
85              readln(inp, lines[i]);
86              END;
87            close(inp);
88
89            { search array for entry containing default value }
90            { NOTE: default from parameter overides default from file }
91
92            { pad any passed default with blanks }
93            IF lowd = 0 THEN
94              FOR j := ord(info[0]) + 1 TO highd DO
95                info[j] := ' '
96            ELSE
97              FOR j := 1 TO highd DO
98                IF info[j] = chr(0) THEN
99                  info[j] := ' ';
100
101           IF default = use_default THEN
102             BEGIN
103             default_k := 0;
104             FOR j := 1 TO i DO
105               BEGIN
106               str_equal := true;
107               FOR k := 1 TO highd DO
108                 IF lines[j][k] <> info[k] THEN
109                   str_equal := false;
110               IF str_equal THEN
111                 default_k := j;
112               END;
113             IF default_k = 0 THEN
```

```
114                 BEGIN
115                 i := i + 1;
116                 default_k := i;
117                 FOR k := 1 TO highd DO
118                   lines[i][k] := info[k];
119                 FOR k := highd + 1 TO max_getinf_len DO
120                   lines[i][k] := ' ';
121                 END;
122               END;
123
124             { output list of available selections }
125
126             x := 78 DIV (highd + 6);
127             writeln(lines[0]);
128             writeln;
129             FOR j := 1 TO i DO
130               BEGIN
131               write(j: 2, ': ');
132               FOR k := 1 TO highd DO
133                 write(lines[j][k]); { output character to screen }
134               write('  ');
135               IF (j MOD x) = 0 THEN
136                 writeln;
137               END;
138             IF allow_other THEN
139               write((i + 1): 2, ': Other');
140             IF ((i + 1) MOD x) <> 0 THEN
141               writeln;
142             writeln;
143
144             { get selection number from screen }
145
146             k := default_k;
147             IF allow_other THEN
148               max_k := i + 1
149             ELSE
150               max_k := i;
151             IF k IN [1..max_k] THEN
152               default := use_default
153             ELSE
154               default := no_default;
155             getin('Select one of the above ', k, default, 1, max_k);
156
157             { assign appropriate values to parameters }
158
159             IF k > i THEN
160
161               BEGIN
162
163               k := 0;
164
165               { === get user entered value for info === }
166
167               write('Enter value :  ');
```

```
  168             readln(lines[0]);
  169
  170             { === Set extra info to blanks === }
  171
  172             FOR i := highd + 1 TO max_getinf_len DO
  173               lines[0][i] := ' ';
  174
  175             END;
  176
  177           { === Copy info from array to parameter === }
  178           IF lowd = 0 THEN
  179             info[0] := chr(highd);
  180           IF highd < max_getinf_len THEN
  181             j := highd
  182           ELSE
  183             j := max_getinf_len;
  184           FOR i := 1 TO j DO
  185             info[i] := lines[k][i];
  186           FOR i := j + 1 TO highd DO
  187             info[i] := ' ';
  188
  189           { === Copy extra info from array to parameter === }
  190           IF lowx = 0 THEN
  191             extra_info[0] := chr(highx);
  192           IF highd + highx < max_getinf_len THEN
  193             j := highx
  194           ELSE
  195             j := max_getinf_len - highd;
  196           FOR i := 1 TO j DO
  197             extra_info[i] := lines[k][i + highd];
  198           FOR i := j + 1 TO highx DO
  199             extra_info[i] := ' ';
  200
  201         END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]GETINF.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM GET_MSPECTRA;
 {$nolist}
 {$list}


  PROCEDURE GET_MSPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: ARRAY [low..high: integer] OF spectr
                           starting_spectra_array_index, starting_spectra_in_file_to_read: integer; VAR number_of_spectra:
                           default: default_type)) ;
{
 WHERE
     s = string to display
     spec = spectral data record
     starting_spectra_array_index = element of array to start reading into  (range is low..high)
     starting_spectra_in_file_to_read = starting spectra number
     number_of_spectra = number of spectra to be loaded into the array
     default = (use_default,no_default for filenames)
 RESULTS
     one or more spectra are selected by filename and read from disk and returned
}

    VAR
      ss: STRING [255];
      temp: spectral_data;
      temp_starting_spectra_in_file_to_read, temp_num, temp_number_of_spectra, num, error_code: integer;

    BEGIN
      temp_starting_spectra_in_file_to_read := starting_spectra_in_file_to_read;
      temp_number_of_spectra := number_of_spectra;
      error_code := 0;
      REPEAT
        starting_spectra_in_file_to_read := temp_starting_spectra_in_file_to_read;
        number_of_spectra := temp_number_of_spectra;
        prompt(6);
        strass(ss, s);
        writec(ss);
        jump(1);
        IF error_code < 0 THEN
          BEGIN
          setscn([bold, blinking]);
          IF error_code = - 4 THEN
            writeln('%GET_MSPECTRA-ERROR Error Opening File :  ', spec[starting_spectra_array_index].filename)
          ELSE
            writeln('%GET_MSPECTRA-ERROR Error in File :  ', spec[starting_spectra_array_index].filename);
          writec('Please Try Again');
          setscn([not_bold, not_blinking]);
          jump(1);
          END;
        writec('enter "?" to return to main menu -or-');
        jump(1);
        readst('enter filename of spectral data file to retrieve', spec[starting_spectra_array_index].filename, default,
        IF (starting_spectra_in_file_to_read = 0) OR (number_of_spectra = 0) OR (number_of_spectra = - 1) THEN
          BEGIN
          check_spectral_file(spec[starting_spectra_array_index].filename, temp.descrip, temp.start, temp.stop, temp.inc,
                              error_code);
          IF error_code = 0 THEN
```

```
                BEGIN
                jump(1);
                IF starting_spectra_in_file_to_read = 0 THEN
                  BEGIN
                  starting_spectra_in_file_to_read := 1;
                  IF num = 1 THEN
                    writeln('This file contains one spectra which has been read into memory')
                  ELSE
                    getin('enter the first column number to read from spectral data file', starting_spectra_in_file_to_read,
                          use_default, 1, num);
                  writeln;
                  END;
                IF number_of_spectra = 0 THEN
                  BEGIN
                  number_of_spectra := 1;
                  IF num = starting_spectra_in_file_to_read THEN
                    BEGIN
                    writeln('This file contains only one valid spectra which has been read into memory')
                    END
                  ELSE
                    BEGIN
                    temp_num := num - starting_spectra_in_file_to_read + 1;
                    IF temp_num > high - low + 1 THEN
                      temp_num := high - low + 1;

                    getin('enter the number of spectra to read from spectral data file', number_of_spectra, use_default, 1, t
                    END;
                  writeln;
                  END
                ELSE
                  BEGIN
                  number_of_spectra := num - starting_spectra_in_file_to_read + 1;
                  IF number_of_spectra > high - low + 1 THEN
                    number_of_spectra := high - low + 1;
                  END;
                END;

              END;

            IF error_code = 0 THEN
              retrieve_mspectra(spec, starting_spectra_array_index, starting_spectra_in_file_to_read, number_of_spectra, erro

            IF error_code <> 0 THEN
              BEGIN
              writeln('%GET_MSPECTRA-ERROR Error ', error_code: 0, ' from CHECK_SPECTRAL_FILE');
              return(2);
              END;

          UNTIL error_code = 0;
        END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]GET_MSPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM GET_SPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE GET_SPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: spectral_data; starting_spectra_in_fi
   11                                 integer; default: default_type)) ;
   12       {
   13        WHERE
   14            s = string to display
   15            spec = spectral data record
   16            starting_spectra_in_file_to_read = starting spectra number
   17            default = (use_default,no_default for filenames)
   18        RESULTS
   19            a spectra selected by filename is read from disk and returned
   20       }
   21
   22           TYPE
   23             t_spectral_data = ARRAY [1..1] OF spectral_data;
   24
   25           VAR
   26             temp_spec: t_spectral_data;
   27             ss: STRING [255];
   28             num: integer;
   29
   30           BEGIN
   31             strass(ss, s);
   32             temp_spec := loophole(t_spectral_data, spec);
   33             num := 1;
   34             get_mspectra(ss, temp_spec, 1, starting_spectra_in_file_to_read, num, default);
   35             spec := loophole(spectral_data, temp_spec);
   36           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]GET_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM MAKE_BLACKBODY_SPECTRA;
 {$nolist}
 {$list}


  FUNCTION MAKE_BLACKBODY_SPECTRA ((t: real; VAR error: boolean): spectral_data) ;
{
 WHERE
     t = color temperature of a blackbody to create (in automatic mode)
                                                    or a value of <= 0 causes the user to be prompted for a value of tempera
     spec = spectral data record
  RESULTS
     make_blackbody_spectra = spectra of blackbody radiator having the color temperature specified
}

{ make_blackbody_spectra calculates the spectral power distribution of a blackbody illuminant
at a desired color temperature using Planck's radiant excitance equation from Grum's Optical Radiation
Measurements Vol. 2 on Colorimetry (p 72) and returns the calculated values }

    VAR
      i, error_code: integer;
      temp: spectral_data;
      query_user: boolean;
      planck_560: real;
      q: char;


    FUNCTION raise(y, x: real): real;

      BEGIN
        IF y > 0 THEN
          raise := exp(x * ln(y))
        ELSE IF y < 0 THEN
          raise := - 999 { signal error on negative y values }
        ELSE
          y := 0; { assume 0^x = 0 }
      END;


    FUNCTION Planck(wave, t: real): real;

      CONST
        {    C1 = 3.74150E-16;} { from GRUMS Vol 2 }
        {     C2 = 1.4388E-2; }
        C1 = 3.741832E-16; { from COLOR SCIENCE }
        C2 = 1.438786E-2;

      BEGIN
        wave := wave * 1E-9;
        Planck := C1 / raise(wave, 5) / (exp(C2 / wave / t) - 1);
      END;


    BEGIN
      error := false;
```

```
60            zero_spectra(temp);
61
62            { select between user query mode and automatic mode }
63            query_user := t <= 0;
64
65            IF query_user THEN
66              BEGIN
67              prompt(1);
68              writec('MAKE BLACKBODY ILLUMINANT');
69              jump(3);
70              writec('The color temperature desired must be entered');
71              jump(2);
72              getre('enter temperature (K)', t, no_default, 500, 25000);
73              END;
74
75            { range check for the automatic mode }
76            error := (t < 500) OR (t > 25000);
77
78            IF NOT error THEN
79              BEGIN
80              WITH temp DO
81                BEGIN
82                stass(descrip, 'BLACKBODY ILLUMINANT');
83                start := starting_wavelength;
84                stop := ending_wavelength;
85                inc := minimum_increment;
86                planck_560 := Planck(560, t) / 100;
87                FOR i := 1 TO (stop - start) DIV inc + 1 DO
88                  val[i] := Planck(start + (i - 1) * inc, t) / planck_560;
89                END;
90              IF query_user THEN
91                BEGIN
92                jump(1);
93                readst('enter a description', temp.descrip, use_default, null_ok);
94                IF regis_terminal THEN
95                  BEGIN
96                  q := 'Y';
97                  readch('Plot this blackbody illuminant on your terminal ?', q, use_default, null_ok);
98                  IF q = 'Y' THEN
99                    plotrg_spectra('BLACKBODY ILLUMINANT PLOT', temp);
100                 END
101               ELSE
102                 return(2);
103               END;
104             MAKE_BLACKBODY_SPECTRA := temp;
105             END;
106         END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MAKE_BLACKBODY_SPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

  *** No lines with errors detected ***
```

```
PROGRAM MAKE_DAYLIGHT_SPECTRA;
 ($nolist)
 ($list)


  FUNCTION MAKE_DAYLIGHT_SPECTRA ((t: real; VAR error: boolean): spectral_data) ;
(
 WHERE
     t = correlated color temperature of illuminant to create (in automatic mode)
          or a value of <= 0 causes the user to be prompted for a value of temperature
     spec = spectral data record
 RESULTS
     make_daylight_spectra = spectra of daylight illuminant having the CCT specified
)

{ MAKE_DAYLIGHT_SPECTRA calculates the spectral power distribution of a daylight illuminant
at a desired correlated color temperature using the three characteristic
vectors from Grum's Optical Radiation Measurements Vol. 2 on Colorimetry
(pp 78-9) and returns the calculated values }

    VAR
      i, error_code: integer;
      temp: spectral_data;
      s: ARRAY [0..2] OF spectral_data;
      x, y, m1, m2: real;
      query_user, error0, error1, error2, error3: boolean;
      q: char;


    FUNCTION raise(y, x: real): real;

      BEGIN
        IF y > 0 THEN
          raise := exp(x * ln(y))
        ELSE IF y < 0 THEN
          raise := - 999 { signal error on negative y values }
        ELSE
          y := 0; { assume 0^x = 0 }
      END;


    BEGIN
      error := false;
      zero_spectra(temp);

      { get characteristic vectors s0, s1, and s2 from disk }
      stass(s[0].filename, 'CLR:DAYLIGHT_VECTORS.DAT');
      retrieve_mspectra(s, 0, 1, 3, error_code);
      error := error_code <> 0;
      say_error_code('ERROR reading CLR:DAYLIGHT_VECTORS.DAT', error_code, true);
      IF NOT error THEN
        BEGIN

        { select between user query mode and automatic mode }
        query_user := t <= 0;
```

```
 60
 61              IF query_user THEN
 62                BEGIN
 63                prompt(1);
 64                writec('MAKE DAYLIGHT ILLUMINANT');
 65                jump(3);
 66                writec('The correlated color temperature desired must be entered');
 67                jump(2);
 68                getre('enter temperature (K)', t, no_default, 4000, 25000);
 69                END;
 70
 71              { range check for the automatic mode }
 72              error := (t < 4000) OR (t > 25000);
 73
 74              IF NOT error THEN
 75                BEGIN
 76                IF t < 7000 THEN
 77                  x := - 4.6070E+9 / sqr(t) / t + 2.9678E+6 / sqr(t) + 0.09911E+3 / t + 0.244063
 78                ELSE
 79                  x := - 2.0064E+9 / sqr(t) / t + 1.9018E+6 / sqr(t) + 0.24748E+3 / t + 0.237040;
 80                y := - 3 * sqr(x) + 2.870 * x - 0.275;
 81                m1 := ( - 1.3515 - 1.7703 * x + 5.9114 * y) / (0.0241 + 0.2562 * x - 0.7341 * y);
 82                m2 := (0.03 - 31.4424 * x + 30.0717 * y) / (0.0241 + 0.2562 * x - 0.7341 * y);
 83
 84                temp := math(s[0], math_constant(s[1], m1, math_constant_multiply, error1), math_add, error0);
 85                temp := math(temp, math_constant(s[2], m2, math_constant_multiply, error2), math_add, error3);
 86                error := error0 OR error1 OR error2 OR error3;
 87                IF error THEN
 88                  BEGIN
 89                  jump(1);
 90                  writeln('ERROR calculating daylight distribution', chr(7), chr(7));
 91                  jump(1);
 92                  END;
 93                stass(temp.descrip, 'DAYLIGHT ILLUMINANT');
 94                IF query_user THEN
 95                  BEGIN
 96                  jump(1);
 97                  readst('enter a description', temp.descrip, use_default, null_ok);
 98                  IF regis_terminal THEN
 99                    BEGIN
100                    q := 'Y';
101                    readch('Plot this daylight illuminant on your terminal ?', q, use_default, null_ok);
102                    IF q = 'Y' THEN
103                      plotrg_spectra('OAYLIGHT ILLUMINANT PLOT', temp);
104                    END
105                  ELSE
106                    return(2);
107                  END;
108                MAKE_DAYLIGHT_SPECTRA := temp;
109                END;
110              END;
111          END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MAKE_DAYLIGHT_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

  *** No lines with errors detected ***
```

```
    1       PROGRAM MAKE_SPECTRA_FROM_BIG_SPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE MAKE_SPECTRA_FROM_BIG_SPECTRA ((VAR bspec: big_spectral_data; VAR spec: spectral_data; start, stop, inc: inte
   11                                                  error: boolean)} ;
   12       {
   13        WHERE
   14            bspec = a spectral data record to containing 1 nm data
   15            spec = a spectral data record to contain 5, 10 or 20 nm data
   16            error = signals an error in the routine
   17        RESULTS
   18            a spectral data record is created from a big spectral data record
   19       }
   20
   21           VAR
   22             i: integer;
   23             out: text;
   24
   25           BEGIN
   26             zero_spectra(spec);
   27             error := NOT (inc IN [5, 10, 20]) OR ((stop - start) MOD inc <> 0) OR (stop <= start) OR
   28                      ((start - starting_wavelength) MOD minimum_increment <> 0);
   29             IF NOT error THEN
   30               BEGIN
   31               spec.filename := bspec.filename;
   32               spec.descrip := bspec.descrip;
   33               spec.start := start;
   34               spec.stop := stop;
   35               spec.inc := inc;
   36               FOR i := start TO stop DO
   37                 IF (i - start) MOD inc = 0 THEN
   38                   spec.val[(i - start) DIV inc + 1] := bspec.val[i];
   39               END;
   40           END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MAKE_SPECTRA_FROM_BIG_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM MAKE_WEIGHTS;
    2        ($nolist)
    7        ($list)
    8
    9
   10         PROCEDURE MAKE_WEIGHTS ((spec: spectral_data; cmf: ARRAY [clow..chigh: integer] OF big_spectral_data; VAR wspec: ARRAY
   11                                  integer] OF spectral_data; starting_array_index, start, stop, inc: integer; VAR error: boolean)
   12       (
   13        WHERE
   14            spec = spectral data of light source to calculate weights from
   15            cmf = 1 nm color matching functions
   16            wspec = array of spectral data to contain the calculated weights
   17            starting_array_index = index for writing first weight spectra to wspec
   18            start = starting wavelength for weights returned
   19            stop = ending wavelength for weights returned
   20            inc = increment for weights returned
   21            error = signals an error condition
   22        RESULTS
   23            ASTM like weights are loaded into an array (to be used or saved)
   24       )
   25
   26       { this procedure calculates the Lagrange coefficeints for the ASTM weight
   27         calculations using a reference by Hugh Fairman, "The Calculation of
   28         Weight Factors for Tristimulus Integration", CR&A, Vol. 10, No. 4, pp.
   29         199 - 203 }
   30
   31           TYPE
   32             coef_3pt = ARRAY [0..2, 1..19] OF real;
   33             coef_4pt = ARRAY [0..3, 1..19] OF real;
   34
   35           VAR
   36             pt3: coef_3pt;
   37             pt4: coef_4pt;
   38             r: real;
   39             source: big_spectral_data;
   40             i, j, k, l, num: integer;
   41
   42           BEGIN
   43             { check to make sure there's room for the weights in the final arrays }
   44             error := (high - starting_array_index < 2);
   45             IF NOT error THEN
   46               calc_lagrange(source, spec, error);
   47             IF NOT error THEN
   48               BEGIN
   49               i := 0;
   50               REPEAT
   51                 i := i + 1;
   52                 cmf[i - 1 + clow] := big_math(source, cmf[i - 1 + clow], math_multiply, error);
   53               UNTIL error OR (i = 3);
   54               IF NOT error THEN
   55                 BEGIN
   56                 zero_spectra(wspec[starting_array_index]);
   57                 wspec[starting_array_index].start := start;
   58                 wspec[starting_array_index].stop := stop;
   59                 wspec[starting_array_index].inc := inc;
```

```
60                FOR i := 1 TO number_of_wavelengths DO
61                  wspec[starting_array_index].val[i] := 0;
62                FOR i := starting_array_index + 1 TO starting_array_index + 2 DO
63                  wspec[i] := wspec[starting_array_index];
64
65                { calculate the coefficeints for the first interval }
66                FOR i := 1 TO 19 DO
67                  BEGIN
68                  r := i / 20;
69                  pt3[0, i] := (r - 1) * (r - 2) / 2;
70                  pt3[1, i] := - r * (r - 2);
71                  pt3[2, i] := r * (r - 1) / 2;
72                  END;
73
74                { calculate the coefficeints for intermediate intervals }
75                FOR i := 1 TO 19 DO
76                  BEGIN
77                  r := 1 + i / 20;
78                  pt4[0, i] := - (r - 1) * (r - 2) * (r - 3) / 6;
79                  pt4[1, i] := r * (r - 2) * (r - 3) / 2;
80                  pt4[2, i] := - r * (r - 1) * (r - 3) / 2;
81                  pt4[3, i] := r * (r - 1) * (r - 2) / 6;
82                  END;
83
84                num := (stop - start) DIV inc + 1; { number of weights in the array }
85
86                FOR i := 1 TO 3 DO { loop through 3 color matching functions }
87                  BEGIN
88
89                  { load values for the wavelengths to be measured }
90                  FOR k := 1 TO num DO
91                    wspec[i - 1 + starting_array_index].val[k] := cmf[i - 1 + clow].val[(k - 1) * inc + start];
92
93                  { 3 pt interpolation for first interval }
94                  FOR j := 1 TO inc - 1 DO
95                    FOR k := 1 TO 3 DO
96                      BEGIN
97                      r := pt3[k - 1, j * 20 DIV inc] * cmf[i - 1 + clow].val[j + start];
98                      wspec[i - 1 + starting_array_index].val[k] := wspec[i - 1 + starting_array_index].val[k] + r;
99                      END;
100
101                 { 3 pt interpolation for last interval }
102                 FOR j := 1 TO inc - 1 DO
103                   FOR k := num - 2 TO num DO
104                     BEGIN
105                     r := pt3[num - k, (inc - j) * 20 DIV inc] * cmf[i - 1 + clow].val[(num - 2) * inc + start + j];
106                     wspec[i - 1 + starting_array_index].val[k] := wspec[i - 1 + starting_array_index].val[k] + r;
107                     END;
108
109                 { 4 pt interpolation for intermediate intervals }
110                 FOR j := 1 TO num - 1 - 2 DO { loop through num-1 intervals }
111                   FOR k := 1 TO inc - 1 DO { for all missing data }
112                     FOR l := 0 TO 3 DO { for all 4 points of an interpolating set }
113                       BEGIN
```

