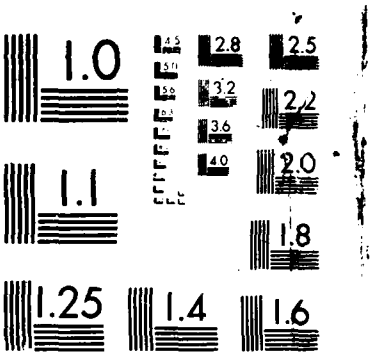
AD-A195 181 PROTOTYPE TECHNOLOGY FOR MONITORING VOLATILE ORGANICS 1/2
VOLUME 2(U) S-CUBED LA JOLLA CA V TAYLOR ET AL. MAR 88
SSS-R-87-8515-VOL-2 ESL-TR-88-01-VOL-2 F08635-84-C-0290

UNCLASSIFIED F/G 7/3 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

DTIC FILE COPY

AD-A195 101

ESL-TR-88-01
VOL II

PROTOTYPE TECHNOLOGY FOR MONITORING VOLATILE ORGANICS, VOLUME II

V. TAYLOR, J. WANDER

S-CUBED CORPORATION
P.O. BOX 1620
LA JOLLA CA 92038-1620

MARCH 1988

FINAL REPORT

AUGUST 1984-NOVEMBER 1987

DTIC
ELECTE
MAY 23 1988
S D
H

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

AFESC

ENGINEERING & SERVICES LABORATORY
AIR FORCE ENGINEERING & SERVICES CENTER
TYNDALL AIR FORCE BASE, FLORIDA 32403

88 5 23 095

NOTICE

The following commercial products (requiring Trademark ®) are mentioned in this report. Because of the frequency of usage, the Trademark was not indicated. If it becomes necessary to reproduce any segment of this document containing any of these names, this notice must be included as part of that reproduction.

Carbopak B
Chromosorb WHP
Data Translation DT2085 AD Board
Epson FX-80 Printer
Hewlett-Packard 5890 Gas Chromatograph
IBM Personal Computer
Luer-Lok
Milli-Q
Millipore
Teflon
Tekmar Automatic Liquid Sampler (ALS, ALS-10)
Tekmar Liquid Sample Concentrator (LSC-2)
Tenax GC
Topaz 5340-00P3 Transformer
Varian 3400 Gas Chromatograph

Mention of the products listed above does not constitute Air Force endorsement or rejection of this product, and use of information contained herein for advertising purposes without obtaining clearance according to existing contractual agreements is prohibited.

Please do not request copies of this report from HQ AFESC/RD (Engineering and Services Laboratory). Additional copies may be purchased from:

Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22314

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA195101

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SSS-R-87-8515	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESL-TR-88-01 Vol II
6a. NAME OF PERFORMING ORGANIZATION S-Cubed Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION HQ AFESC/RDVS
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 1620 La Jolla CA 92038-1620	7b. ADDRESS (City, State, and ZIP Code) Tyndall AFB FL 32403-6001
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-84-C-0298
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
63723F	2103	20	11

11. TITLE (Include Security Classification)

Prototype Technology for Monitoring Volatile Organics, Vol II

12. PERSONAL AUTHOR(S)

Taylor, Victoria, and Wander, Joseph

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 84 Aug TO 87 Nov	March 1988	110(V.2)

16. SUPPLEMENTARY NOTATION

Availability of this report is specified on reverse of front cover.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Purge and trap, gas chromatograph, volatile organic compounds, trichloroethylene, VOC, TCE, water
24	04		
24			

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the development and implementation of the phase I prototype volatile organic compound (VOC) monitor, a turn-key system for analyzing trichloroethylene (TCE) in water. The system incorporates commercial purge and trap and gas chromatography instruments, a microcomputer, and custom-written software. Because this monitor is planned for installation at base facilities and for use by nonspecialized personnel, driving design considerations were reliability and simplicity of operation.

Assembly and lab testing of the unit were completed in September 86, and the monitor was transitioned to the sewage plant (379 CES/DEMH) at Wurtsmith AFB MI for evaluation by base personnel. Most logistical accommodations had been made within 6 months, and, except for one construction related accident, the system worked exactly as planned. Companion documents, including schematics and operating and maintenance instructions, are included in Vol. 1; Vol 2 is a complete downlisting of the software.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph D. Wander	22b. TELEPHONE (Include Area Code) 904 283-4234 22c. OFFICE SYMBOL HQ AFESC/RDVS

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

(The reverse of this page is blank.)

PREFACE

This report was prepared by S-CUBED, a Division of Maxwell Laboratories, Inc., 3398 Carmel Mountain Road, San Diego, California 92121, under Contract Number F08635-84-C-0298, for the Air Force Engineering and Services Center, Engineering and Services Laboratory (AFESC/RDVS), Tyndall Air Force Base, Florida 32403-6001.

This report summarizes the work done between August 23, 1984, and June 30, 1987. The HQ AFESC/RDVS Project Officers were Lt. Robert C. Beggs (1984-February 1985), Maj. Kenneth T. Denbleyker (February 1985 - June 1986), and Dr. Joseph Wander (July 1986 - 1987). Volume I contains the project report and two handbooks written to accompany the system in the field; Volume II contains a complete source code listing of the data acquisition software (in C), written by Ms. Barbara Lentz.

This report has been reviewed by the Public Affairs Officer (PA) and it may be released to the National Technical Information Service (NTIS). At NTIS, this document will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

JOSEPH D. WANDER
Project Officer

THOMAS J. WALKER, Lt Col, USAF, BSC
Chief, Environics Division

KENNETH T. DENBLEYKER, Maj, USAF
Chief, Environmental Sciences Branch

LAWRENCE D. HOKANSON, Col, USAF
Director, Engineering and Services Laboratory

(The reverse of this page is blank.)

TABLE OF CONTENTS

Section	Title	Page
Appendix		
D	Source Code	1

DTIC COPY INSPECTED 1

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification

By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

(The reverse of this page is blank.

APPENDIX D

SOURCE CODE

Material contained in this appendix has been published without change from its original format.

```
/*********************************************************************/
/* adtconv.c                                                         */
/* adtconv.c  collects data from GC and converts it to digital voltage */
/* modification: store results in int array during conversion instead  */
/* of storing individual byte values                                 */
/* levels.  Base address is set for 0x2DE.                           */
/* created 4/8/85          B. Lentz                                  */
/*                                                                   */
/*********************************************************************/

#include "bitset.h"       /* bit setting macros */
#include <stdio.h>

adtconv(dataval, gain, nconv, period)
unsigned int *dataval;        /* raw data converted to decimal */
unsigned int gain;            /* value used in voltage calculation */
long nconv;                   /* number of data conversions to perform */
long period;                  /* number of usec between points (freq) */
{
unsigned char temp;        /* used to clear out register */
char Hperiod;              /* high byte of clock period */
char Lperiod;              /* low period of clock period */
char Hno;                  /* high byte of # of conversions */
char Lno;                  /* low byte of # of conversions */
long loop;                 /* counter for conversion loop */
char error1;               /* first byte of error code */
char error2;               /* second byte of error code */
unsigned char adl;         /* low byte of raw data */
unsigned int sum;         /* value of 1st pt to be averaged */
char setgain;              /* set gain on board */
float volts;               /* voltage of data value */
long index;
int k;
extern unsigned char inportb();
extern unsigned char outportb();

unsigned char start_chan = '\0';   /* only using one channel */
unsigned char end_chan = '\0';

if (gain == 1)
  setgain = '\0';
else if (gain == 10)
  setgain = '\1';
else if (gain == 100)
  setgain = '\2';
else if (gain == 500)
  setgain = '\3';
else
{
  fprintf(stderr, "\nILLEGAL VALUE FOR GAIN\n");
  exit(0);
}

if ((STAT_REG & 0x70) != 0)
{
```

```
56        fprintf(stderr, "\nFATAL ERROR-Illegal status register value\n");
57        fprintf(stderr, "\nStatus Register value is %o\n", STAT_REG);
58        exit(0);
59      }
60        /* stop and clear */
61      COMM_REG(CSTOP);
62      temp = DATA_OUT;
63      while(!(STAT_REG & COMM_WAIT));
64      COMM_REG(CCLEAR);
65
66      while(!(STAT_REG & COMM_WAIT));
67      COMM_REG(CCLOCK);
68      Hperiod = (period/256) & 0377;
69      Lperiod = (period - (Hperiod << 8)) & 0377;
70      while(STAT_REG & WRITE_WAIT);
71      DATA_IN(Lperiod);
72      while(STAT_REG & WRITE_WAIT);
73      DATA_IN(Hperiod);
74
75      while(!(STAT_REG & COMM_WAIT));
76      COMM_REG(CSAD);
77      while(STAT_REG & WRITE_WAIT);
78      DATA_IN(setgain);
79      while(STAT_REG & WRITE_WAIT);
80      DATA_IN(start_chan);
81      while(STAT_REG & WRITE_WAIT);
82      DATA_IN(end_chan);
83
84      Hno = (nconv/256) & 0377;
85      Lno = (nconv - (Hno << 8)) & 0377;
86      while(STAT_REG & WRITE_WAIT);
87      DATA_IN(Lno);
88      while(STAT_REG & WRITE_WAIT);
89      DATA_IN(Hno);
90
91      while (!(STAT_REG & COMM_WAIT));
92      COMM_REG(CRAD);
93
94      index = nconv >> 3;    /* divide by 8 */
95
96      for(loop = 0; loop < index; loop++)
97      {
98        for(k = 0; k < 7; k++)
99        {
100         while(!(STAT_REG & READ_WAIT));
101         adl = DATA_OUT;
102         while(!(STAT_REG & READ_WAIT));
103         adl = DATA_OUT;
104       }
105       while(!(STAT_REG & READ_WAIT));
106       adl = DATA_OUT;
107       while(!(STAT_REG & READ_WAIT));
108       dataval[loop] = (int)(DATA_OUT << 8) + adl;
109     }
110
```

```
111   while(!(STAT_REG & COMM_WAIT));
112   temp = STAT_REG;
113   if(temp & 0x80)
114     ioerr();
115 }
116
117
118
119
```

```
1   /*********************************************************************/
2   /* analyz.c                                                          */
3   /* analyz.c    controls sample collection and analysis modules       */
4   /* created 5/6/86       B. Lentz                                     */
5   /*                                                                   */
6   /* modified 6/23/86                                                  */
7   /*********************************************************************/
8   #include <stdio.h>
9   #include "parm.h"
10  #include "targ.h"
11
12  struct sample samp[MAXUNKS];
13  struct value usr, s3;    /* contains operating parameters */
14
15  main(argc, argv)
16  int argc;
17  char *argv[];
18  {
19  int frstrun;         /* flag first run of day */
20
21  printf("\nInsert data diskette into Drive B\n");
22  g_parm(&usr, &s3);
23  g_sspec(samp);
24  frstrun = caldate();
25
26  if(atoi(argv[1]))
27    usr.calflag = TRUE;
28  if(usr.calflag)
29  {
30    printf("\nCALIBRATION IS REQUIRED\n\n");
31    stdseq();
32    collect(1);
33  }
34  else
35  {
36    seqenc(frstrun);
37    collect(0);
38  }
39  }
40
```

```
/*********************************************************************/
/* analyz.c    controls sample collection and analysis modules       */
/* created 5/6/86       B. Lentz                                     */
/* modified 7/15/86  B. Lentz                                        */
/* printed                                                           */
/*********************************************************************/
#include <stdio.h>
#include "parm.h"
#include "targ.h"

struct sample samp[MAXUNKS];
struct value usr, s3;    /* contains operating parameters */

main(argc, argv)
int argc;
char *argv[];
{
int frstrun;         /* flag first run of day */

printf("\nInsert data diskette into Drive B\n");
g_parm(&usr, &s3);
g_sspec(samp);
   /* set date and time and check date of last calibration */
frstrun = caldate();

if(atoi(argv[1]))
  usr.calflag = TRUE;
if(usr.calflag)
  {
  printf("\nCALIBRATION IS REQUIRED\n\n");
  stdseq();
  collect(1);
  }
else
  {
      /* calibration not required */
  seqenc(frstrun);
  collect(0);
  }
}
```

```
*****************************************************************/
/* caldate                                                      */
/* Checks date of last calibration                              */
/* Created 7/3/86    B. Lentz                                   */
/*                                                              */