```
114                     r := pt4[l, k * 20 DIV inc] * cmf[i - 1 + clow].val[inc * j + start + k];
115                     wspec[i - 1 + starting_array_index].val[j + l] := wspec[i - 1 + starting_array_index].val[j + l] + r;
116                     END;
117
118               { add any excess values in front of start to the first weight }
119               FOR j := 360 TO start - 1 DO
120                 wspec[i - 1 + starting_array_index].val[1] := wspec[i - 1 + starting_array_index].val[1] + cmf[i - 1 + clow
121
122               { add any excess values past stop to the last weight }
123               FOR j := stop + 1 TO 830 DO
124                 wspec[i - 1 + starting_array_index].val[num] := wspec[i - 1 + starting_array_index].val[num] + cmf[i - 1 +
125                                                                 clow].val[j];
126
127               END;
128
129             { determine normalizing constant }
130             r := 0;
131             FOR j := 360 TO 830 DO
132               r := r + cmf[2 - 1 + clow].val[j];
133
134             { normalize the weights }
135             FOR i := 1 TO 3 DO { loop through 3 color matching functions }
136               FOR j := 1 TO num DO
137                 wspec[i - 1 + starting_array_index].val[j] := wspec[i - 1 + starting_array_index].val[j] * 100 / r;
138
139             wspec[starting_array_index + 0].name := 'Xwht      ';
140             wspec[starting_array_index + 0].descrip := 'Weights calculated using ASTM methods   ';
141             wspec[starting_array_index + 1].name := 'Ywht      ';
142             wspec[starting_array_index + 1].descrip := 'Weights calculated using ASTM methods   ';
143             wspec[starting_array_index + 2].name := 'Zwht      ';
144             wspec[starting_array_index + 2].descrip := 'Weights calculated using ASTM methods   ';
145             END;
146           END;
147         IF error THEN
148           BEGIN
149           writec('error encountered in calculation of weights');
150           return(1);
151           END;
152       END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MAKE_WEIGHTS.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM MATH;
 {$nolist}
 {$list}


  FUNCTION MATH ((spec1, spec2: spectral_data; option: math_options; VAR error: boolean): spectral_data) ;
{
 WHERE
     spec1                = spectral data record number 1
     spec2                = spectral data record number 2
     option               = type of operation to perform
     error                = signals error in parameters
 RESULTS
     a spectral data record is loaded with the selected math operation
}

    VAR
      i: integer;
      temp: spectral_data;

    BEGIN
      zero_spectra(temp);
      error := false;

      IF spec1.start = 0 THEN
        BEGIN
        error := true;
        writec('%MATH-WARNING !!! data not in memory for sample 1');
        END;

      IF spec2.start = 0 THEN
        BEGIN
        error := true;
        writec('%MATH-WARNING !!! data not in memory for sample 2');
        END;

{  if (spec1.start <> spec2.start) or (spec1.stop <> spec2.stop) or
     (spec1.inc <> spec2.inc) then
  begin
    error := true;
    writec('%MATH-WARNING !!! spectral data mismatch');
  end; }

      IF spectra_debug OR error THEN
        BEGIN
        writeln('%MATH-INFO #1 START=', spec1.start: 0, ' END=', spec1.stop: 0, ' INC=', spec1.inc: 0);
        writeln('%MATH-INFO #2 START=', spec2.start: 0, ' END=', spec2.stop: 0, ' INC=', spec2.inc: 0);
        write('%MATH-INFO Function Selected :  ');
        CASE option OF
          math_add:
            writeln('math_add');
          math_subtract_1_from_2:
            writeln('math_subtract_1_from_2');
          math_subtract_2_from_1:
            writeln('math_subtract_2_from_1');
```

```
60                math_multiply:
61                  writeln('math_multiply');
62                math_divide_1_by_2:
63                  writeln('math_divide_1_by_2');
64                math_divide_2_by_1:
65                  writeln('math_divide_2_by_1');
66                OTHERWISE
67                  writeln('UNKNOWN FUNCTION');
68                END;
69              IF spectra_debug THEN
70                spectra_debug_pause_execution
71              ELSE
72                return(2);
73              END;
74
75            IF NOT error THEN
76              BEGIN
77
78              temp := spec1; { temp assumes many characteristics of the first set of spectral data }
79              FOR i := 1 TO number_of_wavelengths DO
80                temp.val[i] := 0;
81
82              { find start, stop, inc of the combined files }
83              IF spec1.start > spec2.start THEN
84                temp.start := spec1.start
85              ELSE
86                temp.start := spec2.start;
87              IF spec1.stop < spec2.stop THEN
88                temp.stop := spec1.stop
89              ELSE
90                temp.stop := spec2.stop;
91              IF spec1.inc > spec2.inc THEN
92                temp.inc := spec1.inc
93              ELSE
94                temp.inc := spec2.inc;
95
96              IF spectra_debug THEN
97                BEGIN
98                writeln('%MATH-INFO Wavelength parameters for math operations');
99                writeln('%MATH-INFO START=', temp.start, ' END=', temp.stop, ' INC=', temp.inc);
100               spectra_debug_pause_execution;
101               END;
102
103             temp.filename := '                                                                    ';
104
105             i := temp.start;
106
107             REPEAT
108               CASE option OF
109
110                 math_add:
111                   temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1] + spec2.val[(
112                                                                  spec2.start) DIV spec2.inc + 1];
113
```

```
114                 math_subtract_1_from_2:
115                   temp.val[(i - temp.start) DIV temp.inc + 1] := spec2.val[(i - spec2.start) DIV spec2.inc + 1] - spec1.val[(
116                                                                   spec1.start) DIV spec1.inc + 1];
117
118                 math_subtract_2_from_1:
119                   temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1] - spec2.val[(
120                                                                   spec2.start) DIV spec2.inc + 1];
121
122                 math_multiply:
123                   temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1] * spec2.val[(
124                                                                   spec2.start) DIV spec2.inc + 1];
125
126                 math_divide_1_by_2:
127                   IF abs(spec2.val[(i - spec2.start) DIV spec2.inc + 1]) > 1E-30 THEN
128                     temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1] / spec2.val
129                                                                    spec2.start) DIV spec2.inc + 1]
130                   ELSE
131                     temp.val[(i - temp.start) DIV temp.inc + 1] := - 999;
132
133                 math_divide_2_by_1:
134                   IF abs(spec1.val[(i - spec1.start) DIV spec1.inc + 1]) > 1E-30 THEN
135                     temp.val[(i - temp.start) DIV temp.inc + 1] := spec2.val[(i - spec2.start) DIV spec2.inc + 1] / spec1.val
136                                                                    spec1.start) DIV spec1.inc + 1]
137                   ELSE
138                     temp.val[(i - temp.start) DIV temp.inc + 1] := - 999;
139                 END;
140               i := i + temp.inc;
141             UNTIL i > temp.stop;
142             END;
143           MATH := temp;
144         END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MATH.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM MATH_CONSTANT;
 ($nolist)
 ($list)


  FUNCTION MATH_CONSTANT ((spec: spectral_data; r: real; option: math_constant_options; VAR error: boolean): spectral_dat
{
 WHERE
     spec = spectral data record
     r = constant value for operation
     option = type of operation to perform
     error = signals error in parameters
 RESULTS
     a spectral data record is loaded with the selected math operation
}

    VAR
      i: integer;
      temp: spectral_data;
      temp_error: boolean;


    FUNCTION raise(y, x: real): real;

      BEGIN
        IF y > 0 THEN
          raise := exp(x * ln(y))
        ELSE IF y < 0 THEN
          raise := - 999 { signal error on negative y values }
        ELSE
          y := 0; { assume 0^x = 0 }
      END;


    BEGIN
      zero_spectra(temp);
      error := false;

      IF spec.start = 0 THEN
        BEGIN
        error := true;
        writec('%MATH_CONSTANT-WARNING !!! data not in memory to sum');
        END;

      IF spectra_debug OR error THEN
        BEGIN
        writeln('%MATH_CONSTANT-INFO #1 START=', spec.start, ' END=', spec.stop, ' INC=', spec.inc);
        write('%MATH_CONSTANT-INFO Function Selected :  ');
        CASE option OF
          math_constant_add:
            writeln('math_constant_add');
          math_constant_multiply:
            writeln('math_constant_multiply');
          math_constant_log10:
            writeln('math_constant_log10');
```

```
 60                math_constant_unlog10:
 61                  writeln('math_constant_unlog10');
 62                math_constant_raise:
 63                  writeln('math_constant_raise');
 64                math_constant_k_over_s:
 65                  writeln('math_constant_k_over_s');
 66                OTHERWISE
 67                  writeln('UNKNOWN FUNCTION');
 68                END;
 69              IF spectra_debug THEN
 70                spectra_debug_pause_execution
 71              ELSE
 72                return(2);
 73              END;
 74
 75            IF NOT error THEN
 76              BEGIN
 77              i := spec.start;
 78              temp := spec;
 79              REPEAT
 80                CASE option OF
 81                  math_constant_add:
 82                    temp.val[(i - spec.start) DIV spec.inc + 1] := spec.val[(i - spec.start) DIV spec.inc + 1] + r;
 83                  math_constant_multiply:
 84                    temp.val[(i - spec.start) DIV spec.inc + 1] := spec.val[(i - spec.start) DIV spec.inc + 1] * r;
 85                  math_constant_log10:
 86                    IF spec.val[(i - spec.start) DIV spec.inc + 1] * r <= 0 THEN
 87                      temp.val[(i - spec.start) DIV spec.inc + 1] := - 999
 88                    ELSE
 89                      temp.val[(i - spec.start) DIV spec.inc + 1] := ln(spec.val[(i - spec.start) DIV spec.inc + 1] * r) / ln(1
 90                  math_constant_unlog10:
 91                    temp.val[(i - spec.start) DIV spec.inc + 1] := exp(spec.val[(i - spec.start) DIV spec.inc + 1] * r * ln(10)
 92                  math_constant_raise:
 93                    temp.val[(i - spec.start) DIV spec.inc + 1] := raise(spec.val[(i - spec.start) DIV spec.inc + 1], r);
 94                  math_constant_k_over_s:
 95                    temp.val[(i - spec.start) DIV spec.inc + 1] := sqr(1 - spec.val[(i - spec.start) DIV spec.inc + 1]) / 2 /
 96                                                                   spec.val[(i - spec.start) DIV spec.inc + 1];
 97                  END;
 98                i := i + spec.inc;
 99              UNTIL i > spec.stop;
100              END;
101
102            MATH_CONSTANT := temp;
103          END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]MATH_CONSTANT.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM PLOTRG_LOCUS;
    2        ($nolist)
    7        ($list)
    8
    9
   10         PROCEDURE PLOTRG_LOCUS ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR cmf: ARRAY [low..high: integer] OF big_spe
   11                                  observer_name: PACKED ARRAY [olow..ohigh: integer] OF char; source, sample: coordinate_data)) ;
   12       {
   13        WHERE
   14            s = string to display
   15            cmf = 1 nm color matching functions
   16            observer_name = 02DEG or 10DEG for indicating the observer
   17            source = color coordinates of the light source
   18            sample = color coordinates of the sample
   19        RESULTS
   20            a 1931 (or 1964) CIE chromaticity diagram is drawn on the screeen
   21       }
   22
   23           CONST
   24             xmin = 0.0000;
   25             xmax = 0.8;
   26             ymin = 0.0000;
   27             ymax = 0.9;
   28
   29           VAR
   30             i, ii, error_code: integer;
   31             temp, xt, yt: real;
   32             error: boolean;
   33             ss: STRING [78];
   34
   35             %INCLUDE 'plb:plotrg';

plb:plotrg

2  1        { This code is for a sub program
2  2          Main Program File Name :  PLOTRG.PAS
2  3          Other subprograms needed :  any that use these MIL-PLOT facilities
2  4          Compiling Command : N/A
2  5          Linking Command :  N/A
2  6          Hardware Required : VT-241 REGIS TERMINAL
2  7
2  8       Software Modifications/ Date/ Programmer
2  9          Procedures Written/ 02-18-87/ R.M. Miller
2 10
2 11
2 12                              BRIEF PROGRAM DESCRIPTION
2 13
2 14
2 15                             MIL-PLOT Plotting Routines
2 16
2 17       These procedures enable a user to easily plot data, shapes, labels, etc.,
2 18       by calling various Pascal procedures. To use these procedures in a program
2 19       the following directive must appear after the variable declaration area for
2 20       the main program:
2 21
```

```
2  22          include 'pas:plotrg.pas';
2  23
2  24      The following procedures are included in the package:
2  25
2  26              RG_start
2  27              RG_set_write_mode(mode: RG_writing_modes);
2  28              RG_clear_screen;
2  29              RG_graphics(z: RG_control);
2  30              RG_move(x,y: real);
2  31              RG_draw(x,y: real);
2  32              RG_scale(xmin,xmax,ymin,ymax: real);
2  33              RG_xaxis(yinter,tick_spacing,tick_length,xmin,xmax: real);
2  34              RG_yaxis(xinter,tick_spacing,tick_length,ymin,ymax: real);
2  35              RG_wfreal(value: real; width: integer);
2  36              RG_wreal(r: real; width,ext: integer);
2  37              RG_winteger(i,width: integer);
2  38              RG_set_line_type(n: integer);
2  39              RG_set_text_size(size: integer);
2  40              RG_set_text_direction(angle: integer);
2  41              RG_write_text(s: packed array [low..high: integer] of char);
2  42              RG_set_hls_color(map,h,l,s: integer);
2  43              RG_set_color(map:integer;c: rg_colors);
2  44              RG_set_output_map(map: integer);
2  45
2  46         end of informational heading }
2  47
2  48
2  49      type RG_control=(RG_on,RG_off);
2  50           RG_character_set=(RG_standard,RG_alternate);
2  51           RG_writing_modes=(complement,RG_erase,overlay,replace,normal);
2  52           RG_colors=(red,green,blue,cyan,magenta,yellow,black,white);
2  53
2  54      var  RG_CSI,RG_DCS,RG_ST: packed array [1..2] of char;
2  55           rg_xslope,rg_yslope,rg_xintercept,rg_yintercept: real;
2  56
2  57      procedure RG_start;
2  58      begin
2  59        rg_dcs[1]:=chr(27);
2  60        rg_dcs[2]:='P';
2  61        rg_csi[1]:=chr(27);
2  62        rg_csi[2]:='[';
2  63        rg_st[1]:=chr(27);
2  64        rg_st[2]:='\';
2  65        rg_xslope:=1;
2  66        rg_xintercept:=0;
2  67        rg_yslope:=1;
2  68        rg_yintercept:=0;
2  69      end;
2  70
2  71
2  72      procedure RG_set_write_mode(mode: RG_writing_modes);
2  73      begin
2  74        write('W(');
2  75        case mode of
```

```
2  76          complement       : write('C');
2  77          RG_erase         : write('E');
2  78          normal,overlay   : write('V');
2  79          replace          : write('R');
2  80        end;
2  81        writeln(')');
2  82      end;
2  83
2  84
2  85
2  86      procedure RG_clear_screen;
2  87      begin
2  88        writeln('S(E)');
2  89      end;
2  90
2  91      procedure RG_graphics(z: RG_control);
2  92      begin
2  93       {                              '3p' will display REGIS commands}
2  94        if z=RG_on then writeln(rg_dcs,'p') else writeln(rg_st);
2  95      end;
2  96
2  97      procedure RG_move(x,y: real);
2  98      begin
2  99        writeln('P[',round(x*rg_xslope+rg_xintercept):1,',',
2 100                    round(y*rg_yslope+rg_yintercept):1,'j');
2 101      end;
2 102
2 103      procedure RG_draw(x,y: real);
2 104      begin
2 105        writeln('V[',round(x*rg_xslope+rg_xintercept):1,',',
2 106                    round(y*rg_yslope+rg_yintercept):1,'j');
2 107      end;
2 108
2 109      procedure RG_scale(xmin,xmax,ymin,ymax: real);
2 110      begin
2 111        rg_xslope:=799/(xmax-xmin);
2 112        rg_xintercept:=-xmin*rg_xslope;
2 113        rg_yslope:=-479/(ymax-ymin);
2 114        rg_yintercept:=-ymax*rg_yslope;
2 115      end;
2 116
2 117      procedure RG_wfreal(value: real; width,ext: integer);
2 118
2 119      var
2 120        log_value,width2 : real ;
2 121        width3 : integer ;
2 122
2 123      BEGIN
2 124
2 125        write('T''');
2 126
2 127        if value <> 0
2 128          then log_value := ln(abs(value))/ln(10)
2 129          else log_value := 0 ;
```

```
2 130
2 131         if value < 0.0
2 132           then width2 := width-1
2 133           else width2 := width ;
2 134
2 135         if ext > (trunc(width2)-2) then ext := trunc(width2)-2 ;
2 136         if ext < 0 then ext := 0 ;
2 137
2 138         if (trunc(log_value+1) > width2)
2 139                 or ( (log_value < 0) and ((-log_value+2) > width2) ) { 2 for "0." }
2 140           then
2 141             write(value:width )              { print in SCI Notation }
2 142           else
2 143             if ( (-log_value) > (ext) )
2 144               then
2 145                 write(value:width:trunc(width2-2))
2 146               else
2 147                 if ((ext) > (width2-trunc(log_value)-2))
2 148                   then
2 149                     begin
2 150                       ext := trunc(width2-log_value-2) ;
2 151                       if ext < 0 then ext := 0 ;
2 152                       write ( value:width:ext) ;
2 153                     end
2 154                   else
2 155                     write ( value:width:ext ) ;
2 156
2 157         write('''');
2 158
2 159       END;
2 160
2 161       procedure RG_winteger(i,width: integer);
2 162       begin
2 163         write('T''',i:width,'''');
2 164       end;
2 165
2 166
2 167       procedure RG_set_line_type(n: integer);
2 168       begin
2 169         if (n<0) or (n>9) then n:=1;
2 170         write('W(P',n:1,')');
2 171       end;
2 172
2 173       procedure RG_set_text_size(size: integer);
2 174       begin
2 175         if size<0 then size:=0 else if size>16 then size:=16;
2 176         write('T(S',size:1,')');
2 177       end;
2 178
2 179       procedure RG_set_text_direction(angle: integer);
2 180       begin
2 181         angle:=45*round(angle/45);
2 182         write('T(D',angle:1,',S0)');
2 183       end;
```

```
2 184
2 185       procedure RG_write_text(s: packed array [low..high: integer] of char);
2 186       var i: integer;
2 187       begin
2 188         i:=high;
2 189         while (i>low) and (s[i] in [chr(0),' ']) do i:=i-1;
2 190         write('T''');
2 191         for i:=low to i do write(s[i]);
2 192         write('''');
2 193       end;
2 194
2 195       procedure RG_set_hls_color(map,h,l,s: integer);
2 196       begin
2 197         if map>3 then map:=3 else if map<0 then map:=0;
2 198         write('S(M',map:1,'(AH',h:1,'L',l:1,'S',s:1,'))');
2 199       end;
2 200
2 201       procedure RG_set_color(map:integer;c: rg_colors);
2 202       type rg_color_list=packed array[rg_colors] of char;
2 203       const rg_color=rg_color_list('R','G','B','C','M','Y','D','W');
2 204       begin
2 205         if map>3 then map:=3 else if map<0 then map:=0;
2 206         write('S(M',map:1,'(A',rg_color[c],'))');
2 207       end;
2 208
2 209       procedure RG_set_output_map(map: integer);
2 210       begin
2 211         if map>3 then map:=3 else if map<0 then map:=0;
2 212         write('W(I',map:1,')');
2 213       end;
2 214
2 215       procedure RG_xaxis(yinter,tick_spacing,tick_length,xmin,xmax: real);
2 216       var inc,x: real;
2 217           tick, n_ticks : integer;
2 218
2 219       begin
2 220         if tick_spacing = 0
2 221           then inc := abs(xmax-xmin)
2 222           else inc := abs(tick_spacing);
2 223         if xmax < xmin then inc := -inc ;
2 224         n_ticks := trunc( 1.00001+(xmax-xmin)/inc ) ; { constant allows for FP error }
2 225         x:=xmin;
2 226         rg_move(xmin,yinter);
2 227         for tick := 1 to n_ticks do
2 228           BEGIN
2 229             rg_draw(x,yinter);
2 230             rg_draw(x,yinter-tick_length);
2 231             rg_move(x,yinter);
2 232             x:=x+inc;
2 233           END;
2 234         rg_draw(xmax,yinter);
2 235       end;
2 236
2 237       procedure RG_xlabel(yinter,tick_spacing,tick_length,xmin,xmax: real;
```