/*****************************************************************/

#include
#define
#define


extern usr;


                /* month of last calibration */
                /* day of last calibration */
                /* calibration date in days since previous month */
                /* flag first run of day */

time();
{
    %2d%*c%2d", &month, &day);
ndate[0]) && (day == ndate[1]))
GOTO;

Time %s\n", usr.date, usr.time);
calibration date was %s\n", usr.caldate);
ndate, "%2d%*c%2d", &month, &day);
month)


day) > 7)
;


ndate[1] + montday[ndate[0]];
day) > 7)
```

```
/*********************************************************************/
/* coltest.c                                                         */
/* collect raw data and analyze it.  Display retention times and areas  */
/* test response to trigger                                          */
/* 2/6/86      B. Lentz                                              */
/* modified 4/21/86                                                  */
/* used for testing with temp files                                  */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "bitset.h"
#define  MINMATCH 0.7

collect(calrun)
int calrun;       /* is this a calibration run ? */
{
extern struct value usr;
int position;             /* sample position */
int npeaks;
int j;
int xmax[NPEAKS];
float area[NPEAKS];           /* areas for all peaks */
float rettm;              /* retention times for all peaks */
unsigned int *dataval;
long tarea[MAXSTDS];      /* selected areas that correspond to stds & target */
int trettm[MAXSTDS];      /* selected retention times from sample */
float hit;                /* match factor (pattern ident) */
float match();            /* find std and target peaks */
FILE *prnptr;             /* used to send data to printer */
FILE *floppy;             /* used to send data to floppy file */
char rspns;               /* first letter of user response */

   /* open printer file */
 if((prnptr = fopen("PRN:", "wb")) == 0)
 {
  fprintf(stderr, "\nError transmitting data to printer \n");
  return;
 }
   /* send carriage return and form feed signal to Epson printer */
 fprintf(prnptr, "%c%c", 12, 13);

 printf("\nLoad Blank in position 1 and Start Tekmar Purge and Trap\n");
 do {
  printf("\n\nIs the 'Purge Complete' light illuminated on the LSC? (y or n) :");
 } while(!(rspns = reply()));
 if(rspns < 0)     /* exit */
   return;

 usr.invalid = FALSE;

 for(position = 0; position < usr.sampcnt; position++)
 {
  setdate();
  printf("\n\n\tAnalyzing SAMPLE # %d\t%s\n", position+1, usr.number[position]);
```

```
position, xmax, area);
position);

\n\tpeak\tretention time\t\tarea\n\n");
npeaks; j++)

[j]/(usr.freq*60.0);
\t %d \t  %.3f \t\t%.0f\n", j, rettm, area[j]);


position, npeaks, xmax, area, 0);
position, 1, -1.0, 0.0);
\n*** WARNING *** chromatography problem \n Detected only %d peaks\n", npeaks);
\n*** WARNING *** chromatography problem \n Detected only %d peaks\n", npeaks);

40)

position, npeaks, xmax, area, 0);
position, 1, -1.0, 0.0);
\n*** WARNING *** Sample is too complex for automated analysis\n");
\n*** WARNING *** Sample is too complex for automated analysis\n");


(npeaks, xmax, area, tarea, trettm);
tion, npeaks, xmax, area, hit);
\n\tMatch factor %f\n", hit);
MMAT(H)

\n\n*** WARNING *** Problem with chromatography - Internal Standard peaks not found\n");
\n\n*** WARNING *** Problem with chromatography - Internal Standard peaks not found\n");
= TRUE;


position, trettm, tarea, &calrun);
peaks */
position */
return and form feed signal to Epson printer */
12, 13);
```

```
/*********************************************************************/
/* collect.c                                                         */
/* collect raw data and analyze it.  Display retention times and areas  */
/* test response to trigger                                          */
/* created 2/6/86      B. Lentz                                      */
/*                                                                   */
/* modified 4/21/86                                                  */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "bitset.h"

collect(calrun)
int calrun;
{
extern struct value usr;
int position;              /* sample position */
int npeaks;
int j;
int xmax[NPEAKS];
float area[NPEAKS];
float rettm;
unsigned int *dataval;
long tarea[MAXSTDS];       /* selected areas that correspond to stds & target */
int trettm[MAXSTDS];       /* selected retention times from sample */
float hit;                 /* match factor (pattern ident) */
float match();             /* find std and target peaks */
FILE *prnptr;              /* used to send data to printer */
FILE *floppy;              /* used to send data to floppy file */
char rspns;                /* first letter of user response */
unsigned int core;         /* amount of memory available */
extern unsigned int coreleft();

if((prnptr = fopen("PRN:", "wb")) == 0)
{
  fprintf(stderr, "\nError transmitting data to printer \n");
  return;
}
fprintf(prnptr, "%c%c", 12, 13);

printf("\nLoad Blank in position 1 and Start Tekmar Purge and Trap\n");
do {
  printf("\n\nIs the 'Purge Complete' light illuminated on the LSC? (y or n) :");
} while(!(rspns = reply()));
if(rspns < 0)     /* exit */
  return;

initdio();
if(usr.verbose)
{
  printf("\ngain %u nconv %0 period %0\n", usr.gain, usr.nconv, usr.period);
  printf("runtime %f min\tfrequency %f pts/sec\n\n", usr.runtime, usr.freq);
  printf("threshold values %d\t %d\t %d\t %d\n", usr.thresh[1], usr.thresh[2], usr.thresh[3], usr.thresh[4]);
}
```

```
56  core = coreleft();
57  if(core < (usr.numpt * 2))
58  {
59    fprintf(stderr, "\n\nFATAL ERROR not enough memory available for data arrays\n");
60    return;
61  }
62  usr.invalid = FALSE;
63
64  for(position = 0; position < usr.sampcnt; position++)
65  {
66    setdate();
67    printf("\n\n\tAnalyzing Sample # %d\t%s\n", position+1, usr.number[position]);
68    if ((dataval = alloc(usr.numpt * 2)) == 0)
69    {
70      fprintf(stderr, "not enough memory available\n");
71      return;
72    }
73    dataval = &dataval[0];
74
75    setdio(01, CDIOOUT);
76
77    setdio(0, XTRIG);    /* wait for trigger */
78    printf("\nCollecting Data \n");
79
80    adtconv(dataval, usr.gain, usr.nconv, usr.period);
81
82    smooth(dataval, usr.numpt, usr.smooth);
83
84    npeaks = pk_det(usr.numpt, dataval, &usr, xmax, area);
85    prnres(prnptr, position);
86    if(npeaks > 0)
87    {
88      fprintf(prnptr, "\n\tpeak\tretention time\t\tarea\n\n");
89      for(j = 1; j <= npeaks; j++)
90      {
91        rettm = xmax[j]/(usr.freq*60.0);
92        fprintf(prnptr, "\t %d \t  %.3f \t\t%.0f\n", j, rettm, area[j]);
93      }
94    }
95    if(npeaks < 3)
96    {
97      stoflpy(position, npeaks, xmax, area, 0);
98      fprintf(prnptr, "\n*** WARNING *** chromatography problem \n Detected only %d peaks\n", npeaks);
99      fprintf(stderr, "\n*** WARNING *** chromatography problem \n Detected only %d peaks\n", npeaks);
100   }
101   else if(npeaks > (NPEAKS-1))
102   {
103     stoflpy(position, npeaks, xmax, area, 0);
104     fprintf(prnptr, "\n*** WARNING *** Sample is too complex for automated analysis \n );
105     fprintf(stderr,  \n*** WARNING *** Sample is too complex for automated analysis \n );
106   }
107   else
108   {
109     hit = match(npeaks, xmax, area, tarea, trettm);
110     stoflpy(position, npeaks, xmax, area, hit);
```

```
111        fprintf(prnptr, "\n\tMatch factor %f\n", hit);
112        if(hit < 0.7)
113        {
114          fprintf(prnptr, "\n\n*** WARNING *** Problem with chromatography - Internal Standard peaks not found\n");
115          fprintf(stderr, "\n\n*** WARNING *** Problem with chromatography - Internal Standard peaks not found\n");
116          usr.invalid = TRUE;
117        }
118        else
119          sampler(prnptr, position, trettm, tarea, &calrun);
120      }
121      free(dataval);
122    }
123    fprintf(prnptr, "%c%c", 12, 13);
124    fclose(prnptr);
125  }
126
```

```
1   /*********************************************************************/
2   /* chgtgt.c                                                          */
3   /* chgtgt.c    Change target paramters                               */
4   /* created 8/26/85   B. Lentz                                        */
5   /*                                                                   */
6   /* revised 6/9/86                                                    */
7   /*********************************************************************/
8
9   #include <stdio.h>
10  #include "parm.h"
11  #include "targ.h"
12
13  chgtgt()
14  {
15  extern struct value usr;      /* user parameters */
16  struct sample samp[MAXUNKS];  /* target paramters */
17  char option;                  /* selected option to change */
18  long inum;                    /* numerical value of character input */
19  long targ;                    /* selected target cmpd to change */
20  long atoi();
21  float fnumber;                /* floating point number */
22  float g_fnum();               /* get floating point number */
23  char string[30];              /* user input */
24  int i;                        /* loop counter */
25  char newval[20];              /* new value of parameter */
26  char name[20];                /* parameter name */
27  FILE *fileptr;
28
29     /* clear the screen */
30  system("cls");
31  printf("Reading current values of parameters\nPlease wait...\n");
32  g_sspec(samp);
33  targ = 1;
34
35  if((fileptr = fopen(targfile[targ], "rw")) == 0)
36  {
37    fprintf(stderr, "error opening parameter file\n");
38    return;
39  }
40
41  while(0 == 0)
42  {
43    system("cls");
44    printf("\n\n\na)  Target compound is %s\n", samp[targ].name);
45    printf("b)  Retention Time is %.2f min\n", (samp[targ].rt)/(usr.freq * 60.0));
46    printf("c)  Units of Concentration %s\n", samp[targ].units);
47    printf("d)  Maximum allowable water sample limit(warning limit) is %.2f\n", samp[targ].cutoff);
48    printf("e)  Spike concentration is %.2f \n", samp[targ].spike);
49    printf("f)  Concentration of %s in standard samples:\n", samp[targ].name);
50    printf("\tConcentration in standard A is %.2f\n", samp[targ].conc[1]);
51    printf("\tConcentration in standard B is %.2f\n", samp[targ].conc[2]);
52    printf("\tConcentration in standard C is %.2f\n", samp[targ].conc[3]);
53    printf("g)  Precision: maximum difference between duplicate spike samples = %.2f\n", samp[targ].precisn);
54    printf("h)  Accuracy: maximum difference between actual and measured results = %.2f\n", samp[targ].accurcy);
55    printf("i)  Top concentration of linear range = %.2f %s\n", samp[targ].dilution, samp[targ].units);
```

```
56      printf("j)  Quantitiation limit is %.1f %s\n", samp[targ].qtlimit, samp[targ].units);
57      printf("x)  Exit\n");
58      printf("\nEnter letter of option to change:  ");
59      option = getchar();
60      getchar();
61
62      switch(option)
63      {
64        case 'a':
65        case 'A':
66        {
67          printf("\nChanging name of target compound #%D\n", targ);
68          printf("\nEnter new compound name:  ");
69          if(fgets(string, 25, stdin) == 0)
70          {
71            printf("\nInvalid compound name\nTarget name not changed\n");
72            break;
73          }
74          wrparm("name", string, fileptr);
75          break;
76        }
77        case 'b':
78        case 'B':
79        {
80          printf("\n\n Changing Retention Time (in minutes) of target compound %D", targ);
81          if((fnumber = g_fnum(string)) < 0.0)
82          {
83            printf("\nRetention time of %s not changed\n", samp[targ].name);
84            break;
85          }
86          inum = fnumber * usr.freq * 60.0;
87          if (fnumber > usr.runtime)
88          {
89            printf("\nInvalid Retention Time - Larger than run time\n");
90            printf("\nRetention time of %s not changed\n", samp[targ].name);
91            break;
92          }
93          sprintf(newval, "%d", inum);
94          wrparm("rettime", newval, fileptr);
95          break;
96        }
97        case 'c':
98        case 'C':
99        {
100         printf("\nChanging Units of Concentration\n");
101         printf("\nEnter new unit name:  ");
102         if(fgets(string, 25, stdin) == 0)
103         {
104           printf("\nInvalid name, unit name not changed\n");
105           break;
106         }
107         wrparm("name", string, fileptr);
108         break;
109       }
110       case 'd':
```

```
111     case 'D':
112     {
113       printf("\n\n Changing Maximum allowable limit for concentration");
114       if((inum = getnum(newval)) < 0)
115       {
116         printf("\n Invalid cutoff limit, value not changed\n");
117         break;
118       }
119       wrparm("cutoff", newval, fileptr);
120       break;
121     }
122     case 'e':
123     case 'E':
124     {
125       printf("\n\n Changing concentration of spike added");
126       if((inum = getnum(newval)) < 0)
127       {
128         printf("\n Invalid spike concentration, value not changed\n");
129         break;
130       }
131       wrparm("spike", newval, fileptr);
132       break;
133     }