```
2 238                                                   width,ext: integer);
2 239
2 240       var inc,x: real;
2 241           tick, n_ticks, i : integer;
2 242
2 243       begin
2 244
2 245         if tick_spacing = 0
2 246           then inc := abs(xmin-xmax)
2 247           else inc := abs(tick_spacing);
2 248         if xmax < xmin then inc := -inc ;
2 249         n_ticks := trunc( 1.00001+(xmax-xmin)/inc ) ; { constant allows for FP error }
2 250         x:=xmin;
2 251         for tick := 1 to n_ticks do
2 252           BEGIN
2 253             rg_move(x,yinter-(tick_length*1.1));
2 254             write( 'T"' );
2 255             for i:=1 to (width div 2 ) do write(chr(8));
2 256       {       if (trunc(ln(x)/ln(10)) mod 2 ) = 0 then write( '"0"', chr(8) ); }
2 257             write('"');
2 258             RG_wfreal(x,width,ext) ;
2 259             x:=x+inc ;
2 260           END;
2 261
2 262       end;
2 263
2 264       procedure RG_yaxis(xinter,tick_spacing,tick_length,ymin,ymax: real);
2 265       var inc,y: real;
2 266           tick,n_ticks: integer;
2 267
2 268       begin
2 269         if tick_spacing = 0
2 270           then inc := abs(ymax-ymin)
2 271           else inc := abs(tick_spacing);
2 272         if ymax < ymin then inc := -inc ;
2 273         n_ticks := trunc( 1.00001+(ymax-ymin)/inc ) ; { constant allows for FP error }
2 274         y:=ymin;
2 275         rg_move(xinter,y);
2 276         for tick := 1 to n_ticks do
2 277           BEGIN
2 278             rg_draw(xinter,y);
2 279             rg_draw(xinter-tick_length,y);
2 280             rg_move(xinter,y);
2 281             y:=y+inc;
2 282           END;
2 283         rg_draw(xinter,ymax);
2 284       end;
2 285
2 286       procedure RG_ylabel(xinter,tick_spacing,tick_length,ymin,ymax: real;
2 287                                                   width,ext: integer);
2 288
2 289       var inc,y: real;
2 290           tick,n_ticks,i: integer;
2 291
```

```
2 292      begin
2 293
2 294        if tick_spacing = 0
2 295          then inc := abs(ymax-ymin)
2 296          else inc := abs(tick_spacing);
2 297        if ymax < ymin then inc := -inc ;
2 298        n_ticks := trunc( 1.00001+(ymax-ymin)/inc ) ; { constant allows for FP error }
2 299        y:=ymin;
2 300        for tick := 1 to n_ticks do
2 301          BEGIN
2 302            rg_move(xinter-(tick_length*1.2),y);
2 303            write( 'T""2"');
2 304            for i:=1 to (width) do write(chr(8));
2 305            write('"');
2 306            RG_wfreal(y,width,ext) ;
2 307            y:=y+inc ;
2 308          END;
2 309
2 310      end;
```

USER1:[RMM.COLOR.SPECTRA.SOURCE]PLOTRG_LOCUS.PASF;3

```
  36
  37
  38          PROCEOURE small_box(x, y: real);
  39
  40            CONST
  41              tick = 0.008;
  42
  43            BEGIN
  44              rg_move(x + tick, y + tick);
  45              rg_draw(x + tick, y - tick);
  46              rg_draw(x - tick, y - tick);
  47              rg_draw(x - tick, y + tick);
  48              rg_draw(x + tick, y + tick);
  49            END;
  50
  51
  52          PROCEDURE small_triangle(x, y: real);
  53
  54            CONST
  55              tick = 0.008;
  56
  57            BEGIN
  58              rg_move(x - tick, y - tick);
  59              rg_draw(x, y + tick);
  60              rg_draw(x + tick, y - tick);
  61              rg_draw(x - tick, y - tick);
  62              rg_move(x + tick, y + tick);
  63            END;
  64
  65
  66          PROCEDURE debugr;
  67
```

```
68            BEGIN
69      {     rg_graphics(rg_off);
70          writeln(chr(7),'One more procedure passed');
71          rg_graphics(rg_on); }
72            END;
73
74
75          BEGIN
76            IF regis_terminal THEN
77              BEGIN
78              prompt(1);
79              rg_start;
80              { plotting stuff }
81              rg_graphics(rg_on);
82              rg_clear_screen;
83
84              debugr;
85              { find the minimum ymin values and maximum ymax values }
86
87              rg_scale(xmin - (xmax - xmin) * 1.0, xmax + (xmax - xmin) * 0.10, ymin - (ymax - ymin) * 0.20, ymax + (ymax - ymi
88              debugr;
89              writeln;
90              { draw a box at min and max x and y values }
91              rg_move(xmin, ymin);
92              rg_draw(xmin, ymax);
93              rg_draw(xmax, ymax);
94              rg_draw(xmax, ymin);
95              rg_draw(xmin, ymin);
96              writeln;
97
98              debugr;
99              { make ticks 2% the height of the screen }
100             rg_xaxis(ymin, 0.1, 0.02 * (ymax - ymin), xmin, xmax);
101             { label the x axis }
102             writeln;
103             debugr;
104             rg_xlabel(ymin, 0.2, 0.02 * (ymax - ymin), xmin, xmax, 4, 2);
105             writeln;
106             debugr;
107             { make ticks 2% of the width of the screen }
108             rg_yaxis(xmin, 0.10, 0.02 * (xmax - xmin), ymin, ymax);
109             writeln;
110             debugr;
111             { label the y axis }
112             rg_ylabel(xmin - (xmax - xmin) * 0.15, 0.2, 0.01 * (xmax - xmin), ymin, ymax, 4, 2);
113             debugr;
114             writeln;
115
116             { draw locus }
117             rg_set_output_map(2);
118             rg_move(cmf[4].val[360], cmf[5].val[360]);
119             FOR i := 360 DIV 2 TO 700 DIV 2 DO
120               rg_draw(cmf[4].val[i * 2], cmf[5].val[i * 2]);
121             rg_set_line_type(2);
```

```
122             rg_draw(cmf[4].val[360], cmf[5].val[360]);
123             xt := source[c_x];
124             yt := source[c_y];
125             rg_draw(xt, yt);
126             rg_draw(cmf[4].val[700], cmf[5].val[700]);
127             rg_move(0.45, 0.15);
128             rg_draw(0.45 + 0.07, 0.15 - 0.05);
129             rg_move(0.40, 0.15 - 0.05);
130             rg_write_text('Complementary Region');
131
132             rg_move(cmf[4].val[400], cmf[5].val[400]);
133             rg_draw(cmf[4].val[400] - 0.04, cmf[5].val[400] + 0.025);
134             rg_move(cmf[4].val[400] - 0.1, cmf[5].val[400] + 0.05);
135             rg_winteger(400, 3);
136
137             rg_move(cmf[4].val[485], cmf[5].val[485]);
138             rg_draw(cmf[4].val[485] - 0.025, cmf[5].val[485] + 0.025);
139             rg_move(cmf[4].val[485] - 0.075, cmf[5].val[485] + 0.05);
140             rg_winteger(450, 3);
141
142             rg_move(cmf[4].val[500], cmf[5].val[500]);
143             rg_draw(cmf[4].val[500] - 0.025, cmf[5].val[500] + 0.025);
144             rg_move(cmf[4].val[500] - 0.075, cmf[5].val[500] + 0.05);
145             rg_winteger(500, 3);
146
147             rg_move(cmf[4].val[550], cmf[5].val[550]);
148             rg_draw(cmf[4].val[550] + 0.025, cmf[5].val[550] + 0.025);
149             rg_move(cmf[4].val[550] + 0.03, cmf[5].val[550] + 0.05);
150             rg_winteger(550, 3);
151
152             rg_move(cmf[4].val[600], cmf[5].val[600]);
153             rg_draw(cmf[4].val[600] + 0.025, cmf[5].val[600] + 0.025);
154             rg_move(cmf[4].val[600] + 0.03, cmf[5].val[600] + 0.05);
155             rg_winteger(600, 3);
156
157             rg_move(cmf[4].val[700], cmf[5].val[700]);
158             rg_draw(cmf[4].val[700], cmf[5].val[700] - 0.025);
159             rg_move(cmf[4].val[700] - 0.03, cmf[5].val[700] - 0.025);
160             rg_winteger(700, 3);
161
162             rg_set_line_type(1);
163             xt := source[c_x];
164             yt := source[c_y];
165             small_box(xt, yt);
166             small_box(0.5, 0.8);
167             rg_write_text(' Source');
168
169             rg_set_output_map(3);
170             xt := sample[c_x];
171             yt := sample[c_y];
172             small_triangle(xt, yt);
173             small_triangle(0.6, 0.7);
174             rg_write_text(' Sample');
175
```

```
176             rg_set_output_map(1);
177             ii := abs(round(sample[dominant_wavelength]));
178             xt := cmf[4].val[ii];
179             yt := cmf[5].val[ii];
180             rg_move(xt, yt);
181             IF sample[dominant_wavelength] > 0 THEN
182               rg_draw(source[c_x], source[c_y])
183             ELSE
184               rg_draw(sample[c_x], sample[c_y]);
185
186             rg_set_output_map(3);
187             rg_set_line_type(1);
188             rg_graphics(rg_off);
189
190             gotoxy(0, 2);
191             strass(ss, s);
192             writeln(ss);
193             jump(1);
194             strass(ss, observer_name);
195             writeln('Observer :  ', ss);
196             jump(1);
197             IF sample[dominant_wavelength] < 0 THEN
198               writeln('Complementary Wavelength :')
199             ELSE
200               writeln('Dominant Wavelength :');
201             writeln;
202             writeln(' ': 10, abs(sample[dominant_wavelength]): 7: 2, ' nm');
203             writeln;
204             writeln('Excitation Purity :');
205             writeln;
206             writeln(' ': 10, sample[excitation_purity]: 7: 4);
207             writeln;
208             writeln('Sample chromaticities :');
209             writeln;
210             writeln('    x : ', sample[c_x]: 6: 4, ' y : ', sample[c_y]: 6: 4);
211             writeln;
212             writeln('Source chromaticities :');
213             writeln;
214             writeln('    x : ', source[c_x]: 6: 4, ' y : ', source[c_y]: 6: 4);
215             gotoxy(0, 22);
216             return(1);
217             rg_graphics(rg_on);
218             rg_clear_screen;
219             rg_graphics(rg_off);
220           END;
221         END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]PLOTRG_LOCUS.PASF;3/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM PLOTRG_MSPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE PLOTRG_MSPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: ARRAY [low..high: integer] OF spe
   11                                     starting_spectra_array_index, number_of_spectra: integer)) ;
   12       {
   13        where
   14            s = string to display
   15            spec = array of spectral data record
   16            starting_spectra_array_index = element of array to start displaying  (range is low..high)
   17            number_of_spectra = number of spectra to be displayed
   18        RESULTS
   19            one or more spectral data records are plotted on a REGIS terminal
   20       }
   21
   22           VAR
   23             i, spec_start, spec_stop, display_start, display_stop, num, error_code, plot_no: integer;
   24             ymin, ymax, temp: real;
   25             regis_term, error: boolean;
   26             ss: STRING [78];
   27
   28             %INCLUDE 'plb:plotrg';

plb:plotrg

2  1        { This code is for a sub program
2  2          Main Program File Name :  PLOTRG.PAS
2  3          Other subprograms needed :  any that use these MIL-PLOT facilities
2  4          Compiling Command : N/A
2  5          Linking Command :  N/A
2  6          Hardware Required : VT-241 REGIS TERMINAL
2  7
2  8       Software Modifications/ Date/ Programmer
2  9          Procedures Written/ 02-18-87/ R.M. Miller
2 10
2 11
2 12                            BRIEF PROGRAM DESCRIPTION
2 13
2 14
2 15                           MIL-PLOT Plotting Routines
2 16
2 17       These procedures enable a user to easily plot data, shapes, labels, etc.,
2 18       by calling various Pascal procedures. To use these procedures in a program
2 19       the following directive must appear after the variable declaration area for
2 20       the main program:
2 21
2 22           include 'pas:plotrg.pas';
2 23
2 24       The following procedures are included in the package:
2 25
2 26               RG_start
2 27               RG_set_write_mode(mode: RG_writing_modes);
2 28               RG_clear_screen;
```

```
2  29              RG_graphics(z: RG_control);
2  30              RG_move(x,y: real);
2  31              RG_draw(x,y: real);
2  32              RG_scale(xmin,xmax,ymin,ymax: real);
2  33              RG_xaxis(yinter,tick_spacing,tick_length,xmin,xmax: real);
2  34              RG_yaxis(xinter,tick_spacing,tick_length,ymin,ymax: real);
2  35              RG_wfreal(value: real; width: integer);
2  36              RG_wreal(r: real; width,ext: integer);
2  37              RG_winteger(i,width: integer);
2  38              RG_set_line_type(n: integer);
2  39              RG_set_text_size(size: integer);
2  40              RG_set_text_direction(angle: integer);
2  41              RG_write_text(s: packed array [low..high: integer] of char);
2  42              RG_set_hls_color(map,h,l,s: integer);
2  43              RG_set_color(map:integer;c: rg_colors);
2  44              RG_set_output_map(map: integer);
2  45
2  46         end of informational heading )
2  47
2  48
2  49      type RG_control=(RG_on,RG_off);
2  50           RG_character_set=(RG_standard,RG_alternate);
2  51           RG_writing_modes=(complement,RG_erase,overlay,replace,normal);
2  52           RG_colors=(red,green,blue,cyan,magenta,yellow,black,white);
2  53
2  54      var  RG_CSI,RG_DCS,RG_ST: packed array [1..2] of char;
2  55           rg_xslope,rg_yslope,rg_xintercept,rg_yintercept: real;
2  56
2  57      procedure RG_start;
2  58      begin
2  59        rg_dcs[1]:=chr(27);
2  60        rg_dcs[2]:='P';
2  61        rg_csi[1]:=chr(27);
2  62        rg_csi[2]:='[';
2  63        rg_st[1]:=chr(27);
2  64        rg_st[2]:='\';
2  65        rg_xslope:=1;
2  66        rg_xintercept:=0;
2  67        rg_yslope:=1;
2  68        rg_yintercept:=0;
2  69      end;
2  70
2  71
2  72      procedure RG_set_write_mode(mode: RG_writing_modes);
2  73      begin
2  74        write('W(');
2  75        case mode of
2  76          complement       : write('C');
2  77          RG_erase         : write('E');
2  78          normal,overlay   : write('V');
2  79          replace          : write('R');
2  80        end;
2  81        writeln(')');
2  82      end;
```

```
2  83
2  84
2  85
2  86      procedure RG_clear_screen;
2  87      begin
2  88        writeln('S(E)');
2  89      end;
2  90
2  91      procedure RG_graphics(z: RG_control);
2  92      begin
2  93       {                              '3p' will display REGIS commands}
2  94        if z=RG_on then writeln(rg_dcs,'p') else writeln(rg_st);
2  95      end;
2  96
2  97      procedure RG_move(x,y: real);
2  98      begin
2  99        writeln('P[',round(x*rg_xslope+rg_xintercept):1,',',
2 100                    round(y*rg_yslope+rg_yintercept):1,']');
2 101      end;
2 102
2 103      procedure RG_draw(x,y: real);
2 104      begin
2 105        writeln('V[',round(x*rg_xslope+rg_xintercept):1,',',
2 106                    round(y*rg_yslope+rg_yintercept):1,']');
2 107      end;
2 108
2 109      procedure RG_scale(xmin,xmax,ymin,ymax: real);
2 110      begin
2 111        rg_xslope:=799/(xmax-xmin);
2 112        rg_xintercept:=-xmin*rg_xslope;
2 113        rg_yslope:=-479/(ymax-ymin);
2 114        rg_yintercept:=-ymax*rg_yslope;
2 115      end;
2 116
2 117      procedure RG_wfreal(value: real; width,ext: integer);
2 118
2 119      var
2 120        log_value,width2 : real ;
2 121        width3 : integer ;
2 122
2 123      BEGIN
2 124
2 125        write('T''');
2 126
2 127        if value <> 0
2 128          then log_value := ln(abs(value))/ln(10)
2 129          else log_value := 0 ;
2 130
2 131        if value < 0.0
2 132          then width2 := width-1
2 133          else width2 := width ;
2 134
2 135        if ext > (trunc(width2)-2) then ext := trunc(width2)-2 ;
2 136        if ext < 0 then ext := 0 ;
```

```
2 137
2 138         if (trunc(log_value+1) > width2)
2 139                 or ( (log_value < 0) and ((-log_value+2) > width2) ) { 2 for "0.' )
2 140           then
2 141             write(value:width )              { print in SCI Notation )
2 142           else
2 143             if ( (-log_value) > (ext) )
2 144               then
2 145                 write(value:width:trunc(width2-2))
2 146               else
2 147                 if ((ext) > (width2-trunc(log_value)-2))
2 148                   then
2 149                     begin
2 150                       ext := trunc(width2-log_value-2) ;
2 151                       if ext < 0 then ext := 0 ;
2 152                       write ( value:width:ext) ;
2 153                     end
2 154                   else
2 155                     write ( value:width:ext ) ;
2 156
2 157         write('''');
2 158
2 159       END;
2 160
2 161       procedure RG_winteger(i,width: integer);
2 162       begin
2 163         write('T''',i:width,'''');
2 164       end;
2 165
2 166
2 167       procedure RG_set_line_type(n: integer);
2 168       begin
2 169         if (n<0) or (n>9) then n:=1;
2 170         write('W(P',n:1,')');
2 171       end;
2 172
2 173       procedure RG_set_text_size(size: integer);
2 174       begin
2 175         if size<0 then size:=0 else if size>16 then size:=16;
2 176         write('T(S',size:1,')');
2 177       end;
2 178
2 179       procedure RG_set_text_direction(angle: integer);
2 180       begin
2 181         angle:=45*round(angle/45);
2 182         write('T(O',angle:1,',S0)');
2 183       end;
2 184
2 185       procedure RG_write_text(s: packed array [low..high: integer] of char);
2 186       var i: integer;
2 187       begin
2 188         i:=high;
2 189         while (i>low) and (s[i] in [chr(0),' ']) do i:=i-1;
2 190         write('T''');
```

```
2 191         for i:=low to i do write(s[i]);
2 192         write('''');
2 193       end;
2 194
2 195       procedure RG_set_hls_color(map,h,l,s: integer);
2 196       begin
2 197         if map>3 then map:=3 else if map<0 then map:=0;
2 198         write('S(M',map:1,'(AH',h:1,'L',l:1,'S',s:1,'))');
2 199       end;
2 200
2 201       procedure RG_set_color(map:integer;c: rg_colors);
2 202       type rg_color_list=packed array[rg_colors] of char;
2 203       const rg_color=rg_color_list('R','G','B','C','M','Y','D','W');
2 204       begin
2 205         if map>3 then map:=3 else if map<0 then map:=0;
2 206         write('S(M',map:1,'(A',rg_color[c],'))');
2 207       end;
2 208
2 209       procedure RG_set_output_map(map: integer);
2 210       begin
2 211         if map>3 then map:=3 else if map<0 then map:=0;
2 212         write('W(I',map:1,')');
2 213       end;
2 214
2 215       procedure RG_xaxis(yinter,tick_spacing,tick_length,xmin,xmax: real);
2 216       var inc,x: real;
2 217           tick, n_ticks : integer;
2 218
2 219       begin
2 220         if tick_spacing = 0
2 221           then inc := abs(xmax-xmin)
2 222           else inc := abs(tick_spacing);
2 223         if xmax < xmin then inc := -inc ;
2 224         n_ticks := trunc( 1.00001+(xmax-xmin)/inc ) ; { constant allows for FP error }
2 225         x:=xmin;
2 226         rg_move(xmin,yinter);
2 227         for tick := 1 to n_ticks do
2 228           BEGIN
2 229             rg_draw(x,yinter);
2 230             rg_draw(x,yinter-tick_length);
2 231             rg_move(x,yinter);
2 232             x:=x+inc;
2 233           END;
2 234         rg_draw(xmax,yinter);
2 235       end;
2 236
2 237       procedure RG_xlabel(yinter,tick_spacing,tick_length,xmin,xmax: real;
2 238                                                   width,ext: integer);
2 239
2 240       var inc,x: real;
2 241           tick, n_ticks, i : integer;
2 242
2 243       begin
2 244
```