134     case 'f':
135     case 'F':
136     {
137       printf("\n\n Changing concentration of %s in standard A", samp[targ].name);
138       if((inum = getnum(newval)) < 0)
139       {
140         printf("\n Invalid concentration, value not changed\n");
141         break;
142       }
143       wrparm("stdconc1", newval, fileptr);
144       printf("\n\n Changing concentration of %s in standard B", samp[targ].name);
145       if((inum = getnum(newval)) < 0)
146       {
147         printf("\n Invalid concentration, value not changed\n");
148         break;
149       }
150       wrparm("stdconc2", newval, fileptr);
151       printf("\n\n Changing concentration of %s in standard C", samp[targ].name);
152       if((inum = getnum(newval)) < 0)
153       {
154         printf("\n Invalid concentration, value not changed\n");
155         break;
156       }
157       wrparm("stdconc3", newval, fileptr);
158       break;
159     }
160     case 'g':
161     case 'G':
162     {
163       printf("\n\n Changing precision control value");
164       if((inum = getnum(newval)) < 0)
165       {
```

```
166            printf("\n Invalid precision, value not changed\n");
167            break;
168            }
169          wrparm("precisn", newval, fileptr);
170          break;
171          }
172        case 'h':
173        case 'H':
174        {
175          printf("\n\n Changing accuracy control value");
176          if((inum = getnum(newval)) < 0)
177          {
178            printf("\n Invalid accuracy, value not changed\n");
179            break;
180            }
181          wrparm("accurcy", newval, fileptr);
182          break;
183          }
184        case 'i':
185        case 'I':
186        {
187          printf("\n\n Changing top of linear range. Concentration should be in %s", samp[targ].units);
188          if((inum = getnum(newval)) < 0)
189          {
190            printf("\n Invalid concentration, linear range not changed\n");
191            break;
192            }
193          wrparm("dilution", newval, fileptr);
194          break;
195          }
196        case 'j':
197        case 'J':
198        {
199          printf("\n\n Changing quantitation limit. Concentration should be in %s", samp[targ].units);
200          if((inum = getnum(newval)) < 0)
201          {
202            printf("\n Invalid concentration, quantitation limit not changed\n");
203            break;
204            }
205          wrparm("qtlimit", newval, fileptr);
206          break;
207          }
208
209        case 'x':
210        case 'X':
211        {
212          fclose(fileptr);
213          return;
214          break;
215          }
216        default:
217        {
218          printf("\n\tInvalid selection\n");
219          break;
220          }
```

```
221 │ │ └─}
222 │ │  g_sspec(samp);
223 │ └─}
224 └─}
225
226
227
```

```
1   /*********************************************************************/
2   /* change.c                                                          */
3   /* change.c  Changes values of user parameters in "parm" file        */
4   /* created 8/15/85              B. Lentz                             */
5   /*                                                                   */
6   /* revised 6/12/85                                                   */
7   /*********************************************************************/
8
9   #include <stdio.h>
10  #include "parm.h"
11
12  struct value usr, s3;
13
14  main()
15  {
16  char newval[10];
17  char string[10];
18  int base = 0;
19  long inumber;
20  long atoi();
21  float fnumber;
22  float g_fnum();
23  long getnum();
24  long i;
25  char name[20];
26  int option;
27  int level;        /* level of option to change */
28  int rspn;
29  FILE *fptr;
30
31  if((fptr = fopen(targfile[0], "rw")) == 0)
32  {
33  fprintf(stderr, "\tERROR opening parameter file\n");
34  return;
35  }
36  while(0 == 0)
37  {
38  g_parm(&usr, &s3);
39  prnparm(&usr);
40  printf("\nEnter letter of option to change: ");
41  option = getchar();
42  getchar();
43
44  switch(option)
45  {
46  case 'a':
47  case 'A':
48  {
49  printf("\n\n\n");
50  printf("\n\nChanging Verbose Option\n");
51  do
52  {
53  printf("\nSet Verbose option ON ? (y or n): ");
54  rspn = getchar();
55  getchar();
```

```
56          if(rspn == 'y' || rspn == 'Y')
57            strcpy(newval, "ON");
58          else
59            strcpy(newval, "OFF");
60        } while (rspn < 1);
61      wrparm("verbose", newval, fptr);
62      break;
63      }
64    case 'b':
65    case 'B':
66    {
67      printf("\n\n\n");
68      printf("\n\nChanging chromatographic sample time\n");
69      if((getnum(newval)) < 0)
70      {
71        printf("\nChromatographic sample time not changed\n");
72        break;
73      }
74      wrparm("runtime", newval, fptr);
75      break;
76    }
77    case 'c':
78    case 'C':
79    {
80      printf("\n\n\n");
81      for(i = 1; i < 4; i++)
82      {
83        sprintf(name, "stdRT%d", i);
84        printf("\nChanging Retention Time(in minutes) of internal standard #%d", i);
85        if((fnumber = g_fnum(string)) <= 0.0)
86        {
87          printf("\nRetention time of internal standard #%d not changed\n", i);
88          break;
89        }
90        inumber = fnumber * usr.freq * 60.0;
91        if (fnumber > usr.runtime)
92        {
93          printf("Invalid Retention Time - Larger than run time\n");
94          break;
95        }
96        sprintf(newval, "%d", inumber);
97        wrparm(name, newval, fptr);
98      }
99      break;
100   }
101   case 'd':
102   case 'D':
103   {
104     printf("\n\n\n");
105     for(i = 1; i < 4; i++)
106     {
107       printf("\nChanging AREA of internal standard #%d (for match calculation)", i);
108       sprintf(name, "stdarea%d", i);
109       if((getnum(newval)) <= 0)
110       {
```

```
111                 printf("\nArea of standard #%d not changed\n", i);
112                 break;
113                }
114               wrparm(name, newval, fptr);
115              }
116              break;
117             }
118            case 'e':
119            case 'E':
120            {
121             printf("\n\n\n");
122             printf("\n\nChanging minimum area required for standard\n");
123             if((getnum(newval)) <= 0)
124             {
125              printf("\nMinimum area required of standards not changed\n");
126              break;
127             }
128             wrparm("minarea", newval, fptr);
129             break;
130            }
131            case 't':
132            case 'T':
133            {
134             chgtgt();
135             break;