```
2 245         if tick_spacing = 0
2 246           then inc := abs(xmin-xmax)
2 247           else inc := abs(tick_spacing);
2 248         if xmax < xmin then inc := -inc ;
2 249         n_ticks := trunc( 1.00001+(xmax-xmin)/inc ) ; { constant allows for FP error }
2 250         x:=xmin;
2 251         for tick := 1 to n_ticks do
2 252           BEGIN
2 253             rg_move(x,yinter-(tick_length*1.1));
2 254             write( 'T"' );
2 255             for i:=1 to (width div 2 ) do write(chr(8));
2 256       {      if (trunc(ln(x)/ln(10)) mod 2 ) = 0 then write( '"0"', chr(8) ); }
2 257             write('"');
2 258             RG_wfreal(x,width,ext) ;
2 259             x:=x+inc ;
2 260           END;
2 261
2 262       end;
2 263
2 264       procedure RG_yaxis(xinter,tick_spacing,tick_length,ymin,ymax: real);
2 265       var inc,y: real;
2 266           tick,n_ticks: integer;
2 267
2 268       begin
2 269         if tick_spacing = 0
2 270           then inc := abs(ymax-ymin)
2 271           else inc := abs(tick_spacing);
2 272         if ymax < ymin then inc := -inc ;
2 273         n_ticks := trunc( 1.00001+(ymax-ymin)/inc ) ; { constant allows for FP error }
2 274         y:=ymin;
2 275         rg_move(xinter,y);
2 276         for tick := 1 to n_ticks do
2 277           BEGIN
2 278             rg_draw(xinter,y);
2 279             rg_draw(xinter-tick_length,y);
2 280             rg_move(xinter,y);
2 281             y:=y+inc;
2 282           END;
2 283         rg_draw(xinter,ymax);
2 284       end;
2 285
2 286       procedure RG_ylabel(xinter,tick_spacing,tick_length,ymin,ymax: real;
2 287                                                   width,ext: integer);
2 288
2 289       var inc,y: real;
2 290           tick,n_ticks,i: integer;
2 291
2 292       begin
2 293
2 294         if tick_spacing = 0
2 295           then inc := abs(ymax-ymin)
2 296           else inc := abs(tick_spacing);
2 297         if ymax < ymin then inc := -inc ;
2 298         n_ticks := trunc( 1.00001+(ymax-ymin)/inc ) ; { constant allows for FP error }
```

```
2 299         y:=ymin;
2 300         for tick := 1 to n_ticks do
2 301           BEGIN
2 302             rg_move(xinter-(tick_length*1.2),y);
2 303             write( 'T""2"');
2 304             for i:=1 to (width) do write(chr(8));
2 305             write('"');
2 306             RG_wfreal(y,width,ext) ;
2 307             y:=y+inc ;
2 308           END;
2 309
2 310       end;
```

USER1:[RMM.COLOR.SPECTRA.SOURCE]PLOTRG_MSPECTRA.PASF;3

```
   29
   30
   31           BEGIN
   32             regis_term := regis_terminal;
   33
   34             IF regis_term THEN
   35               BEGIN
   36
   37               IF spectra_debug THEN
   38                 BEGIN
   39                 writeln('%PLOTRG_MSPECTRA-INFO starting_spectra_array_index = ', starting_spectra_array_index: 0);
   40                 writeln('%PLOTRG_MSPECTRA-INFO number_of_spectra = ', number_of_spectra: 0);
   41                 spectra_debug_pause_execution;
   42                 END;
   43
   44               prompt(1);
   45               strass(ss, s);
   46               writec(ss);
   47               jump(4);
   48
   49               rg_start;
   50
   51               error_code := 0; { everything's ok to start with }
   52
   53               IF NOT (starting_spectra_array_index IN [low..high]) THEN
   54                 error_code := - 1; { invalid array index }
   55
   56               IF error_code = 0 THEN
   57                 IF number_of_spectra < 1 THEN
   58                   error_code := - 2; { bad number parameter }
   59
   60               IF error_code = 0 THEN
   61                 IF NOT (starting_spectra_array_index + number_of_spectra - 1 IN [low..high]) THEN
   62                   error_code := - 2; { invalid array index addressed }
   63
   64               IF error_code <> 0 THEN
   65                 BEGIN
   66                 writeln('%PLOTRG_MSPECTRA-ERROR parameter error code = ', error_code: 0);
   67                 return(2);
```

```
68                END;
69
70              IF error_code = 0 THEN
71                BEGIN
72                spec_start := starting_spectra_array_index;
73                spec_stop := starting_spectra_array_index + number_of_spectra - 1;
74
75                num := 0;
76                FOR i := spec_start TO spec_stop DO
77                  IF spec[i].start > 1 THEN
78                    num := num + 1;
79                jump(1);
80                IF num > 0 THEN
81                  writec('List of spectra to be plotted')
82                ELSE
83                  BEGIN
84                  writec('No spectral data in array passed to plotting routine');
85                  jump(2);
86                  writec('Oescriptions in array passed to plotting routine');
87                  END;
88                jump(1);
89                FOR i := spec_start TO spec_stop DO
90                  BEGIN
91                  IF spec[i].start >= starting_wavelength THEN
92                    BEGIN
93                    write('No. ', i: 2, ' ');
94                    display_descrip(output, spec[i], 52, true, true, true);
95                    writeln;
96                    END;
97                  END;
98                return(2);
99
100               IF num > 0 THEN
101                 BEGIN
102
103                 { plotting stuff }
104                 rg_graphics(rg_on);
105                 rg_clear_screen;
106
107                 { find the minimum ymin values and maximum ymax values }
108                 ymin := 1E30;
109                 ymax := - 1E30;
110                 FOR i := spec_start TO spec_stop DO
111                   BEGIN
112                   IF spec[i].start >= starting_wavelength THEN
113                     BEGIN
114                     temp := spectra_ymin(spec[i]);
115                     IF temp < ymin THEN
116                       ymin := temp;
117                     temp := spectra_ymax(spec[i]);
118                     IF temp > ymax THEN
119                       ymax := temp;
120                     END;
121                   END;
```

```
122                 { check for max and min being equal or nearly so }
123                 IF abs(ymax - ymin) < 1E-20 THEN
124                   BEGIN
125                   IF ymax = 0 THEN
126                     BEGIN
127                     ymin := - 0.5;
128                     ymax := 0.5;
129                     END
130                   ELSE
131                     BEGIN
132                     ymin := 0.5 * ymin;
133                     ymax := 1.5 * ymax;
134                     END;
135                   END;
136
137                 rg_scale(starting_wavelength - (ending_wavelength - starting_wavelength) * 0.20,
138                         ending_wavelength + (ending_wavelength - starting_wavelength) * 0.10, ymin - (ymax - ymin) * 0.20,
139                         ymax + (ymax - ymin) * 0.05);
140                 writeln;
141                 { draw a box at min and max x and y values }
142                 rg_move(starting_wavelength, ymin);
143                 rg_draw(starting_wavelength, ymax);
144                 rg_draw(ending_wavelength, ymax);
145                 rg_draw(ending_wavelength, ymin);
146                 rg_draw(starting_wavelength, ymin);
147                 writeln;
148
149                 { make ticks 2% the height of the screen }
150                 rg_xaxis(ymin, 10, 0.02 * (ymax - ymin), starting_wavelength, ending_wavelength);
151                 { label the x axis }
152                 writeln;
153                 rg_xlabel(ymin, 50, 0.02 * (ymax - ymin), 400, 750, 4, 0);
154                 writeln;
155
156                 { make ticks 1% of the width of the screen }
157                 rg_yaxis(starting_wavelength, 0.10 * (ymax - ymin), 0.01 * (ending_wavelength - starting_wavelength), ymin, y
158                 writeln;
159
160                 { label the y axis }
161                 IF trunc(ln(abs(ymax - ymin)) / ln(10)) >= 0 THEN
162                   rg_ylabel(starting_wavelength, 0.10 * (ymax - ymin), 0.01 * (ending_wavelength - starting_wavelength), ymin
163                             6 - trunc(ln(abs(ymax - ymin)) / ln(10)) - 2)
164                 ELSE
165                   rg_ylabel(starting_wavelength - (ending_wavelength - starting_wavelength) * 0.15, 0.10 * (ymax - ymin),
166                             0.01 * (ending_wavelength - starting_wavelength), ymin, ymax, 6, 6 - 2);
167
168                 writeln;
169
170                 plot_no := 1;
171                 { loop for plotting each spectra }
172                 FOR i := spec_start TO spec_stop DO
173                   BEGIN
174                   IF spec[i].start >= starting_wavelength THEN
175                     BEGIN
```

```
176                     { loop through three output map colors }
177                     rg_set_output_map(3 - (plot_no - 1) MOD 3);
178                     { after changing colors, change the line types }
179                     rg_set_line_type(((i - spec_start) DIV 3 + 1) MOD 10);
180                     rg_move(spec[i].start, spec[i].val[1]);
181                     num := spec[i].start;
182                     REPEAT
183                       rg_draw(num, spec[i].val[(num - spec[i].start) DIV spec[i].inc + 1]);
184                       num := num + spec[i].inc;
185                     UNTIL num > spec[i].stop;
186                     writeln;
187                     rg_move(ending_wavelength + 5, ymax - (plot_no - 0.5) * (ymax - ymin) / 15); {plan on 15 labels for the k
188                     rg_winteger(i, 2);
189                     rg_draw(ending_wavelength + 40, ymax - (plot_no - 0.5) * (ymax - ymin) / 15); {plan on 15 labels for the
190                     plot_no := plot_no + 1;
191                     END;
192                   END;
193                 rg_set_output_map(3);
194                 rg_set_line_type(1);
195                 rg_graphics(rg_off);
196                 gotoxy(0, 22);
197                 return(1);
198                 rg_graphics(rg_on);
199                 rg_clear_screen;
200                 rg_graphics(rg_off);
201                 END;
202               END;
203             END;
204         END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]PLOTRG_MSPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM PLOTRG_SPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE PLOTRG_SPECTRA ((s: PACKED ARRAY [slow..shigh: integer] OF char; VAR spec: spectral_data)) ;
   11       {
   12        WHERE
   13            s = string to display
   14            spec = spectral data record
   15        RESULTS
   16            a spectral data record is plotted on a REGIS terminal
   17       }
   18
   19           TYPE
   20             t_spectral_data = ARRAY [1..1] OF spectral_data;
   21
   22           VAR
   23             temp_spec: t_spectral_data;
   24             ss: STRING [78];
   25
   26           BEGIN
   27             temp_spec := loophole(t_spectral_data, spec);
   28             strass(ss, s);
   29             plotrg_mspectra(ss, temp_spec, 1, 1);
   30           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]PLOTRG_SPECTRA.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM REGIS_TERMINAL;
    2        {$nolist}
    8        {$list}
    9
   10
   11         FUNCTION REGIS_TERMINAL (: boolean) ;
   12       { RESULTS
   13            a boolean value of TRUE if the process terminal has the REGIS characteristic for plotting
   14       }
   15
   16
   17           BEGIN
   18             REGIS_TERMINAL := test_term_char(0, tt2$m_regis);
   19           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]REGIS_TERMINAL.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM REPORT_MSPECTRA;
 {$nolist}
 {$list}


  PROCEDURE REPORT_MSPECTRA ((out_fn: PACKED ARRAY [olow..ohigh: integer] OF char; s: PACKED ARRAY [slow..shigh: integer]
                              VAR spec: ARRAY [low..high: integer] OF spectral_data; starting_spectra_array_index,
                              number_of_spectra, width, ext, line_length: integer; VAR error_code: integer)) ;
{
 WHERE
     out_fn = filename that report should be written to
     s = string to include at top of report
     spec = spectral data records
     starting_spectra_array_index = element of array to start displaying  (range is low..high)
     number_of_spectra = number of spectra to be displayed
     width = filed width for writing real numbers  (values < 2 are set to the default of 12)
     ext = number of digits to display past the decimal point (valid range is 0 to width-2, -1 signals scientific notatio
     line_length = line length of display device
     error_code = signals an error condition
 RESULTS
     a report is generated for the spectral data record(s)
}

    VAR
      spectra_per_line, i, spec_start, spec_stop, display_start, display_stop, num: integer;
      error: boolean;
      out: text;
      ss, out_filename: STRING [78];

    BEGIN
      IF width < 2 THEN
        width := 12;
      IF ext > width - 2 THEN
        ext := - 1;

      error_code := 0; { everything's ok to start with }

      IF NOT (starting_spectra_array_index IN [low..high]) THEN
        error_code := - 1; { invalid array index }

      IF error_code = 0 THEN
        IF number_of_spectra < 1 THEN
          error_code := - 2; { bad number parameter }

      IF error_code = 0 THEN
        IF NOT (starting_spectra_array_index + number_of_spectra - 1 IN [low..high]) THEN
          error_code := - 2; { invalid array index addressed }

      IF error_code <> 0 THEN
        BEGIN
        writec('REPORT_MSPECTRA - parameter error');
        return(2);
        END;

      IF error_code = 0 THEN
```

```
60              BEGIN
61              strass(ss, s);
62              strass(out_filename, out_fn);
63              spec_start := starting_spectra_array_index;
64              spec_stop := starting_spectra_array_index + number_of_spectra - 1;
65
66              IF number_of_spectra > 1 THEN
67                BEGIN
68                error := false;
69                FOR i := spec_start + 1 TO spec_stop DO
70                  BEGIN
71                  error := error OR (spec[i].start <> spec[spec_start].start);
72                  error := error OR (spec[i].stop <> spec[spec_start].stop);
73                  error := error OR (spec[i].inc <> spec[spec_start].inc);
74                  END;
75                IF error THEN
76                  BEGIN
77                  writec('MULTIPLE SPECTRA TO BE REPORTED HAVE MISMATCHED');
78                  writec('WAVELENGTH PARAMETERS AND WILL BE REPORTED INDIVIDUALLY');
79                  return(2);
80                  error_code := - 100; { spectra have different wavelength parameters, treat them individually }
81                  FOR i := spec_start TO spec_stop DO
82                    BEGIN
83                    report_spectra(out_filename, 'SPECTRA ARE BEING REPORTED INDIVIDUALLY  ' + ss, spec[i], width, ext, line_le
84                                   error_code);
85                    say_error_code('ERROR while reporting individual spectra', error_code, true);
86                    END;
87                  END;
88                END;
89              END;
90
91            IF error_code = 0 THEN
92              BEGIN
93              i := 0;
94              rewrite(out, out_filename, 'SPECTRA_REPORT.LIS', i);
95              IF i = - 1 THEN
96                error_code := - 201; { cannot open file for spectral report }
97              END;
98
99            IF error_code = 0 THEN
100             BEGIN
101             IF spec[spec_start].start = 0 THEN
102               BEGIN
103               IF number_of_spectra = 1 THEN
104                 writec('This array has not been loaded')
105               ELSE
106                 writec('These arrays have not been loaded');
107               return(2);
108               END
109             ELSE
110               BEGIN
111               writeln(out, ' ': (line_length - 40) DIV 2, 'S P E C T R A L   D A T A   R E P O R T');
112               IF ord(ss[0]) < line_length THEN
113                 writeln(out, ' ': (line_length - ord(ss[0])) DIV 2, ss)
```

```
114                 ELSE
115                   writeln(out, ss);
116                 writeln(out);
117                 FOR i := spec_start TO spec_stop DO
118                   BEGIN
119                   write(out, ' ': 5, 'Spectra', i: 3, ' Descrip : ');
120                   display_descrip(out, spec[i], line_length - 26 - 2, false, true, true);
121                   writeln(out);
122                   END;
123                 writeln(out);
124                 writeln(out, ' ': 5, 'Spectral data reported from ', spec[spec_start].start: 0, ' nm to ', spec[spec_start].sto
125                         ' nm every ', spec[spec_start].inc: 0, ' nm');
126
127                 spectra_per_line := (line_length - 2 - 7) DIV (width + 2);
128                 IF spectra_per_line < 1 THEN
129                   spectra_per_line := 1;
130
131                 display_stop := spec_start - 1;
132
133                 REPEAT { set up to report spectra }
134                   writeln(out);
135                   display_start := display_stop + 1;
136                   display_stop := display_start + spectra_per_line - 1;
137                   IF display_stop > spec_stop THEN
138                     display_stop := spec_stop;
139
140                   write(out, 'Spectra'); { display spectra numbers at the top of the first display screen }
141                   FOR num := display_start TO display_stop DO
142                     BEGIN
143                     write(out, ' ': 2);
144                     IF (width - 2) DIV 2 > 0 THEN
145                       write(out, ' ': (width - 2) DIV 2, num: 2, ' ': width - 2 - (width - 2) DIV 2)
146                     ELSE IF width = 3 THEN
147                       write(out, ' ', num: 2)
148                     ELSE
149                       write(out, num: 2);
150                     END;
151                   writeln(out);
152                   writeln(out);
153                   write(out, ' ': 7); { display some labels at the top of the first display screen }
154                   FOR num := display_start TO display_stop DO
155                     BEGIN
156                     write(out, ' ': 2);
157                     IF width < 10 THEN
158                       FOR i := 1 TO width DO
159                         write(out, spec[num].name[i])
160                     ELSE
161                       write(out, spec[num].name: 10, ' ': width - 10);
162                     END;
163                   writeln(out);
164                   writeln(out);
165                   FOR i := 1 TO (spec[spec_start].stop - spec[spec_start].start) DIV spec[spec_start].inc + 1 DO
166                     BEGIN
167                     write(out, (i - 1) * spec[spec_start].inc + spec[spec_start].start: 3, ' nm ');
```

```
168                     FOR num := display_start TO display_stop DO
169                       IF ext > - 1 THEN
170                         write(out, ' ': 2, spec[num].val[i]: width: ext)
171                       ELSE
172                         write(out, ' ': 2, spec[num].val[i]: width);
173                     writeln(out);
174                     END;
175                 UNTIL display_stop = spec_stop;
176                 END;
177               close(out);
178               END;
179           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]REPORT_MSPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM REPORT_SPECTRA;
2        ($nolist)
7        ($list)
8
9
10        PROCEDURE REPORT_SPECTRA ((out_fn: PACKED ARRAY [olow..ohigh: integer] OF char; s: PACKED ARRAY [slow..shigh: integer]
11                                   VAR spec: spectral_data; width, ext, line_length: integer; VAR error_code: integer)) ;
12      {
13       WHERE
14           out_fn = filename that report should be written to
15           s = string to include at top of report
16           spec = spectral data record
17           error_code = signals an error condition
18       RESULTS
19           a report is generated for the spectral data record
20      }
21
22          TYPE
23            t_spectral_data = ARRAY [1..1] OF spectral_data;
24
25          VAR
26            temp_spec: t_spectral_data;
27            ss: STRING [78];
28            out_filename: STRING [70];
29
30          BEGIN
31            temp_spec := loophole(t_spectral_data, spec);
32            strass(ss, s);
33            strass(out_filename, out_fn);
34            report_mspectra(out_filename, ss, temp_spec, 1, 1, width, ext, line_length, error_code);
35          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]REPORT_SPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM RETRIEVE_BIG_MSPECTRA;
    2        ($nolist)
    7        ($list)
    8
    9
   10         PROCEDURE RETRIEVE_BIG_MSPECTRA ((VAR spec: ARRAY [low..high: integer] OF big_spectral_data; VAR error_code: integer))
   11       {
   12        WHERE
   13            spec = spectral data record
   14            error_code = error code that is passed back to calling routine
   15        RESULTS
   16            one or more big spectral data records are loaded from a values in a data file
   17       }
   18
   19           VAR
   20             i, len: integer;
   21             line: STRING [78];
   22             inp: text;
   23
   24           BEGIN
   25             error_code := 0; { everything's ok to start with }
   26
   27             IF error_code = 0 THEN
   28               BEGIN
   29               len := 0;
   30               reset(inp, spec[low].filename, '.DAT', len); { open the file }
   31               IF ioerror(inp) THEN
   32                 error_code := - 4;
   33               END;
   34
   35             IF error_code = 0 THEN
   36               BEGIN
   37               noioerror(inp);
   38               readln(inp, spec[low].descrip); { read description of data }
   39               IF ioerror(inp) THEN
   40                 error_code := - 5;
   41               END;
   42
   43             IF error_code = 0 THEN
   44               BEGIN
   45               readln(inp, line); { read the line that should contain start, stop, inc, & number of spectra }
   46               IF ioerror(inp) THEN
   47                 error_code := - 6
   48               END;
   49
   50             IF error_code = 0 THEN
   51               BEGIN
   52               readln(inp, line); { read the line containing space delimited descriptions of the columns of data }
   53               IF ioerror(inp) THEN
   54                 error_code := - 10
   55               END;
   56
   57             i := 359;
   58             WHILE (error_code = 0) AND (i < 830) DO
   59               BEGIN
```

```
60              i := i + 1;
61              read(inp, len);
62              IF i <> len THEN
63                error_code := - 11;
64              FOR len := low TO high DO
65                BEGIN
66                read(inp, spec[len].val[i]);
67                IF ioerror(inp) THEN
68                  error_code := - 11;
69                END;
70              readln(inp);
71              IF ioerror(inp) THEN
72                error_code := - 11;
73              END;
74          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]RETRIEVE_BIG_MSPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM RETRIEVE_MSPECTRA;
 {$nolist}
 {$list}