136            }
137            case 'x':
138            case 'X':
139            {
140             fclose(fptr);
141             return;
142             break;
143            }
144            default:
145            {
146             printf("\n\tInvalid selection\n");
147             break;
148            }
149           }
150          }
151         }
152
153
```

```
/*********************************************************************/
/* column.c                                                          */
/* Calculates and prints column factor.  Monitoring this factor gives  */
/* an indication of the condition of the column.  The value decreases  */
/* as the column deteriorates.                                       */
/* created 8/28/86      B. Lentz                                     */
/*                                                                   */
/* Calling Modules:  collect.c                                       */
/* Modules Called:   none                                            */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>

column(prnptr, ymax, tarea, stindx, nstd)
FILE *prnptr;           /* used to send data to printer */
int ymax[];             /* peak heights */
long tarea[];           /* selected areas that correspond to stds & target */
int stindx[];           /* array index to standard peaks */
int nstd;               /* number of internal standards */
{
int i;
float cfactor;       /* column factor */

fprintf(prnptr, "\n\tColumn factors:");
for(i = 1; i <= nstd; i++)
{
cfactor = (float) ymax[stindx[i]]/tarea[i];
fprintf(prnptr, "\t%.4f", cfactor);
}
}
```

```
/*********************************************************************/
/* copsys.c                                                          */
/* format a new diskette and copy operating system and               */
/* datasystem files to it                                            */
/* created 7/21/86   B.Lentz                                         */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#define  SPACE ' '

main()
{
int ntime[4];
int ndate[4];
char string[100];

printf("\n\n\nThis utility creates a new system operating disk\n");
printf("These disks should be replaced monthly\n");
printf("\n\nInsert a NEW diskette into Drive B\nReady (y or n): ");
while(!reply())
  printf("New Diskette in Drive B? (y or n): ");
printf("\nPlease Wait...");
system("type response | format b:/s > NUL");
system("copy a:*.* b: > NUL");
system("del b:*.dat > NUL");
sysdate(ndate, ntime);
sprintf(string, "copy a:QC%d%d.dat b:", ndate[0], ndate[2]);
system(string);
printf("\n\nNew system diskette is ready for use in Drive A\n");
printf("\nRemove diskette from Drive B and place a label on it\n");
printf("Place QC archive diskette in Drive B\n\nReady (y or n): ");
while(!reply())
  printf("QC Archive Diskette in Drive B? (y or n): ");
printf("\nPlease Wait...");
system("copy a:*.dat b: ");
printf("\nRemove QC archive diskette from Drive B and place data diskette in Drive B\n");
printf("\nReady to continue? (y or n): ");
while(!reply())
  printf("Ready to continue? (y or n): ");
}
```

```
/*****************************************************************/
/* error.c                                                       */
/* print error messages                                          */
/* created 5/9/86    B. Lentz                                    */
/*****************************************************************/

#include <stdio.h>

error(code)
int code;
{
switch(code)
{
 case 0:
 {
  fprintf(stderr, "\n\nProblem with chromatography- Standard peaks not found\n");
  break;
 }
 case 1:
 default:
 {
  fprintf(stderr, "\n\nData System Error\n");
  break;
 }
}
}
```

```
1   /*********************************************************************/
2   /* filelist.c                                                        */
3   /* get list of data files on floppy disk                             */
4   /* created 7/1/86    B.Lentz                                         */
5   /*                                                                   */
6   /*********************************************************************/
7
8   #include <stdio.h>
9   #define TRUE   (0==0)
10  #define FALSE  !TRUE
11
12  filelist()
13  {
14  extern char *filedir();       /* returns list of matching filenames */
15  char *list;             /* list of filenames */
16  char *next;
17  char filespec[20];      /* file specification */
18  char filename[20];
19  int count;              /* filename count used to divide output in thirds */
20  int length;
21  FILE *prnptr;
22  int reply();            /* get yes or no reply from user */
23
24  printf("\n\nInsert Data diskette into Drive B\nReady? (y or n): ");
25  while(!reply())
26    printf("Data Diskette in Drive B? (y or n): ");
27  printf("\n\n\tGathering filenames from disk B\n\tPlease Wait\n");
28  system("ls -m b:*.*");
29  printf("\nDo you wish to have a printout of the available files? (y or n): ");
30  if(reply())
31  {
32   if((prnptr = fopen("PRN:", "w")) == 0)
33   {
34    fprintf(stderr, "\nError transmitting data to printer \n");
35    return;
36   }
37   strcpy(filespec, "b:*.*");
38   if((list = filedir(filespec, 0)) == NULL)
39   {
40    printf("\nThere are no data files archived on the floppy disk\n");
41    return;
42   }
43   count = 0;
44   for(next = list; *next != NULL;)
45   {
46    fprintf(prnptr, "%s\t\t\t", next);
47    next += strlen(next) + 1;
48
49    if(++count > 1)
50    {
51     fprintf(prnptr, "\n");
52     count = 0;
53    }
54   }
55   fclose(prnptr);
```

```
56      free(list);
57    }
58    printf("\nENTER filename to retrieve: ");
59    fgets(filename, 19, stdin);
60    while(strcmp(filename, "\n") != 0)
61    {
62      length = strlen(filename);
63      filename[length-1] = '\0';
64      retriev(filename);
65      printf("\nENTER filename to retrieve: ");
66      fgets(filename, 19, stdin);
67    }
68  }
69
```

```
/*********************************************************************/
/* fixseq.c                                                          */
/* correct any errors made during sequence entry(entering sample     */
/* numbers and positions for the purge and trap)                     */
/* created 4/4/86    B. Lentz   revised 6/11/86                      */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

fixseq(frstpos, lastpos)
int frstpos;      /* first position that can be changed */
int lastpos;      /* last sample position on autosampler to be used */
{
extern struct value usr;
extern char label[];  /* date and time information used in name */