  PROCEDURE RETRIEVE_MSPECTRA ((VAR spec: ARRAY [low..high: integer] OF spectral_data; starting_spectra_array_index,
                                starting_spectra_in_file_to_read, number_of_spectra: integer; VAR error_code: integer)) ;
{
 WHERE
     spec = spectral data record
     starting_spectra_array_index = element of array to start reading into  (range is low..high)
     starting_spectra_in_file_to_read = identifies which column of data in file to read (range is 1..what's there)
     number_of_spectra = number of spectra to be loaded into the array
     error_code = error code that is passed back to calling routine
 RESULTS
     one or more spectral data records are loaded from a values in a data file
}

    CONST
      max_reals_converted_per_line = 100;

    VAR
      number_of_spectra_in_file, number_of_data_lines, chr_ptr, index, getsub_code, rval_code, i, len, num, skip_before,
      skip_after: integer;
      inp: text;
      delim: char;
      temp: ARRAY [1..max_reals_converted_per_line] OF real; { temp holds real numbers converted from a line read from a
      line, sub_line: STRING [255];
      temp_name: sstring;
      error, eos: boolean;

    BEGIN
      IF spectra_debug THEN
        BEGIN
        writeln('starting_spectra_array_index=', starting_spectra_array_index);
        writeln('starting_spectra_in_file_to_read=', starting_spectra_in_file_to_read);
        writeln('number_of_spectra=', number_of_spectra);
        END;

      error_code := 0; { everything's ok to start with }

      IF NOT (starting_spectra_array_index IN [low..high]) THEN
        error_code := - 1; { invalid array index }

      IF error_code = 0 THEN
        IF number_of_spectra < 1 THEN
          error_code := - 2; { bad number parameter }

      IF error_code = 0 THEN
        IF (starting_spectra_in_file_to_read < 1) OR (starting_spectra_in_file_to_read > max_reals_converted_per_line) TH
          error_code := - 8; { bad starting spectra num in file}

      IF error_code = 0 THEN
        IF (starting_spectra_in_file_to_read + number_of_spectra - 1 > max_reals_converted_per_line) THEN
          error_code := - 2;
```

```
60            IF error_code = 0 THEN
61              BEGIN
62              IF spec[spec_start].start = 0 THEN
63                error_code := - 90 { spectra is empty }
64              ELSE
65                BEGIN
66                i := 0;
67                rewrite(out, spec[spec_start].filename, 'test.dat', i);
68                IF i <> - 1 THEN
69                  BEGIN
70                  writeln(out, spec[spec_start].descrip);
71                  writeln(out, spec[spec_start].start: 0, ' ', spec[spec_start].stop: 0, ' ', spec[spec_start].inc: 0, ' ',
72                          number_of_spectra: 0);
73
74                  FOR i := spec_start TO spec_stop DO
75                    write(out, spec[i].name, ' ');
76                  writeln(out);
77
78                  FOR wave := 1 TO (spec[spec_start].stop - spec[spec_start].start) DIV spec[spec_start].inc + 1 DO
79                    BEGIN
80                    FOR i := spec_start TO spec_stop DO
81                      write(out, spec[i].val[wave], ' ');
82                    writeln(out);
83                    END;
84
85                  close(out);
86                  END
87                ELSE
88                  error_code := - 91;
89                END;
90              END;
91          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAVE_MSPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM SAVE_MSPECTRA_AS_XYOATA;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SAVE_MSPECTRA_AS_XYOATA ((VAR spec: ARRAY [low..high: integer] OF spectral_data; starting_spectra_array_index
11                                            number_of_spectra: integer; VAR error_code: integer)) ;
12      {
13       WHERE
14           spec = spectral data record
15           starting_spectra_array_index = element of array to start saving
16           number_of_spectra = number of spectra to be saved
17           error_code = signals error
18       RESULTS
19           one or more spectral data records are written to a data file in X,Y1,Y2,Y3.... format
20      }
21
22          VAR
23            i, spec_start, spec_stop, wave: integer;
24            out: text;
25            error: boolean;
26
27          BEGIN
28            error_code := 0; { everything's ok to start with }
29
30            IF NOT (starting_spectra_array_index IN [low..high]) THEN
31              error_code := - 1; { invalid array index }
32
33            IF error_code = 0 THEN
34              IF number_of_spectra < 1 THEN
35                error_code := - 2; { bad number parameter }
36
37            IF error_code = 0 THEN
38              IF NOT (starting_spectra_array_index + number_of_spectra - 1 IN [low..high]) THEN
39                error_code := - 2; { invalid array index addressed }
40
41            IF error_code = 0 THEN
42              BEGIN
43              spec_start := starting_spectra_array_index;
44              spec_stop := starting_spectra_array_index + number_of_spectra - 1;
45
46              IF number_of_spectra > 1 THEN
47                BEGIN
48                error := false;
49                FOR i := spec_start + 1 TO spec_stop DO
50                  BEGIN
51                  error := error OR (spec[i].start <> spec[spec_start].start);
52                  error := error OR (spec[i].stop <> spec[spec_start].stop);
53                  error := error OR (spec[i].inc <> spec[spec_start].inc);
54                  END;
55                IF error THEN
56                  error_code := - 100; { spectra have different wavelength parameters, treat them individually }
57                END;
58              END;
59
```

```
60            IF error_code = 0 THEN
61              BEGIN
62              IF spec[spec_start].start = 0 THEN
63                error_code := - 90
64              ELSE
65                BEGIN
66                i := 0;
67                rewrite(out, spec[spec_start].filename, 'test.dat', i);
68                IF i <> - 1 THEN
69                  BEGIN
70
71                  FOR wave := 1 TO (spec[spec_start].stop - spec[spec_start].start) DIV spec[spec_start].inc + 1 DO
72                    BEGIN
73                    write(out, spec[spec_start].start + (wave - 1) * spec[spec_start].inc, ' ');
74                    FOR i := spec_start TO spec_stop DO
75                      write(out, spec[i].val[wave], ' ');
76                    writeln(out);
77                    END;
78
79                  close(out);
80                  END
81                ELSE
82                  error_code := - 91;
83                END;
84              END;
85          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAVE_MSPECTRA_AS_XYDATA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM SAVE_SPECTRA;
    2        ($nolist)
    7        ($list)
    8
    9
   10         PROCEDURE SAVE_SPECTRA ((VAR spec: spectral_data; VAR error_code: integer)) ;
   11       {
   12        WHERE
   13            spec = spectral data record
   14            error_code = signals error
   15        RESULTS
   16            one spectral data record is written to a data file
   17       }
   18
   19           TYPE
   20             t_spectral_data = ARRAY [1..1] OF spectral_data;
   21
   22           VAR
   23             temp_spec: t_spectral_data;
   24
   25           BEGIN
   26             temp_spec := loophole(t_spectral_data, spec);
   27             save_mspectra(temp_spec, 1, 1, error_code);
   28           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAVE_SPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM SAVE_SPECTRA_AS_XYDATA;
    2        ($nolist)
    7        ($list)
    8
    9
   10          PROCEDURE SAVE_SPECTRA_AS_XYDATA ((VAR spec: spectral_data; VAR error_code: integer)) ;
   11       {
   12        WHERE
   13            spec = spectral data record
   14            error_code = signals an error
   15        RESULTS
   16            one spectral data record is written to a data file in X,Y format
   17       }
   18
   19           TYPE
   20             t_spectral_data = ARRAY [1..1] OF spectral_data;
   21
   22           VAR
   23             temp_spec: t_spectral_data;
   24
   25           BEGIN
   26             temp_spec := loophole(t_spectral_data, spec);
   27             save_mspectra_as_xydata(temp_spec, 1, 1, error_code);
   28           END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAVE_SPECTRA_AS_XYDATA.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
1       PROGRAM SAY_ERROR;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SAY_ERROR ((s: PACKED ARRAY [low..high: integer] OF char; error, pause: boolean)} ;
11      {
12       WHERE
13           s = message to be displayed if there is an error
14           error = signals an error
15           pause = signals that the user should be prompted to press RETURN to continue
16       RESULTS
17           the user is notified upon ERROR being TRUE
18      }
19
20          VAR
21            ss: STRING [70];
22
23          BEGIN
24            IF error THEN
25              BEGIN
26              strass(ss, s);
27              writec(ss);
28              write(chr(7));
29              IF pause THEN
30                return(1);
31              END;
32          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAY_ERROR.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM SAY_ERROR_CODE;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE SAY_ERROR_CODE ((s: PACKED ARRAY [low..high: integer] OF char; error_code: integer; pause: boolean)) ;
   11       (
   12        WHERE
   13            s =  message to be displayed if there is an error
   14            error_code =  integer returned by a spectra routine to signal errors
   15            pause =  signals that the user should be prompted to press RETURN to continue
   16        RESULTS
   17            the user is notified upon an error signalled by error_code having a non-zero value
   18       )
   19
   20           VAR
   21             ss: STRING [70];
   22
   23           BEGIN
   24             IF error_code <> 0 THEN
   25               BEGIN
   26               strass(ss, s);
   27               writec(ss);
   28               writeln('Error Code = ', error_code: 0);
   29               write(chr(7));
   30               IF pause THEN
   31                 return(1);
   32               END;
   33           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SAY_ERROR_CODE.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM SELECT_SHOW_COORDINATES;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SELECT_SHOW_COORDINATES ((VAR color, difference: show_coordinate_data; degree: integer)) ;
11      {
12       WHERE
13           color = array that selects color coordinates to be displayed or reported
14           difference = array that selects color coordinate differences to be displayed or reported
15       RESULTS
16           values are returned in the above two arrays
17      }
18
19          CONST
20            margin = 10;
21
22          VAR
23            coord: coordinate_data;
24            sh: show_coordinate_type;
25            lname: lstring;
26            name: string5;
27            lname_descriptor, name_descriptor: descriptor_block;
28            i, j, k, l: integer;
29            q, q1: char;
30
31
32          FUNCTION iindex(typ: show_coordinate_type): integer;
33
34            BEGIN
35              iindex := ord(typ) - ord(tristimulus) + 1;
36            END;
37
38
39          FUNCTION sindex(typ: integer): show_coordinate_type;
40
41            BEGIN
42              typ := typ + ord(tristimulus) - 1;
43              sindex := loophole(show_coordinate_type, typ);
44            END;
45
46
47          PROCEDURE yes_no(i: integer);
48
49            BEGIN
50              IF i < 1 THEN
51                writeln('  NO')
52              ELSE
53                writeln(' YES');
54            END;
55
56
57          PROCEDURE toggle(VAR i: integer);
58
59            BEGIN
```

```
60              IF i < 1 THEN
61                i := 1
62              ELSE
63                i := 0;
64            END;
65
66
67          BEGIN
68            lname_descriptor := make_descriptor_block(lname, 40);
69            name_descriptor := make_descriptor_block(name, 5);
70            q := 'N';
71            REPEAT
72              prompt(4);
73              writec('DISPLAY COLOR COORDINATES MENU');
74              jump(2);
75              write(' ': margin, ' 0:  ', 'TURN THE DISPLAY OF ALL REGULAR COORDINATES ON OR OFF');
76              writeln;
77              write(' ': margin, ' 1:  ', 'Tristimulus Values:  X, Y, & Z          ');
78              yes_no(color[tristimulus]);
79              write(' ': margin, ' 2:  ', 'Chromaticity Coordinates:  x, y, & z    ');
80              yes_no(color[Chromaticity_1]);
81              write(' ': margin, ' 3:  ', 'Chromaticity Coordinates:  u, v, & v''   ');
82              yes_no(color[Chromaticity_2]);
83              write(' ': margin, ' 4:  ', 'Dominant Wavelength & Excitation Purity ');
84              yes_no(color[Domwave_purity]);
85              write(' ': margin, ' 5:  ', 'CIELAB coordinates  L*, a*, & b*        ');
86              yes_no(color[CIELAB_1]);
87              write(' ': margin, ' 6:  ', 'CIELAB coordinates  c* & h              ');
88              yes_no(color[CIELAB_2]);
89              write(' ': margin, ' 7:  ', 'CIELUV coordinates  L*, u*, & v*        ');
90              yes_no(color[CIELUV_1]);
91              write(' ': margin, ' 8:  ', 'CIELUV coordinates  c*, s, & h          ');
92              yes_no(color[CIELUV_2]);
93              jump(1);
94              readch('change the display of color coordinates ? (Y,N)', q, use_default, null_ok);
95              IF q = 'Y' THEN
96                BEGIN
97                getin('Select number from above', i, no_default, 0, iindex(CIELUV_2));
98                IF i = 0 THEN
99                  BEGIN
100                 jump(1);
101                 q1 := 'Y';
102                 readch('turn the display of all regular coordinates ON ? (Y,N)', q1, use_default, null_ok);
103                 IF q1 <> 'N' THEN
104                   FOR i := 1 TO iindex(CIELUV_2) DO
105                     color[sindex(i)] := 1
106                 ELSE
107                   FOR i := 1 TO iindex(CIELUV_2) DO
108                     color[sindex(i)] := 0;
109                 END
110               ELSE
111                 toggle(color[sindex(i)]);
112               END;
113           UNTIL q <> 'Y';
```

```
114             q := 'N';
115             REPEAT
116               prompt(4);
117               writec('DISPLAY COLOR DIFFERENCES MENU');
118               jump(2);
119               write(' ': margin, ' 0:  ', 'TURN THE DISPLAY OF ALL REGULAR DIFFERENCES ON OR OFF');
120               writeln;
121               write(' ': margin, ' 1:  ', 'Tristimulus Values:  dX, dY, & dZ       ');
122               yes_no(difference[tristimulus]);
123               write(' ': margin, ' 2:  ', 'Chromaticity Coordinates:  dx, dy, & dz ');
124               yes_no(difference[Chromaticity_1]);
125               write(' ': margin, ' 3:  ', 'Chromaticity Coordinates: du, dv, & dv'' ');
126               yes_no(difference[Chromaticity_2]);
127               write(' ': margin, ' 4:  ', 'Dominant Wavelength & Excitation Purity ');
128               yes_no(difference[Domwave_purity]);
129               write(' ': margin, ' 5:  ', 'CIELAB coordinates  dL*, da*, db*, & dE ');
130               yes_no(difference[CIELAB_1]);
131               write(' ': margin, ' 6:  ', 'CIELAB coordinates  dc* & dh            ');
132               yes_no(difference[CIELAB_2]);
133               write(' ': margin, ' 7:  ', 'CIELUV coordinates  dL*, du*, dv*, & dE ');
134               yes_no(difference[CIELUV_1]);
135               write(' ': margin, ' 8:  ', 'CIELUV coordinates  dc*, ds, & dh       ');
136               yes_no(difference[CIELUV_2]);
137               jump(1);
138               readch('change the display of color differences ? (Y,N)', q, use_default, null_ok);
139               IF q = 'Y' THEN
140                 BEGIN
141                 getin('Select number from above', i, no_default, 0, iindex(CIELUV_2));
142                 IF i = 0 THEN
143                   BEGIN
144                   jump(1);
145                   q1 := 'Y';
146                   readch('turn the display of all regular coordinates ON ? (Y,N)', q1, use_default, null_ok);
147                   IF q1 <> 'N' THEN
148                     FOR i := 1 TO iindex(CIELUV_2) DO
149                       difference[sindex(i)] := T
150                   ELSE
151                     FOR i := 1 TO iindex(CIELUV_2) DO
152                       difference[sindex(i)] := 0;
153                   END
154                 ELSE
155                   toggle(difference[sindex(i)]);
156                 END;
157             UNTIL q <> 'Y';
158             q := 'N';
159             REPEAT
160               prompt(4);
161               writec('DISPLAY SPECIAL COLOR DIFFERENCES MENU');
162               jump(2);
163               write(' ': margin, ' 0:  ', 'TURN OFF THE DISPLAY OF ALL REGULAR DIFFERENCES');
164               writeln;
165               FOR sh := special_1 TO special_5 DO
166                 BEGIN
167                 write(' ': margin, iindex(sh) - iindex(special_1) + 1: 2, ':  ');
```

```
168                 IF difference[sh] > 0 THEN
169                   BEGIN
170                   j := - difference[sh];
171                   special(coord, coord, coord, lname_descriptor, name_descriptor, degree, j);
172                   IF j <> 0 THEN
173                     difference[sh] := 0;
174                   END;
175                 IF difference[sh] > 0 THEN
176                   BEGIN
177                   write(lname);
178                   yes_no(difference[sh]);
179                   END
180                 ELSE
181                   writeln('     special calculation not selected');
182                 END;
183               jump(1);
184               readch('change the display of special color differences ? (Y,N)', q, use_default, null_ok);
185               IF q = 'Y' THEN
186                 BEGIN
187                 jump(1);
188                 getin('Select number from above', i, no_default, 0, 5);
189                 IF i = 0 THEN
190                   FOR sh := special_1 TO special_5 DO
191                     difference[sh] := 0
192                 ELSE
193                   BEGIN
194                   IF difference[sindex(i + iindex(CIELUV_2))] > 0 THEN
195                     difference[sindex(i + iindex(CIELUV_2))] := 0
196                   ELSE
197                     BEGIN
198                     prompt(4);
199                     writec('SPECIAL COLOR DIFFERENCE CALCULATIONS AVAILABLE');
200                     jump(1);
201                     FOR k := 1 TO 99 DO
202                       BEGIN
203                       j := - k;
204                       special(coord, coord, coord, lname_descriptor, name_descriptor, degree, j);
205                       IF j = 0 THEN
206                         writeln(' ': margin, k: 2, ':  ', lname);
207                       END;
208                     REPEAT
209                       l := 0;
210                       writeln;
211                       readin('select one of the above or 0 to not select', l, use_default);
212                       IF l <> 0 THEN
213                         BEGIN
214                         j := - l;
215                         special(coord, coord, coord, lname_descriptor, name_descriptor, degree, j);
216                         IF j <> 0 THEN
217                           BEGIN
218                           writeln;
219                           writeln(' * * * Invalid special calculation number. Please re-enter');
220                           writeln;
221                           END
```

```
222                         ELSE
223                           BEGIN
224                           difference[sindex(i + iindex(CIELUV_2))] := l;
225                           l := 0;
226                           END;
227                         END;
228                     UNTIL l = 0;
229                     END;
230                   END;
231                 END;
232             UNTIL q <> 'Y';
233           END;
```

Invocation line:
SOURCE:SELECT_SHOW_COORDINATES.PASF/LIST/DOUBLE/NOMAIN

*** No lines with errors detected ***

```
1       PROGRAM SHOW_CALC;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SHOW_CALC ((VAR out: text; VAR coordinates: coordinate_data; to_show: coordinate_type_set; options:
11                              show_calc_options_set; space_before, space_between, space_after, field_width, extension: integer));
12      {
13       WHERE
14           out = file for output
15           coordinates = coordinate data to display
16           to_show:
17           options = options to use when displaying value
18           space_before = number of spaces to write before a value or label
19           space_between = number of spaces to write between a value and label
20           space_after = number of spaces to write after a value
21           field_width = width of space to write value in
22           extension = number of digits to display in each value past the decimal place
23       RESULTS
24           a value is displayed using the options selected
25      }
26
27      {
28           show_calc_options = ( show_calc_label,
29                                 show_calc_colon,
30                                 show_calc_delta,
31                                 show_calc_value,
32                                 show_calc_nl,
33                                 show_calc_nl_when_done);
34      }
35
36          VAR
37            coord: coordinate_type;
38            i: integer;
39
40
41          PROCEDURE write_space(VAR out: text;
42                                width: integer);
43
44            BEGIN
45              IF width > 0 THEN
46                write(out, ' ': width);
47            END;
48
49
50          BEGIN
51            FOR coord := succ(first) TO pred(last) DO
52
53              IF coord IN to_show THEN
54                BEGIN
55                write_space(out, space_before);
56
57                IF show_calc_delta IN options THEN
58                  write(out, 'd')
59                ELSE
```

```
60                  write(out, ' ');
61
62                IF show_calc_label IN options THEN
63                  write(out, coordinate_names[coord]);
64
65                IF show_calc_colon IN options THEN
66                  write(out, ':');
67
68                write_space(out, space_between);
69
70                IF show_calc_value IN options THEN
71                  BEGIN
72                  IF NOT calc_color_bad_value(coordinates[coord]) THEN
73                    write(out, coordinates[coord]: field_width: extension)
74                  ELSE
75                    FOR i := 1 TO field_width DO
76                      write(out, '*');
77                  END;
78
79                write_space(out, space_after);
80
81                IF show_calc_nl IN options THEN
82                  writeln(out);
83
84                END;
85
86              IF show_calc_nl_when_done IN options THEN
87                writeln(out);
88            END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SHOW_CALC.PASF;3/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM SHOW_DESCRIPS;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE SHOW_DESCRIPS ((s: PACKED ARRAY [lows..highs: integer] OF char; VAR spec: ARRAY [low..high: integer] OF spect
   11          ;
   12       (
   13        WHERE
   14            s = string to display
   15            spec = array of spectra_data to display descriptions of
   16        RESULTS
   17            descriptions (short & long) are displayed on the users' terminal
   18       )
   19
   20           VAR
   21             i: integer;
   22             ss: STRING [78];
   23
   24           BEGIN
   25             strass(ss, s);
   26             prompt(1);
   27             writec(ss);
   28             FOR i := low TO high DO
   29               BEGIN
   30               write('No. ', i: 2, ' ');
   31               display_descrip(output, spec[i], 71, true, true, true);
   32               writeln;
   33               END;
   34             gotoxy(0, high - low + 1 + 3);
   35           END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SHOW_DESCRIPS.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
1       PROGRAM SHOW_DIFFERENCE_CALC;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SHOW_DIFFERENCE_CALC ((VAR out: text; VAR sample1, sample2: spectral_data; color: show_coordinate_data; degre
11                                         user_viewing: boolean)) ;
12      {
13       WHERE
14           out = text file
15           sample 1 = STANDARD spectra for difference calculations
16           sample 2 = SAMPLE spectra for difference calculations
17           color = selection of differences to show
18           degree = 2 for two degree observer, 10 for ten degree observer
19           user_viewing = use RETURN(1) to pause just prior to overfilling the screen
20       RESULTS
21           differences are written to file variable OUT
22      )
23
24          VAR
25            i, lines: integer;
26            lname: lstring;
27            name: string5;
28            lname_descriptor, name_descriptor: descriptor_block;
29            diff: coordinate_data;
30            sh: show_coordinate_type;
31            do_special: boolean;
32
33
34          BEGIN
35            lname_descriptor := make_descriptor_block(lname, 40);
36            name_descriptor := make_descriptor_block(name, 5);
37            IF user_viewing THEN
38              BEGIN
39              cursor(home, 1);
40              erase(eoscreen);
41              END;
42            write(out, 'STANDARD    ');
43            display_descrip(out, sample1, 66, true, true, true);
44            writeln(out);
45            write(out, 'DIFFERENCE  ');
46            display_descrip(out, sample2, 66, true, true, true);
47            writeln(out);
48            { calculate regular differences }
49            diff := calc_difference(sample1.coordinates, sample2.coordinates);
50            IF color[tristimulus] > 0 THEN
51              BEGIN
52              writeln(out);
53              writeln(out, 'Tristimulus Value Differences :');
54              write(out, ' ': 10);
55              show_calc(out, diff, [t_x..t_z],
56                        [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
57              END;
58            IF (color[chromaticity_1] > 0) OR (color[chromaticity_2] > 0) THEN
59              BEGIN
```

```
60              writeln(out);
61              writeln(out, 'Chromaticity Coordinate Differences :');
62              END;
63            IF color[chromaticity_1] > 0 THEN
64              BEGIN
65              write(out, ' ': 10);
66              show_calc(out, diff, [c_x..c_z],
67                      [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
68              END;
69            IF color[chromaticity_2] > 0 THEN
70              BEGIN
71              write(out, ' ': 10);
72              show_calc(out, diff, [u, v, v_prime],
73                      [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
74              END;
75            IF color[domwave_purity] > 0 THEN
76              BEGIN
77              writeln(out);
78              writeln(out, 'Dominant Wavelength and Excitation Purity Oifferences :');
79              write(out, ' ': 10);
80              show_calc(out, diff, [dominant_wavelength], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value],
81                      7, 2);
82              show_calc(out, diff, [excitation_purity],
83                      [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
84              END;
85            IF (color[CIELAB_1] > 0) OR (color[CIELAB_2] > 0) THEN
86              BEGIN
87              writeln(out);
88              writeln(out, 'CIELAB coordinate differences :');
89              END;
90            IF color[CIELAB_1] > 0 THEN
91              BEGIN
92              write(out, ' ': 2);
93              show_calc(out, diff, [l_star..b_star], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value], 0, 1
94              show_calc(out, diff, [E_ab], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_wh
95                      0, 1, 5, 7, 2);
96              END;
97            IF color[CIELAB_2] > 0 THEN
98              BEGIN
99              write(out, ' ': 2);
100             show_calc(out, diff, [c_star_ab..h_ab],
101                     [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
102             END;
103           IF (color[CIELUV_1] > 0) OR (color[CIELUV_2] > 0) THEN
104             BEGIN
105             writeln(out);
106             writeln(out, 'CIELUV coordinate differences :');
107             END;
108           IF color[CIELUV_1] > 0 THEN
109             BEGIN
110             write(out, ' ': 2);
111             show_calc(out, diff, [l_star], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value], 0, 1, 5, 7,
112             show_calc(out, diff, [u_star..v_star], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value], 0, 1
113             show_calc(out, diff, [E_uv], [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_wh
```

```
114                         0, 1, 5, 7, 2);
115               END;
116             IF color[CIELUV_2] > 0 THEN
117               BEGIN
118               write(out, ' ': 2);
119               show_calc(out, diff, [c_star_uv..h_uv],
120                         [show_calc_label, show_calc_delta, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0, 1, 5,
121               END;
122
123             IF user_viewing THEN
124               BEGIN
125               { now how many (if any) special color differences there are }
126               lines := 0;
127               FOR sh := special_1 TO special_5 DO
128                 BEGIN
129                 IF color[sh] > 0 THEN
130                   BEGIN
131                   i := - color[sh];
132                   special(sample1.coordinates, sample2.coordinates, diff, lname_descriptor, name_descriptor, degree, i);
133                   IF i = 0 THEN
134                     lines := lines + 1;
135                   END;
136                 END;
137               { the following counting of lines displayed relies on color containing values of 0 for not displaying and 1 for d
138               i := color[tristimulus] * 3;
139               IF (color[chromaticity_1] > 0) OR (color[chromaticity_2] > 0) THEN
140                 i := i + 2;
141               i := i + color[chromaticity_1];
142               i := i + color[chromaticity_2];
143               i := i + color[domwave_purity] * 3;
144               IF (color[CIELAB_1] > 0) OR (color[CIELAB_2] > 0) THEN
145                 i := i + 2;
146               i := i + color[CIELAB_1];
147               i := i + color[CIELAB_2];
148               IF (color[CIELUV_1] > 0) OR (color[CIELUV_2] > 0) THEN
149                 i := i + 2;
150               i := i + color[CIELUV_1];
151               i := i + color[CIELUV_2];
152               IF 2 + i + lines + 1 + 2 > 23 THEN
153                 BEGIN
154                 writeln;
155                 return(1);
156                 prompt(1);
157                 write(out, 'STANDARD    ');
158                 display_descrip(out, sample1, 66, true, true, true);
159                 writeln(out);
160                 write(out, 'DIFFERENCE  ');
161                 display_descrip(out, sample2, 66, true, true, true);
162                 writeln(out);
163                 jump(4);
164                 END;
165               END;
166             writeln(out);
167             FOR sh := special_1 TO special_5 DO
```

```
168             BEGIN
169             IF color[sh] > 0 THEN
170               BEGIN
171               i := color[sh];
172               special(sample1.coordinates, sample2.coordinates, diff, lname_descriptor, name_descriptor, degree, i);
173               IF i = 0 THEN
174                 writeln(out, lname, ':  ', name, ' :  ', diff[last]: 10: 5);
175               END;
176             END;
177         END;
```

Invocation line:
SOURCE:SHOW_DIFFERENCE_CALC.PASF/LIST/DOUBLE/NOMAIN

*** No lines with errors detected ***

```
1       PROGRAM SHOW_SAMPLE_CALC;
2        {$nolist}
7        {$list}
8
9
10        PROCEDURE SHOW_SAMPLE_CALC ((VAR out: text; sample: coordinate_data; color: show_coordinate_data)) ;
11      {
12       WHERE
13           out = text file
14           sample = coordinates to be displayed
15           color = selection of coordinates to show
16       RESULTS
17           sample color calculations are displayed on the terminal
18      }
19
20
21          BEGIN
22            IF color[tristimulus] > 0 THEN
23              BEGIN
24              writeln(out);
25              writeln(out, 'Tristimulus Values :');
26              write(out, ' ': 10);
27              show_calc(out, sample, [t_x..t_z], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0
28                        2);
29              END;
30            IF (color[chromaticity_1] > 0) OR (color[chromaticity_2] > 0) THEN
31              BEGIN
32              writeln(out);
33              writeln(out, 'Chromaticity Coordinates :');
34              END;
35            IF color[chromaticity_1] > 0 THEN
36              BEGIN
37              write(out, ' ': 10);
38              show_calc(out, sample, [c_x..c_z], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_done], 0
39                        4);
40              END;
41            IF color[chromaticity_2] > 0 THEN
42              BEGIN
43              write(out, ' ': 10);
44              show_calc(out, sample, [u, v, v_prime], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_don
45                        5, 7, 4);
46              END;
47            IF color[domwave_purity] > 0 THEN
48              BEGIN
49              writeln(out);
50              writeln(out, 'Dominant Wavelength and Excitation Purity :');
51              write(out, ' ': 10);
52              show_calc(out, sample, [dominant_wavelength], [show_calc_label, show_calc_colon, show_calc_value], 0, 1, 5, 7, 2)
53              show_calc(out, sample, [excitation_purity], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when
54                        1, 5, 7, 4);
55              END;
56            IF (color[CIELAB_1] > 0) OR (color[CIELAB_2] > 0) THEN
57              BEGIN
58              writeln(out);
59              writeln(out, 'CIELAB coordinates :');
```

```
60              END;
61            IF color[CIELAB_1] > 0 THEN
62              BEGIN
63              write(out, ' ': 10);
64              show_calc(out, sample, [l_star..b_star], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_do
65                      5, 7, 2);
66              END;
67            IF color[CIELAB_2] > 0 THEN
68              BEGIN
69              write(out, ' ': 10);
70              show_calc(out, sample, [c_star_ab..h_ab], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_d
71                      5, 7, 2);
72              END;
73            IF (color[CIELUV_1] > 0) OR (color[CIELUV_2] > 0) THEN
74              BEGIN
75              writeln(out);
76              writeln(out, 'CIELUV coordinates :');
77              END;
78            IF color[CIELUV_1] > 0 THEN
79              BEGIN
80              write(out, ' ': 10);
81              show_calc(out, sample, [l_star], [show_calc_label, show_calc_colon, show_calc_value], 0, 1, 5, 7, 2);
82              show_calc(out, sample, [u_star..v_star], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_do
83                      5, 7, 2);
84              END;
85            IF color[CIELUV_2] > 0 THEN
86              BEGIN
87              write(out, ' ': 10);
88              show_calc(out, sample, [c_star_uv..h_uv], [show_calc_label, show_calc_colon, show_calc_value, show_calc_nl_when_d
89                      5, 7, 2);
90              END;
91          END;

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SHOW_SAMPLE_CALC.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
0001          subroutine special(coord1,coord2,coord3,lname,sname,degree,number)
0002
0003          ! SPECIAL is a FORTRAN subroutine called by the COLOR program to allow special color difference calculations
0004          ! to be calculated and displayed by the color difference menu option in the COLOR program samples menu
0005          !
0006          ! Written by R. Mitchell Miller
0007
0008
0009          real*8   coord1(26),        ! array of color coordinates for sample 1
0010      -           coord2(26),        ! array of color coordinates for sample 2
0011      -           coord3(26)         ! array of color differences calculated
0012          character*40 lname          ! 40 character long description of special calculation
0013          character*5  sname          ! 5 character short description of special calculation
0014          integer*4    degree,        ! value of 2 indicates two degree observer, 10 indicates ten degree observer
0015      -                number         ! number of special calculation to use ( value > 0 indicates calculation to be performed
0016                                       !                                       and lname & sname returned )
0017                                       !                                     ( value < 0 indicates that no calculations are
0018                                       !                                       requested, but lname & sname should be returned
0019                                       !                                       for special calculation number abs(number)
0020          !
0021          ! RETURNS
0022          !     coord3[26] contains the result (or -999 if calculation was not possible)
0023          !     number = 0 for successful completion
0024          !              1 for invalid special calculation number
0025          !
0026          ! NOTES:
0027          !     The following lists the contents of the color coordinate arrays
0028          !
0029          !
0030          !               Index No.  Name
0031          !
0032          !                   1      UNUSED
0033          !                   2      t_x,
0034          !                   3      t_y,
0035          !                   4      t_z,
0036          !                   5      c_x,
0037          !                   6      c_y,
0038          !                   7      c_z,
0039          !                   8      dominant_wavelength,
0040          !                   9      excitation_purity,
0041          !                  10      u,
0042          !                  11      v,
0043          !                  12      u_prime,
0044          !                  13      v_prime,
0045          !                  14      l_star,
0046          !                  15      a_star,
0047          !                  16      b_star,
0048          !                  17      C_star_ab,
0049          !                  18      h_ab,
0050          !                  19      E_ab,
0051          !                  20      u_star,
0052          !                  21      v_star,
0053          !                  22      C_star_uv,
0054          !                  23      s_uv,
0055          !                  24      h_uv,
0056          !                  25      E_uv,
0057          !                  26      SPECIAL CALCULATION
```

```
0058
0059
0060            GOTO (01,02),ABS(NUMBER)
0061            number = -1
0062            GOTO 999
0063
0064      01    sname = 'dE_ab'
0065            lname = 'CIELAB delta E                          '
0066            if (number .gt. 0) then
0067               coord3(26) = ( (coord1(14)-coord2(14))**2.
0068           -                + (coord1(15)-coord2(15))**2.
0069           -                + (coord1(16)-coord2(16))**2.)**0.5
0070            endif
0071            number = 0
0072            GOTO 999
0073
0074      02    if (degree .eq. 10) then        ! an example of a calculation only for the 10 degree observer
0075              sname = 'dE_uv'
0076              lname = 'CIELUV delta E                          '
0077              if (number .gt. 0) then
0078                   coord3(26) = ( (coord1(14)-coord2(14))**2.
0079           -                    + (coord1(20)-coord2(20))**2.
0080           -                    + (coord1(21)-coord2(21))**2.)**0.5
0081              endif
0082              number = 0
0083            else
0084              number = -1 !  if degree indicates an incorrect observer
0085            endif
0086            GOTO 999
0087
0088
0089
0090
0091
0092
0093
0094      999   return
0095            end
```

PROGRAM SECTIONS

	Name	Bytes	Attributes
0	$CODE	261	PIC CON REL LCL SHR EXE RO NOWRT LONG
1	$PDATA	90	PIC CON REL LCL SHR NOEXE RO NOWRT LONG
2	$LOCAL	76	PIC CON REL LCL NOSHR NOEXE RO WRT LONG
	Total Space Allocated	427	

ENTRY POINTS

Address	Type	Name
0-00000000		SPECIAL

VARIABLES

Address	Type	Name	Address	Type	Name	Address	Type	Name	Address	Type	Name
AP-00000018@	I*4	DEGREE	AP-00000010@	CHAR	LNAME	AP-0000001C@	I*4	NUMBER	AP-00000014@	CHAR	SNAME

ARRAYS

Address	Type	Name	Bytes	Dimensions
AP-00000004@	R*8	COOR01	208	(26)
AP-00000008@	R*8	COOR02	208	(26)
AP-0000000C@	R*8	COOR03	208	(26)

LABELS

Address	Label	Address	Label	Address	Label
0-00000060	1	0-000000AC	2	0-00000104	999

COMMAND QUALIFIERS

```
FOR/LIST SPECIAL

/CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
/DEBUG=(NOSYMBOLS,TRACEBACK)
/STANDARD=(NOSYNTAX,NOSOURCE_FORM)
/SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP,NODICTIONARY,SINGLE)
/WARNINGS=(GENERAL,NODECLARATIONS,NOULTRIX)
/CONTINUATIONS=19  /NOCROSS_REFERENCE  /NOD_LINES  /NOEXTEND_SOURCE  /F77
/NOG_FLOATING  /I4  /NOMACHINE_CODE  /OPTIMIZE  /NOANALYSIS
```

COMPILATION STATISTICS

Run Time:	1.70 seconds
Elapsed Time:	3.96 seconds
Page Faults:	229
Dynamic Memory:	322 pages

```
PROGRAM SPECTRAL_SUMMATION;
 {$nolist}
 {$list}


  FUNCTION SPECTRAL_SUMMATION ((spec: spectral_data; starting_wavelength, ending_wavelength: integer; VAR error: boolean)

{
 WHERE
     spec                = spectral data record
     starting_wavelength = starting wavelength of summation
     ending_wavelength   = ending wavelength of summation
     error               = signals error in parameters
 RESULTS
     the summation of the spectral data record is returned
}

    VAR
      i: integer;
      temp: real;

    BEGIN
      temp := 0;
      error := false;

      IF spec.start = 0 THEN
        BEGIN
        error := true;
        writec('%SUMMATION-WARNING !!! data not in memory to sum');
        END;

      IF starting_wavelength < spec.start THEN
        BEGIN
        error := true;
        writec('%SUMMATION-WARNING !!! starting_wavelength < spec.start ');
        END;

      IF ending_wavelength > spec.stop THEN
        BEGIN
        error := true;
        writec('%SUMMATION-WARNING !!! ending_wavelength > spec.stop');
        END;

  { *** add a check for the starting and ending wavelengths matching the
       data }

      IF spectra_debug OR error THEN
        BEGIN
        writeln('%SUMMATION-INFO START=', spec.start: 0, ' END=', spec.stop: 0, ' INC=', spec.inc: 0);
        writeln('%SUMMATION-INFO STARTING WAVELENGTH TO SUM ', starting_wavelength: 0, 'nm');
        writeln('%SUMMATION-INFO ENDING WAVELENGTH TO SUM ', ending_wavelength: 0, 'nm');
        IF spectra_debug THEN
          spectra_debug_pause_execution
        ELSE
          return(2);
```

```
60              END;
61
62            IF NOT error THEN
63              BEGIN
64              i := starting_wavelength;
65              temp := 0;
66              REPEAT
67                temp := temp + spec.val[(i - spec.start) DIV spec.inc + 1];
68                i := i + spec.inc;
69              UNTIL i > ending_wavelength;
70              END;
71
72            SPECTRAL_SUMMATION := temp;
73          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SPECTRAL_SUMMATION.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
Pascal-2 VAX/VMS 2.2B    6-Aug-89   7:39 PM    Site #1-1    Page 1-1
Oregon Software, Inc.  6915 SW Macadam Ave.  Suite # 200  Portland  OR  97219  USA
USER1:[RMM.COLOR.SPECTRA.SOURCE]SPECTRA_DEBUG_PAUSE_EXECUTION.PASF;2

    1       PROGRAM SPECTRA_DEBUG_PAUSE_EXECUTION;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE SPECTRA_DEBUG_PAUSE_EXECUTION;
   11       {
   12        RESULTS
   13            a vale of TRUE indicates program execution should be stopped until a return is pressed after encountered an error
   14       }
   15
   16
   17           BEGIN
   18             IF spectra_debug_pause THEN
   19               return(1);
   20           END;


Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SPECTRA_DEBUG_PAUSE_EXECUTION.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM SPECTRA_YMAX;
 {$nolist}
 {$list}


  FUNCTION SPECTRA_YMAX ((spec: spectral_data): real) ;
{
 WHERE
     spec = spectral data record
 RESULTS
     the maximum value in the spectral data record is returned
)

    VAR
      i: integer;
      temp: real;
      error: boolean;

    BEGIN
      error := false;

      IF spec.start = 0 THEN
        BEGIN
        error := true;
        writec('%SPECTRA_MAX-WARNING !!! data not in memory to sum');
        END;

      IF NOT error THEN
        BEGIN
        i := spec.start + spec.inc;
        temp := spec.val[1];
        REPEAT
          IF spec.val[(i - spec.start) DIV spec.inc + 1] > temp THEN
            temp := spec.val[(i - spec.start) DIV spec.inc + 1];
          i := i + spec.inc;
        UNTIL i > spec.stop;
        END
      ELSE
        temp := - 999;

      IF spectra_debug OR error THEN
        BEGIN
        writeln('%SPECTRA_YMAX-INFO START=', spec.start: 0, ' END=', spec.stop: 0, ' INC=', spec.inc: 0);
        writeln('%SPECTRA_YMAX-INFO Minimum value = ', temp);
        IF spectra_debug THEN
          spectra_debug_pause_execution
        ELSE
          return(2);
        END;

      SPECTRA_YMAX := temp;
    END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SPECTRA_YMAX.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM SPECTRA_YMIN;
 {$nolist}
 {$list}


  FUNCTION SPECTRA_YMIN ((spec: spectral_data): real) ;
{
 WHERE
     spec = spectral data record
 RESULTS
     the minimum value in the spectral data record is returned
}

    VAR
      i: integer;
      temp: real;
      error: boolean;

    BEGIN
      error := false;

      IF spec.start = 0 THEN
        BEGIN
        error := true;
        writec('%SPECTRA_MIN-WARNING !!! data not in memory to sum');
        END;

      IF NOT error THEN
        BEGIN
        i := spec.start + spec.inc;
        temp := spec.val[1];
        REPEAT
          IF spec.val[(i - spec.start) DIV spec.inc + 1] < temp THEN
            temp := spec.val[(i - spec.start) DIV spec.inc + 1];
          i := i + spec.inc;
        UNTIL i > spec.stop;
        END
      ELSE
        temp := - 999;

      IF spectra_debug OR error THEN
        BEGIN
        writeln('%SPECTRA_YMIN-INFO START=', spec.start: 0, ' END=', spec.stop: 0, ' INC=', spec.inc: 0);
        writeln('%SPECTRA_YMIN-INFO Minimum value = ', temp);
        IF spectra_debug THEN
          spectra_debug_pause_execution
        ELSE
          return(2);
        END;