int pos;          /* sample position on purge and trap */
int type;         /* sample type */
int num;          /* temporary storage for int */
char rspns;       /* first letter of user response to query */
char string[20];  /* temporary string storage */
char filename[10];

do
{
 system("cls");
 strcpy(filename, "CON:");
 prnseq(filename);
 printf("\n\n\t CHANGE ANY SAMPLE NUMBERS? (y or n):  ");
 if(rspns = reply())
 {
  do {
   printf("\n\tEnter POSITION NUMBER of sample to be changed:  ");
   pos = select();

  }  while(pos < frstpos || pos > lastpos);
  pos--;
  if(usr.type[pos] == SAMPLE)
  {
   sprintf(string, "del b:%s", usr.number[pos]);
   system(string);
  }
  printf("\n\t\t\tSample Type Numbers\n\t   0) BLANK\t\t\t4) WATER SAMPLE\n");
  printf("\t   1) STANDARD #A\t\t5) SPIKE\n");
  printf("\t   2) STANDARD #B\n\t   3) STANDARD #C\n\n");
  printf("\tPosition %d\tENTER sample TYPE number: ", pos+1);
  type = select();
  while(type < 0 || type > 5)
  {
   printf("\t\tENTER sample TYPE number: ", pos+1);
   type = select();
  }
  samptype(&pos, type);
 }
```

```
56  └─} while(rspns);
57   strcpy(filename, "PRN:");
58   prnseq(filename);
59   return;
60  └─}
61
```

```
1   /*********************************************************************/
2   /* fmtdis.c                                                          */
3   /* format a new diskette for data collection                         */
4   /* created 8/8/86   B.Lentz                                          */
5   /*                                                                   */
6   /*********************************************************************/
7
8   #include <stdio.h>
9
10  main()
11  {
12  printf("\n\n***WARNING*** All data on diskette in drive B will be lost\n");
13  printf("\n\nInsert a new diskette into Drive B\nReady? (y or n): ");
14  while(!reply())
15    printf("New Diskette in Drive B? (y or n): ");
16  printf("\nPlease Wait...");
17  system("type response | format b: > NUL");
18  printf("\n\nNew data diskette is ready for use in Drive B\n");
19  printf("\nReady to continue? (y or n): ");
20  while(!reply())
21    printf("Ready to continue? (y or n): ");
22  }
23
```

```
/*********************************************************************/
/* get current system date and time                                  */
/* created 8/7/85                revised                             */
/* B. Lentz                                                          */
/*********************************************************************/

getdate(date, time)
int date[];
int time[];
{
struct regval { int ax,bx,cx,dx,si,di,ds,es; } srv;
int ndat[4];

srv.ax = 0x2a00;
sysint21(&srv,&srv);
date[2] = srv.cx;          /* year */
ndat[1] = srv.dx;

srv.ax = 0x2c00;
sysint21(&srv,&srv);
ndat[2] = srv.cx;

date[0] = ndat[1] >> 8;    /* month */
date[1] = ndat[1] & 0xff;  /* day */
time[0] = ndat[2] >> 8;    /* hour */
time[1] = ndat[2] & 0xff;  /* minutes */
}
```

```
1   /*************************************************************/
2   /* getname.c                                                 */
3   /* check for valid filename - alphanumerics only             */
4   /* created 8/13/86   B. Lentz                                */
5   /*************************************************************/
6
7   #include <stdio.h>
8
9   getname(filename)
10  char *filename;
11  {
12  char *string, *ptr;
13  char c;
14  FILE *fptr;
15  extern char *calloc();
16
17  string = calloc(20, 1);
18  ptr = string;
19  if(!(gets(filename, 16)))
20    printf("\n\t\tERROR reading sample number:\t %s ", filename);
21  strcpy(string, filename);
22
23  while((c = *string) != NULL)
24  {
25   if(isalnum(c) == 0)
26   {
27    printf("\n\t\t\tInvalid character in %s\n", filename);
28    printf("\t\t\tRE-ENTER sample IDENTIFICATION: ");
29    free(ptr);
30    return(1);
31   }
32   string++;
33  }
34  string = ptr;
35
36  strcat(filename, ".dat");
37  sprintf(string, "b:%s", filename);
38  if((fptr = fopen(string, "r")) != 0)
39  {
40   fclose(fptr);
41   printf("\n\tThis filename already exists on disk, please select another name\n");
42   printf("\t\t\tRE-ENTER sample IDENTIFICATION:  );
43   free(ptr);
44   return(1);
45  }
46  else
47  {
48   if((fptr = fopen(string, "w")) == 0)
49   {
50        /* out of space on floppy */
51    free(ptr);
52    return(2);
53   }
54   fclose(fptr);
55   free(ptr);
```

```
56 │ │ return(0);
57 │ └─}
58 └─}
59
```

```
/*********************************************************************/
/* getnum.c                                                          */
/* getnum.c    Prompt user for numerical input and then verify it.   */
/* If valid string is found, returns length of number string,        */
/* otherwise returns zero.                                           */
/* created 8/15/85            B. Lentz                               */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>

long getnum(newval)
char *newval;
{
int i;                  /* counter for loop */
long num;               /* numerical value of parameter */
long atoi();
char string[20];        /* input string containing new value */

printf("\n\tEnter new value:  ");
if(fgets(string, 20, stdin) == 0)
  printf("\nerror in input\n");
sscanf(string, "%s", newval);
num = atoi(newval);
do
{
  if(*newval < '0' || *newval > '9')
    if(*newval != '.')
    {
      fprintf(stderr, "Invalid input - enter numerical value \n");
      return(-1);
    }
}   while(*(++newval) != '\0');
 if(num < 0 || num > 0xFFFF)
   return(-1);
 return(num);
}
```

```
/*********************************************************************/
/* g_fnum.c                                                          */
/* g_fnum.c    Prompt user for floating point numerical input and    */
/* then verify it. If valid string is found, returns length of number */
/* string, otherwise returns zero.                                   */
/* created 6/6/86            B. Lentz                                */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>

double g_fnum(newval)
char *newval;
{
int i;                  /* counter for loop */
float fnum;             /* numerical value of parameter */
double atof();
char string[20];        /* input string containing new value */

printf("\n\tEnter new value:  ");
if(fgets(string, 20, stdin) == 0)
  printf("\nerror in input\n");
sscanf(string, "%s", newval);
fnum = atof(newval);
do
{
 if(*newval < '0' || *newval > '9')
  if(*newval != '.')
  {
   fprintf(stderr, "invalid input - enter numerical value \n");
   return(-1);
  }
}   while(*(++newval) != '\0');
return(fnum);
}