      SPECTRA_YMIN := temp;
    END;
```

```
Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]SPECTRA_YMIN.PASF;2/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
PROGRAM STASS;
 {$nolist}
 {$list}


  PROCEDURE STASS ((VAR t: PACKED ARRAY [tlow..thigh: integer] OF char; s: PACKED ARRAY [slow..shigh: integer] OF char))
(
 WHERE
     t = target string to hold a passable copy of the source conformant array string
     s = source string which is not passable by value since it a conformant array
 RESULTS
     the source string is copied to the target string
)
   { NOTE:  This routine is a slightly modified version of  the Pascal-2 ASSIGN procedure supplied in their string packag

    VAR
      slen: integer;
      i: integer;


    PROCEDURE string_abort(s: PACKED ARRAY [low..high: integer] OF char);
   (where
                    s = string to display on terminal before terminating program
   )

       VAR
         i: integer;

       BEGIN (abort)
         write('Illegal string argument in "');
         FOR i := low TO high DO
           write(s[i]);
         writeln('"');
         exitst(4);
       END (abort) ;


    BEGIN (strass)
      IF (tlow <> 0) AND (tlow <> 1) OR (slow <> 0) AND (slow <> 1) THEN
        string_abort('strass - bad arguments');
      IF slow = 0 THEN
        slen := ord(s[0])

   (User option here - the following code removes all trailing blanks
    from a literal string when inserting it into a variable string...
    This may be annoying to some users as you may wish the actually assign
    blanks to the end of a string in this manner.  To change the code
    comment out the following :

        else begin
         slen := shigh;
         while (s.....
         end;

    and replace the "else begin" with
```

```
60
61              else slen := shigh;
62
63          )
64              ELSE
65                BEGIN
66                slen := shigh;
67                WHILE (s[slen] = ' ') AND (slen > 1) DO
68                  slen := slen - 1;
69                END;
70              IF slen > thigh THEN
71                string_abort('strass - destination string too short');
72              FOR i := 1 TO slen DO
73                t[i] := s[i];
74              IF tlow = 0 THEN
75                t[0] := chr(slen)
76              ELSE IF tlow = 1 THEN
77                FOR i := slen + 1 TO thigh DO
78                  t[i] := ' ';
79           END (strass) ;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]STASS.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM STRASS;
 ($nolist)
 ($list)


  PROCEDURE STRASS ((VAR t: PACKED ARRAY [tlow..thigh: integer] OF char; VAR s: PACKED ARRAY [slow..shigh: integer] OF ch
(
 WHERE
     t = target string to hold a passable copy of the source conformant array string
     s = source string which is not passable by value since it a conformant array
 RESULTS
     the source string is copied to the target string
)
   { NOTE:  This routine is a slightly modified version of  the Pascal-2 ASSIGN procedure supplied in their string packag

    VAR
      slen: integer;
      i: integer;


    PROCEDURE string_abort(s: PACKED ARRAY [low..high: integer] OF char);
   (where
                    s = string to display on terminal before terminating program
   )

       VAR
         i: integer;

       BEGIN (abort)
         write('Illegal string argument in "');
         FOR i := low TO high DO
           write(s[i]);
         writeln('"');
         exitst(4);
       END (abort) ;


    BEGIN (strass)
      IF (tlow <> 0) AND (tlow <> 1) OR (slow <> 0) AND (slow <> 1) THEN
        string_abort('strass - bad arguments');
      IF slow = 0 THEN
        slen := ord(s[0])

   (User option here - the following code removes all trailing blanks
    from a literal string when inserting it into a variable string...
    This may be annoying to some users as you may wish the actually assign
    blanks to the end of a string in this manner.  To change the code
    comment out the following :

        else begin
         slen := shigh;
         while (s.....
         end;

    and replace the "else begin" with
```

```
60
61              else slen := shigh;
62
63          }
64             ELSE
65               BEGIN
66               slen := shigh;
67               WHILE (s[slen] = ' ') AND (slen > 1) DO
68                 slen := slen - 1;
69               END;
70             IF slen > thigh THEN
71               string_abort('strass - destination string too short');
72             FOR i := 1 TO slen DO
73               t[i] := s[i];
74             IF tlow = 0 THEN
75               t[0] := chr(slen)
76             ELSE IF tlow = 1 THEN
77               FOR i := slen + 1 TO thigh DO
78                 t[i] := ' ';
79           END (strass) ;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]STRASS.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
PROGRAM TRUNCATE_1_LIKE_2;
 ($nolist)
 ($list)


  FUNCTION TRUNCATE_1_LIKE_2 ((spec1, spec2: spectral_data; VAR error: boolean): spectral_data) ;
{
 WHERE
     spec1 = spectral data record number 1
     spec2 = spectral data record number 2
     error = signals error in parameters
 RESULTS
     a spectral data record truncated to the wavelength limits of the second
}

    VAR
      i: integer;
      temp: spectral_data;

    BEGIN
      error := false;
      IF spec1.start = 0 THEN
        BEGIN
        error := true;
        writec('%TRUNC-WARNING !!! data not in memory for sample 1');
        END;
      IF spec2.start = 0 THEN
        BEGIN
        error := true;
        writec('%TRUNC-WARNING !!! data not in memory for sample 2');
        END;

      IF NOT error THEN
        BEGIN

        temp := spec1;
        FOR i := 1 TO number_of_wavelengths DO
          temp.val[i] := 0;

        { find start, stop to truncate }
        IF spec1.start < spec2.start THEN
          temp.start := spec2.start;
        IF spec1.stop > spec2.stop THEN
          temp.stop := spec2.stop;
        { *** needs more error checking above }

        i := temp.start;
        REPEAT
          temp.val[(i - temp.start) DIV temp.inc + 1] := spec1.val[(i - spec1.start) DIV spec1.inc + 1];
          i := i + temp.inc;
        UNTIL i > temp.stop;
        END
      ELSE
        zero_spectra(temp);

```

```
60            TRUNCATE_1_LIKE_2 := temp;
61          END;
```

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]TRUNCATE_1_LIKE_2.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM ZERO_SPECTRA;
    2        {$nolist}
    7        {$list}
    8
    9
   10         PROCEDURE ZERO_SPECTRA ((VAR spec: spectral_data)) ;
   11       {
   12        WHERE
   13            spec = spectra data record
   14        RESULTS
   15            a zeroed spectral data record
   16       }
   17
   18           VAR
   19             i: integer;
   20             coord: coordinate_type;
   21
   22           BEGIN
   23             WITH spec DO
   24               BEGIN
   25               filename := '
   26               descrip := '                                                    ';
   27               name := '          ';                                                    ';
   28               start := 0;
   29               stop := 0;
   30               inc := 0;
   31               FOR i := 1 TO number_of_wavelengths DO
   32                 val[i] := - 999;
   33               FOR coord := succ(first) TO pred(last) DO
   34                 coordinates[coord] := - 999;
   35               END;
   36           END;

Invocation line:
USER1:[RMM.COLOR.SPECTRA.SOURCE]ZERO_SPECTRA.PASF;2/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***
```

```
    1       PROGRAM COLOR;
    2        {$nolist}
   14           {$list}
   15           { include the user i/o routines }
   16           %INCLUDE 'plb:milpk4';

plb:milpk4

2   1       { MILPK4.PAS }
2   2
2   3       %include 'plb:milpk4.hdr';

plb:milpk4.hdr

3   1       { FILE: MILPK4.HDR }
3   2
3   3       { MILPK4 is a set of generally useful user interface and assorted routines.  The source has not been included as these
3   4         routines are very similar to MILPAK, MILPK1, MILPK2, and MILPK3, some of which have already been included.
3   5
3   6         These routines were written or conceived by R. Mitchell Miller.  Some modifications and enhancements have been performe
3   7         by Eric Townsend of Computer Task Group and Dave Shannon of Mead Imaging.  Their assitance is greatly appreciated.
3   8
3   9       { MILPK4 CONST and TYPE declarations}
3  10
3  11       TYPE
3  12            default_type=(use_default,no_default);
3  13            null_type=(null_ok,no_null);
3  14            curmat_layout=(horizonal,vertical);
3  15            charset=set of char;
3  16            getsubdel = set of char ;
3  17
3  18            string3=packed array[1..3] of char;
3  19            string6=packed array[1..6] of char;
3  20            string14=packed array[1..14] of char;
3  21            string255=string[255];
3  22            date_string = PACKED ARRAY [1..8] OF char;
3  23
3  24
3  25       { MILPK4 PROCEDURE and FUCNTION declarations}
3  26
3  27       procedure abort; external;
3  28
3  29       function abtchr: char; external;
3  30
3  31       procedure ckterm(var error: boolean); external;
3  32
3  33       procedure curmat ( position, num_columns, num_rows, x_offset, y_offset
3  34                          : integer; layout: curmat_layout); external;
3  35
3  36       procedure getch (s: packed array[low..high:integer] of char;
3  37                        var value: char; default: default_type;
3  38                        acceptable: charset ; null_string: null_type); external;
3  39
3  40       procedure getfn(var temp: string14; defdev,defext: string3);  external;
3  41
```

```
3  42      procedure getin(s: packed array[low..high:integer] of char;
3  43                      var value: integer; default: default_type;
3  44                      low_val,high_val: integer); external;
3  45
3  46      procedure getre(s: packed array[low..high:integer] of char; var r: real;
3  47                      default: default_type;low_val,high_val: real); external;
3  48
3  49      procedure getsub( str : packed array [lo..hi:integer] of char;
3  50                        delimiters : getsubdel ;
3  51                        var chrptr : integer ;
3  52                        var substr : packed array [low..high:integer] of char;
3  53                        var delimiter : char ;
3  54                        var error : integer ;
3  55                        var eos : boolean ) ; external ;
3  56
3  57      PROCEDURE ival( st : packed array[lo..hi:integer] of char;
3  58                      var num, code : integer ) ;  external ;
3  59
3  60      procedure jump(x2: integer);  external;
3  61
3  62      procedure prompt(x1: integer);  external;
3  63
3  64      procedure readc(var q: char);  external;
3  65
3  66      procedure rdct(var q: char; time: integer;
3  67                           VAR timeout: boolean );  external;
3  68
3  69      procedure readch(s: packed array[low..high:integer] of char;
3  70                           var r: char;  default: default_type;
3  71                           null_string: null_type ); external;
3  72
3  73      procedure rdcht(s: packed array[low..high:integer] of char;
3  74                           var r: char;  default: default_type;
3  75                           null_string: null_type; time: integer;
3  76                           VAR timeout: boolean ); external;
3  77
3  78      procedure readfn(var temp: string14; defdev,defext: string3);  external;
3  79
3  80      procedure readin(s: packed array[low..high:integer] of char;
3  81                      var i: integer; default: default_type); external;
3  82
3  83      procedure readre(s: packed array[low..high:integer] of char; var r: real;
3  84                      default: default_type); external;
3  85
3  86      procedure readst(s1: packed array [low1..high1: integer] of char;
3  87                      var s: packed array[lo..high:integer] of char;
3  88                      default: default_type; null_string: null_type); external;
3  89
3  90      procedure return(x2: integer);  external;
3  91
3  92      procedure rpterr(s: packed array [low..hi: integer] of char);
3  93        external;
3  94
3  95      PROCEDURE rval( st : packed array[lo..hi:integer] of char;
```

```
3  96                       var num : real ; var code : integer ) ; external ;
3  97
3  98       function snglch: boolean; external;
3  99
3 100       function stpeof: boolean; external;
3 101
3 102       procedure start;  external;
3 103
3 104       procedure tell(direction: packed array[lower..upper:integer] of char);
3 105          external;
3 106
3 107       procedure wfreal(var out: text; value: real; width: integer); external;
3 108
3 109
3 110       procedure writec(s: packed array[lower..upper:integer] of char);  external;
3 111
3 112
3 113       FUNCTION eldays( date: date_string ): integer; external;
3 114
3 115       FUNCTION stdate( days: integer ): date_string; external;
3 116
3 117       PROCEDURE readdt(prompt: PACKED ARRAY [low1..high1: integer] OF char;
3 118                        VAR date: integer;
3 119                        default: default_type ); external;
3 120
3 121       PROCEDURE getdt(prompt: packed array[low1..high1:integer] of char;
3 122                       VAR date: integer;
3 123                       default: default_type;
3 124                       low_val, high_val: integer); external;
3 125
3 126       PROCEDURE copystr( VAR s1 : PACKED ARRAY [lo1..hi1:integer] OF char;
3 127                          VAR s2 : PACKED ARRAY [lo2..hi2:integer] OF char );EXTERNAL;
3 128
3 129       %include 'plb:ansi.hdr';
```

plb:ansi.hdr

```
4   1       { FILE: ANSI.HDR }
4   2
4   3       { ANSI TYPE and CONST declarations }
4   4
4   5       type  terminal_modes=(vt52,vt100);
4   6             cursor_movements=(home,up,down,left,right);
4   7             erase_options=(screen,line,eoscreen,boscreen,eol,bol);
4   8             screen_attributes=(off,bold,underscored,blinking,negative,
4   9                                not_bold,not_underscored,not_blinking,positive);
4  10             attribute_values=array [screen_attributes] of integer;
4  11             attribute_set= set of screen_attributes;
4  12
4  13       const attribute=attribute_values(0,1,4,5,7,22,24,25,27);
4  14
4  15       { ANSI PROCEDURE declarations }
4  16
4  17
```

plb:ansi.hdr

```
4  18      procedure cursor(location: cursor_movements; number: integer); external;
4  19
4  20      procedure erase(value: erase_options); external;
4  21
4  22      procedure gotoxy(column,line: integer); external;
4  23
4  24      procedure labdhw(x,y: integer;
4  25                s: packed array [low..high: integer] of char); external;
4  26
4  27      procedure labdw(x,y: integer;
4  28                s: packed array [low..high: integer] of char); external;
4  29
4  30      procedure setscn(value: attribute_set); external;
4  31
4  32      procedure settrm(mode: terminal_modes); external;   { ANSI screen control routines }
```

plb:milpk4.hdr

```
3 130
3 131     %include 'plb:pas2.hdr';
```

plb:pas2.hdr

```
4   1     { FILE: PAS2.HDR }
4   2
4   3     { Pascal-2 EXTERNAL PROCEDURES AND FUNCTIONS }
4   4
4   5       { This function reads a single character from the keyboard without
4   6         echoing it to the display }
4   7
4   8     function readsn: char; external;
4   9
4  10       { This procedure causes a program to exit
4  11         (when passes 1 normal exit,<>1=error }
4  12
4  13     procedure exitst ( status: integer ); external;
4  14
4  15       { This routine returns the current date and time of day }
4  16
4  17     procedure timestamp (var day, month, year, hour, minute, sec : integer );
4  18       external;
4  19
4  20       { This routine simulates random access to text files:
4  21         Get Position for SETPOS}
4  22
4  23     procedure getpos (var f: text; var block, offset: integer); external;
4  24
4  25       { This routine simulates random access to text files:
4  26         positions file pointer }
4  27
4  28     procedure setpos (var f: text; var block, offset: integer); external;
```

```
3 131        { Pascal-2 PROCEDURES and FUNCTIONS }
```

plb:milpk4

```
2   4
2   5       LABEL 1;
2   6
2   7       procedure abort;
2   8       begin
2   9         goto 1;
2  10       end;
2  11
2  12       function abtchr;
2  13       begin
2  14         abtchr:=abort_char;
2  15       end;
2  16
2  17       procedure prompt;
2  18       var x2: integer;
2  19       begin
2  20         erase(screen);
2  21         writec(program_name);
2  22         gotoxy(0,x1);
2  23       end;
2  24
2  25       function snglch;
2  26       begin
2  27         snglch:=singlechar;
2  28       end;
2  29
2  30       function stpeof;
2  31       begin
2  32         stpeof:=stop_on_eof;
2  33       end;
```

COLOR.PASF

```
   16
   17          {$nolist}
```

Invocation line:
COLOR.PASF/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM COLOR;
    2        {$nolist}

spec:spectra.hdr

3 162       {$LIST}
3 163         { SPLINE.HDR contains definitions for cubic spline and interpolation routines }
3 164         %INCLUDE 'spec:spline.hdr';

spec:spline.hdr

4   1       { SPLINE.HDR  contains definitions of procedures that perform taut cubic splines and interpolations of data
4   2
4   3
4   4          written by R. Mitchell Miller }
4   5
4   6       CONST
4   7         sxy_max_data = 300;
4   8
4   9       TYPE
4  10         {[F-]}
4  11            spline_xy_data = record
4  12              break : array[1..sxy_max_data] of real;    { returned by SPLINE_XY, used by INTERP_XY }
4  13              coef: array[1..4,1..sxy_max_data] of real; { returned by SPLINE_XY, used by INTERP_XY }
4  14              gamma: real;       { controls tautness, 0 = regular cubic, 2.5 = normal }
4  15              order,             { returned by SPLINE_XY, used by INTERP_XY }
4  16              number_of_points,  { specifies number of data points passed to SPLINE_XY }
4  17              number_of_knots,   { number of knots that describe the data, returned from SPLINE_XY }
4  18              deriv,             { specifies the n-th derivative of the function will be interpolated.
4  19                                   0 evaluates function }
4  20              index: integer;    { specifies which knot to start looking at, first = 1
4  21                                   (this saves time by bypassing initial knots when needed }
4  22              error: boolean;    { signals an error in SPLINE_XY or INTERP_XY }
4  23            end;
4  24         {[F+]}
4  25
4  26
4  27       PROCEDURE Spline_xy(VAR xdata, ydata: ARRAY [nlo..nhi: integer] OF real;
4  28                           VAR sd: spline_xy_data);
4  29         EXTERNAL;
4  30       {
4  31        WHERE
4  32            xdata = array of x data values to spline (in increasing x value)
4  33            ydata = array of y data values to spline
4  34            sd = record to hold the spline results
4  35        RESULTS
4  36            sd = spline results that can be used by Interp_xy
4  37       }
4  38
4  39       FUNCTION Interp_xy(x: real;
4  40                          VAR sd: spline_xy_data): real;
4  41         EXTERNAL;
4  42       {
4  43        WHERE
4  44            x = xvalue to interpolate a taut cubic spline value for
```

spec:spline.hdr

```
4  45          sd = spline results from Spline_xy
4  46       RESULTS
4  47          Interp_xy = y value interpolated for the x value passed to the routine
4  48      }
```

spec:spectra.hdr

```
3 165      {$NOLIST}
```

Invocation line:
COLOR.PASF/LIST/DOUBLE

*** No lines with errors detected ***

```
    1       PROGRAM INTERP_XY;
    2        {$nolist}

spec:spectra.hdr

2 162       {$LIST}
2 163         { SPLINE.HDR contains definitions for cubic spline and interpolation routines }
2 164         %INCLUDE 'spec:spline.hdr';

spec:spline.hdr

3   1       { SPLINE.HDR  contains definitions of procedures that perform taut cubic splines and interpolations of data
3   2
3   3
3   4          written by R. Mitchell Miller }
3   5
3   6       CONST
3   7         sxy_max_data = 300;
3   8
3   9       TYPE
3  10         {[F-]}
3  11            spline_xy_data = record
3  12              break : array[1..sxy_max_data] of real;    { returned by SPLINE_XY, used by INTERP_XY }
3  13              coef: array[1..4,1..sxy_max_data] of real; { returned by SPLINE_XY, used by INTERP_XY }
3  14              gamma: real;       { controls tautness, 0 = regular cubic, 2.5 = normal }
3  15              order,             { returned by SPLINE_XY, used by INTERP_XY }
3  16              number_of_points,  { specifies number of data points passed to SPLINE_XY }
3  17              number_of_knots,   { number of knots that describe the data, returned from SPLINE_XY }
3  18              deriv,             { specifies the n-th derivative of the function will be interpolated.
3  19                                   0 evaluates function }
3  20              index: integer;    { specifies which knot to start looking at, first = 1
3  21                                   (this saves time by bypassing initial knots when needed }
3  22              error: boolean;    { signals an error in SPLINE_XY or INTERP_XY }
3  23            end;
3  24         {[F+]}
3  25
3  26
3  27       PROCEDURE Spline_xy(VAR xdata, ydata: ARRAY [nlo..nhi: integer] OF real;
3  28                           VAR sd: spline_xy_data);
3  29         EXTERNAL;
3  30       {
3  31        WHERE
3  32            xdata = array of x data values to spline (in increasing x value)
3  33            ydata = array of y data values to spline
3  34            sd = record to hold the spline results
3  35        RESULTS
3  36            sd = spline results that can be used by Interp_xy
3  37       }
3  38
3  39       FUNCTION Interp_xy(x: real;
3  40                          VAR sd: spline_xy_data): real;
3  41         EXTERNAL;
3  42       {
3  43        WHERE
3  44            x = xvalue to interpolate a taut cubic spline value for
```

```
3  45          sd = spline results from Spline_xy
3  46       RESULTS
3  47          Interp_xy = y value interpolated for the x value passed to the routine
3  48      }

spec:spectra.hdr

2 165      {$NOLIST}

SOURCE:INTERP_XY.PASF

    7       {$list}
    8
    9
   10        FUNCTION INTERP_XY {(x: real; VAR sd: spline_xy_data): real} ;
   11      {
   12       WHERE
   13           x = xvalue to interpolate a taut cubic spline value for
   14           sd = spline results from Spline_xy
   15       RESULTS
   16           Interp_xy = y value interpolated for the x value passed to the routine
   17      }
   18
   19      { SPLINE_XY and INTERP_XY routines were converted from FORTRAN examples in the following reference:
   20
   21                  16.  C. de Boor, A Practical Guide to Splines,
   22                       Springer-Verlag, New York, 1978, pp. 299-320.
   23      }
   24
   25          VAR
   26            j, fmmjdr, n, p: integer;
   27            temp, dx: real;
   28
   29          LABEL
   30            9;
   31
   32
   33          BEGIN { Procedure INTERP }
   34            temp := 0;
   35            IF sd.error THEN
   36              GOTO 9;
   37
   38            { set index to a value of 1 so prevent array bounds errors }
   39            IF sd.index < 1 THEN
   40              sd.index := 1;
   41
   42            fmmjdr := sd.order - sd.deriv;
   43            IF fmmjdr <= 0 THEN
   44              BEGIN
   45              writeln('ERROR !!! Derivative not defined');
   46              sd.error := true;
   47              GOTO 9;
   48              END;
   49
```

```
50            IF (x < sd.break[1]) OR (x > sd.break[sd.number_of_knots + 1]) THEN
51              BEGIN
52              writeln('ERROR !!! Limits requested are out of range');
53              sd.error := true;
54              GOTO 9;
55              END;
56
57            WHILE (x > sd.break[sd.index + 1]) AND (sd.index < sd.number_of_knots) DO
58              sd.index := sd.index + 1;
59            dx := x - sd.break[sd.index];
60            p := fmmjdr;
61            FOR n := sd.order DOWNTO (sd.deriv + 1) DO
62              BEGIN
63              temp := temp / p * dx + sd.coef[n, sd.index];
64              p := p - 1;
65              END;
66
67          9:
68            BEGIN
69            IF sd.error THEN
70              BEGIN
71              writeln;
72              writeln('%INTERP_XY-E-Error was Encountered');
73              return(2);
74              END;
75            END;
76            INTERP_XY := temp;
77          END;

Invocation line:
SOURCE:INTERP_XY.PASF/NOMAIN/LIST/DOUBLE

 *** No lines with errors detected ***
```

```
    1       PROGRAM SPLINE_XY;
    2        ($nolist)

spec:spectra.hdr

2 162       ($LIST)
2 163         { SPLINE.HOR contains definitions for cubic spline and interpolation routines }
2 164         %INCLUDE 'spec:spline.hdr';

spec:spline.hdr

3   1       { SPLINE.HOR  contains definitions of procedures that perform taut cubic splines and interpolations of data
3   2
3   3
3   4          written by R. Mitchell Miller }
3   5
3   6       CONST
3   7         sxy_max_data = 300;
3   8
3   9       TYPE
3  10         {[F-]}
3  11           spline_xy_data = record
3  12             break : array[1..sxy_max_data] of real;    { returned by SPLINE_XY, used by INTERP_XY }
3  13             coef: array[1..4,1..sxy_max_data] of real; { returned by SPLINE_XY, used by INTERP_XY }
3  14             gamma: real;       { controls tautness, 0 = regular cubic, 2.5 = normal }
3  15             order,             { returned by SPLINE_XY, used by INTERP_XY }
3  16             number_of_points,  { specifies number of data points passed to SPLINE_XY }
3  17             number_of_knots,   { number of knots that describe the data, returned from SPLINE_XY }
3  18             deriv,             { specifies the n-th derivative of the function will be interpolated.
3  19                                  0 evaluates function }
3  20             index: integer;    { specifies which knot to start looking at, first = 1
3  21                                  (this saves time by bypassing initial knots when needed }
3  22             error: boolean;    { signals an error in SPLINE_XY or INTERP_XY }
3  23           end;
3  24         {[F+]}
3  25
3  26
3  27       PROCEOURE Spline_xy(VAR xdata, ydata: ARRAY [nlo..nhi: integer] OF real;
3  28                           VAR sd: spline_xy_data);
3  29         EXTERNAL;
3  30       {
3  31        WHERE
3  32            xdata = array of x data values to spline (in increasing x value)
3  33            ydata = array of y data values to spline
3  34            sd = record to hold the spline results
3  35        RESULTS
3  36            sd = spline results that can be used by Interp_xy
3  37       }
3  38
3  39       FUNCTION Interp_xy(x: real;
3  40                          VAR sd: spline_xy_data): real;
3  41         EXTERNAL;
3  42       {
3  43        WHERE
3  44            x = xvalue to interpolate a taut cubic spline value for
```

```
3  45          sd = spline results from Spline_xy
3  46       RESULTS
3  47          Interp_xy = y value interpolated for the x value passed to the routine
3  48       }
```

spec:spectra.hdr

```
2 165       {$NOLIST}
```

SOURCE:SPLINE_XY.PASF

```
    7       {$list}
    8
    9
   10        PROCEDURE SPLINE_XY ((VAR xdata, ydata: ARRAY [nlo..nhi: integer] OF real; VAR sd: spline_xy_data)} ;
   11       {
   12        WHERE
   13            xdata = array of x data values to spline (in increasing x value)
   14            ydata = array of y data values to spline
   15            sd = record to hold the spline results
   16        RESULTS
   17            sd = spline results that can be used by Interp_xy
   18       }
   19
   20       { SPLINE_XY and INTERP_XY routines were converted from FORTRAN examples in the following reference:
   21
   22                   16.  C. de Boor, A Practical Guide to Splines,
   23                        Springer-Verlag, New York, 1978, pp. 299-320.
   24       }
   25
   26           LABEL
   27             9;
   28
   29           VAR
   30             i, j, m, method, ntaum1: integer;
   31             alpha, c, d, del, denom, divdif, entry, entry3, factor, factr2, gam, onemg3, onemzt, ratio, sixth, temp, x, z, zeta
   32             zt2: real;
   33             s_temp: ARRAY [1..sxy_max_data, 1..6] OF real;
   34
   35             { Function ALPH local to procedure SPLINE }
   36
   37
   38           FUNCTION alph(x: real): real;
   39
   40             VAR
   41               x1: real;
   42
   43             BEGIN
   44               x1 := onemg3 / x;
   45               IF 1 < x1 THEN
   46                 alph := 1
   47               ELSE
   48                 alph := x1;
   49             END;
```

```
50
51
52          PROCEDURE docalc1;
53
54            BEGIN
55              sd.coef[2, m] := divdif - s_temp[i, 1] * (2 * s_temp[i, 4] + s_temp[i + 1, 4]) / 6;
56              sd.coef[4, m] := (s_temp[i + 1, 4] - s_temp[i, 4]) / s_temp[i, 1];
57            END;
58
59
60          PROCEDURE docalc2;
61
62            BEGIN
63              onemzt := gam * (1 - z);
64              IF onemzt = 0 THEN
65                BEGIN
66                sd.coef[2, m] := divdif;
67                sd.coef[3, m] := 0;
68                sd.coef[4, m] := 0;
69                END
70              ELSE
71                BEGIN
72                zeta := 1 - onemzt;
73                alpha := alph(zeta);
74                c := s_temp[i + 1, 4] * s_temp[i, 6];
75                d := s_temp[i, 4] / 6;
76                del := zeta * s_temp[i, 1];
77                sd.break[m + 1] := xdata[i] + del;
78                sd.coef[2, m] := divdif - s_temp[i, 1] * (2 * d + c);
79                sd.coef[4, m] := 6 * (c * alpha - d) / s_temp[i, 1];
80                m := m + 1;
81                sd.coef[4, m] := sd.coef[4, m - 1] + 6 * (1 - alpha) * c / (s_temp[i, 1] * sqr(onemzt) * onemzt);
82                sd.coef[3, m] := sd.coef[3, m - 1] + del * sd.coef[4, m - 1];
83                sd.coef[2, m] := sd.coef[2, m - 1] + del * (sd.coef[3, m - 1] + (del / 2) * sd.coef[4, m - 1]);
84                sd.coef[1, m] := sd.coef[1, m - 1] + del * (sd.coef[2, m - 1] + (del / 2) * (sd.coef[3, m - 1] + (del / 3) *
85                                 sd.coef[4, m - 1]));
86                END;
87            END;
88
89
90          PROCEDURE docalc3;
91
92            BEGIN
93              IF z = 0 THEN
94                BEGIN
95                sd.coef[2, m] := divdif;
96                sd.coef[3, m] := 0;
97                sd.coef[4, m] := 0;
98                END
99              ELSE
100               BEGIN
101               zeta := gam * z;
102               onemzt := 1 - zeta;
103               c := s_temp[i + 1, 4] / 6;
```

```
104                 d := s_temp[i, 4] * s_temp[i, 6];
105                 m := m + 1;
106                 del := zeta * s_temp[i, 1];
107                 sd.break[m] := xdata[i] + del;
108                 zt2 := sqr(zeta);
109                 alpha := alph(onemzt);
110                 factor := sqr(onemzt) * alpha;
111                 sd.coef[1, m] := ydata[i] + divdif * del + sqr(s_temp[i, 1]) * (d * onemzt * (factor - 1) + c * zeta * (zt2 - 1
112                 sd.coef[2, m] := divdif + s_temp[i, 1] * (d * (1 - 3 * factor) + c * (3 * zt2 - 1));
113                 sd.coef[3, m] := 6 * (d * alpha * onemzt + c * zeta);
114                 sd.coef[4, m] := 6 * (c - d * alpha) / s_temp[i, 1];
115                 sd.coef[4, m - 1] := sd.coef[4, m] - 6 * d * (1 - alpha) / (del * zt2);
116                 sd.coef[2, m - 1] := sd.coef[2, m] - del * (sd.coef[3, m] - (del / 2) * sd.coef[4, m - 1]);
117                 END;
118               END;
119
120
121             BEGIN { Procedure SPLINE }
122
123               FOR i := 1 TO sxy_max_data DO
124                 BEGIN
125                 FOR j := 1 TO 6 DO
126                   s_temp[i, j] := 0;
127                 END;
128
129               FOR i := 1 TO sxy_max_data DO
130                 BEGIN
131                 FOR j := 1 TO 4 DO
132                   sd.coef[j, i] := 0;
133                 sd.break[i] := 0;
134                 END;
135
136               sd.error := false;
137
138               IF sd.number_of_points < 4 THEN
139                 BEGIN
140                 writeln('Number of data points = ', sd.number_of_points: 1, '.  This must be greater than 4');
141                 writeln('   to use this SPLINE routine !!!');
142                 sd.error := true;
143                 GOTO 9;
144                 END;
145               ntaum1 := sd.number_of_points - 1;
146               FOR i := 1 TO ntaum1 DO
147                 BEGIN
148                 s_temp[i, 1] := xdata[i + 1] - xdata[i];
149                 IF s_temp[i, 1] <= 0 THEN
150                   BEGIN
151                   writeln('Data points are disordered !!!');
152                   writeln;
153                   writeln('Data [', i, '] = ', xdata[i], '     ', 'Data [', (i + 1), '] = ', xdata[i + 1]);
154                   sd.error := true;
155                   GOTO 9;
156                   END;
157                 s_temp[i + 1, 4] := (ydata[i + 1] - ydata[i]) / s_temp[i, 1];
```

```
158             END;
159           FOR i := 2 TO ntaum1 DO
160             s_temp[i, 4] := s_temp[i + 1, 4] - s_temp[i, 4];
161
162           i := 2;
163           s_temp[2, 2] := s_temp[1, 1] / 3;
164           sixth := 1 / 6;
165           method := 2;
166           gam := sd.gamma;
167           IF gam <= 0 THEN
168             method := 1
169           ELSE IF gam > 3 THEN
170             BEGIN
171             gam := gam - 3;
172             method := 3;
173             END;
174           onemg3 := 1 - gam / 3;
175           FOR i := 2 TO (ntaum1 - 1) DO
176             BEGIN
177             z := 0.5;
178             IF method <> 1 THEN
179               BEGIN
180               IF ((method = 2) AND ((s_temp[i, 4] * s_temp[i + 1, 4]) >= 0)) OR (method = 3) THEN
181                 BEGIN
182                 temp := abs(s_temp[i + 1, 4]);
183                 denom := abs(s_temp[i, 4]) + temp;
184                 IF denom <> 0 THEN
185                   BEGIN
186                   z := temp / denom;
187                   IF abs(z - 0.5) <= sixth THEN
188                     z := 0.5;
189                   END;
190                 END;
191               END;
192             s_temp[i, 5] := z;
193
194             IF z = 0.5 THEN
195               BEGIN
196               s_temp[i, 2] := s_temp[i, 2] + s_temp[i, 1] / 3;
197               s_temp[i, 3] := s_temp[i, 1] / 6;
198               END
199             ELSE IF z < 0.5 THEN
200               BEGIN
201               zeta := gam * z;
202               onemzt := 1 - zeta;
203               zt2 := sqr(zeta);
204               alpha := alph(onemzt);
205               factor := zeta / (alpha * (zt2 - 1) + 1);
206               s_temp[i, 6] := zeta * factor / 6;
207               s_temp[i, 2] := s_temp[i, 2] + s_temp[i, 1] * ((1 - alpha * onemzt) * factor / 2 - s_temp[i, 6]);
208               IF s_temp[i, 2] <= 0 THEN
209                 s_temp[i, 2] := 1;
210               s_temp[i, 3] := s_temp[i, 1] / 6;
211               END
```

```
212             ELSE
213               BEGIN
214               onemzt := gam * (1 - z);
215               zeta := 1 - onemzt;
216               alpha := alph(zeta);
217               factor := onemzt / (1 - alpha * zeta * (1 + onemzt));
218               s_temp[i, 6] := onemzt * factor / 6;
219               s_temp[i, 2] := s_temp[i, 2] + s_temp[i, 1] / 3;
220               s_temp[i, 3] := s_temp[i, 6] * s_temp[i, 1];
221               END;
222
223             IF i = 2 THEN
224               BEGIN
225               s_temp[1, 5] := 0.5;
226               s_temp[1, 2] := s_temp[1, 1] / 6;
227               s_temp[1, 3] := s_temp[2, 2];
228               entry3 := s_temp[2, 3];
229               IF z = 0.5 THEN
230                 BEGIN
231                 ratio := s_temp[2, 1] / s_temp[1, 2];
232                 s_temp[2, 2] := s_temp[2, 1] + s_temp[1, 1];
233                 s_temp[2, 3] := - 1 * s_temp[1, 1];
234                 END
235               ELSE IF z > 0.5 THEN
236                 BEGIN
237                 ratio := s_temp[2, 1] / s_temp[1, 2];
238                 s_temp[2, 2] := s_temp[2, 1] + s_temp[1, 1];
239                 s_temp[2, 3] := - 1 * s_temp[1, 1] * 6 * alpha * s_temp[2, 6];
240                 END
241               ELSE
242                 BEGIN
243                 factr2 := zeta * (alpha * (zt2 - 1) + 1) / (alpha * (zeta * zt2 - 1) + 1);
244                 ratio := factr2 * s_temp[2, 1] / s_temp[1, 2];
245                 s_temp[2, 2] := factr2 * s_temp[2, 1] + s_temp[1, 1];
246                 s_temp[2, 3] := - 1 * s_temp[1, 1] * factr2;
247                 END;
248               s_temp[2, 2] := ratio * s_temp[1, 3] + s_temp[2, 2];
249               s_temp[2, 3] := ratio * entry3 + s_temp[2, 3];
250               s_temp[1, 4] := s_temp[2, 4];
251               s_temp[2, 4] := ratio * s_temp[1, 4];
252               END
253             ELSE
254               BEGIN
255               s_temp[i, 2] := ratio * s_temp[i - 1, 3] + s_temp[i, 2];
256               s_temp[i, 4] := ratio * s_temp[i - 1, 4] + s_temp[i, 4];
257               END;
258
259             IF z = 0.5 THEN
260               BEGIN
261               ratio := - 1 * (s_temp[i, 1] / 6) / s_temp[i, 2];
262               s_temp[i + 1, 2] := s_temp[i, 1] / 3;
263               END
264             ELSE IF z > 0.5 THEN
265               BEGIN
```

```
266                 ratio := - 1 * (s_temp[i, 1] / 6) / s_temp[i, 2];
267                 s_temp[i + 1, 2] := s_temp[i, 1] * ((1 - zeta * alpha) * factor / 2 - s_temp[i, 6]);
268                 END
269               ELSE
270                 BEGIN
271                 ratio := - 1 * (s_temp[i, 6] * s_temp[i, 1] / s_temp[i, 2]);
272                 s_temp[i + 1, 2] := s_temp[i, 1] / 3;
273                 END;
274               END;
275             i := ntaum1;
276             s_temp[i, 5] := 0.5;
277             entry := ratio * s_temp[i - 1, 3] + s_temp[i, 2] + s_temp[i, 1] / 3;
278             s_temp[i + 1, 2] := s_temp[i, 1] / 6;
279             s_temp[i + 1, 4] := ratio * s_temp[i - 1, 4] + s_temp[i, 4];
280
281             IF z = 0.5 THEN
282               BEGIN
283               ratio := s_temp[i, 1] / s_temp[i - 1, 2];
284               s_temp[i, 2] := ratio * s_temp[i - 1, 3] + s_temp[i, 1] + s_temp[i - 1, 1];
285               s_temp[i, 3] := - 1 * s_temp[i - 1, 1];
286               END
287             ELSE IF z > 0.5 THEN
288               BEGIN
289               factr2 := onemzt * (alpha * (sqr(onemzt) - 1) + 1) / (alpha * (sqr(onemzt) * onemzt - 1) + 1);
290               ratio := factr2 * s_temp[i, 1] / s_temp[i - 1, 2];
291               s_temp[i, 2] := ratio * s_temp[i - 1, 3] + factr2 * s_temp[i - 1, 1] + s_temp[i, 1];
292               s_temp[i, 3] := - 1 * factr2 * s_temp[i - 1, 1];
293               END
294             ELSE
295               BEGIN
296               ratio := s_temp[i, 1] * 6 * s_temp[i - 1, 6] * alpha / s_temp[i - 1, 2];
297               s_temp[i, 2] := ratio * s_temp[i - 1, 3] + s_temp[i, 1] + s_temp[i - 1, 1];
298               s_temp[i, 3] := - 1 * s_temp[i - 1, 1];
299               END;
300
301             s_temp[i, 4] := ratio * s_temp[i - 1, 4];
302             ratio := - 1 * entry / s_temp[i, 2];
303             s_temp[i + 1, 2] := ratio * s_temp[i, 3] + s_temp[i + 1, 2];
304             s_temp[i + 1, 4] := ratio * s_temp[i, 4] + s_temp[i + 1, 4];
305
306             { Back Substitution }
307
308             s_temp[sd.number_of_points, 4] := s_temp[sd.number_of_points, 4] / s_temp[sd.number_of_points, 2];
309             FOR i := i DOWNTO 2 DO
310               s_temp[i, 4] := (s_temp[i, 4] - s_temp[i, 3] * s_temp[i + 1, 4]) / s_temp[i, 2];
311             s_temp[1, 4] := (s_temp[1, 4] - s_temp[1, 3] * s_temp[2, 4] - entry3 * s_temp[3, 4]) / s_temp[1, 2];
312
313             { Construct Polynomial Pieces }
314
315             sd.break[1] := xdata[1];
316             m := 1;
317
318             FOR i := 1 TO ntaum1 DO
319               BEGIN
```

```
320             sd.coef[1, m] := ydata[i];
321             sd.coef[3, m] := s_temp[i, 4];
322             divdif := (ydata[i + 1] - ydata[i]) / s_temp[i, 1];
323             z := s_temp[i, 5];
324             IF z = 0.5 THEN
325               docalc1
326             ELSE IF z > 0.5 THEN
327               docalc2
328             ELSE
329               docalc3;
330             m := m + 1;
331             sd.break[m] := xdata[i + 1];
332             END;
333
334           sd.number_of_knots := m - 1;
335           sd.order := 4;
336           sd.index := 1; {setup for INTERP_XY to start looking at the first knot }
337         9:
338           BEGIN
339           IF sd.error THEN
340             BEGIN
341             writeln;
342             writeln('Procedure SPLINE');
343             write('ERROR was encountered... Press RETURN to continue ');
344             readln;
345             END;
346           END;
347         END; { Procedure SPLINE }
```

Invocation line:
SOURCE:SPLINE_XY.PASF/NOMAIN/LIST/DOUBLE

*** No lines with errors detected ***

Vita

R. Mitchell Miller studied professional photography for one year at RIT before transferring into the Photographic Science program. While pursuing a BS and later the combined BS/MS degrees he received the Raymond Davis Scholarship from SPSE and the Neblette Scholarship from Mrs. Neblette. He has worked for RIT as a Research Associate/Lecturer in the Imaging and Photographic Science Division, as a Research Scientist in the Eastman Kodak Research Laboratories, and currently holds the position of Research Scientist at Mead Imaging.