```

```
1   /*********************************************************************/
2   /* g_parm.c                                                          */
3   /* g_parm.c reads text file containing parameters into "usr" structure */
4   /* created 4/1/85    B. Lentz                                        */
5   /*                                                                   */
6   /* modification 3/13/86 added date parameter                         */
7   /* modified 6/12/86  rt's                                            */
8   /*                                                                   */
9   /*********************************************************************/
10  #include <stdio.h>
11  #include "parm.h"
12
13  g_parm(usr, s3)
14  struct value *usr;      /* user value */
15  struct value *s3;       /* S-CUBED */
16
17  {
18  FILE *pfile;          /* file pointer to text file containing parameters */
19  int j;                /* counter */
20  char keyword[16];
21  char string[20];      /* temporary string storage for comparison */
22  char str2[20];        /* temporary string storage for comparison */
23  int badparm;          /* flag for valid parameter */
24
25  if ((pfile = fopen(targfile[0], "r")) == 0)
26  {
27   fprintf(stderr, "\nERROR opening parameters file %s\n", targfile[0]);
28   exit(0);
29  }
30  while((fscanf(pfile, "%s", keyword)) != 0)
31  {
32   if(strcmp(keyword, "runtime") == 0)
33   {
34    fscanf(pfile, "%f %f", &usr->runtime, &s3->runtime);
35   }
36   else if(strcmp(keyword, "caldate") == 0)
37   {
38    fscanf(pfile, "%s %s", &usr->caldate, &s3->caldate);
39   }
40   else if(strcmp(keyword, "date") == 0)
41   {
42    fscanf(pfile, "%s %s", &usr->date, &s3->date);
43   }
44   else if(strcmp(keyword, "calflag") == 0)
45   {
46    fscanf(pfile, "%d %d", &usr->calflag, &s3->calflag);
47   }
48   else if(strcmp(keyword, "spikcnt") == 0)
49   {
50    fscanf(pfile, "%d %d", &usr->spikcnt, &s3->spikcnt);
51   }
52   else if(strcmp(keyword, "verbose") == 0)
53   {
54    fscanf(pfile, "%s %s", keyword, string);
55    usr->verbose = logstr(keyword);
```

```
56        s3->verbose = logstr(string);
57        }
58      else if(strcmp(keyword, "gain") == 0)
59      {
60        fscanf(pfile, "%u %u", &usr->gain, &s3->gain);
61      }
62      else if(strcmp(keyword, "freq") == 0)
63      {
64        fscanf(pfile, "%f %f", &usr->freq, &s3->freq);
65      }
66      else if(strcmp(keyword, "smooth") == 0)
67      {
68        fscanf(pfile, "%d %d", &usr->smooth, &s3->smooth);
69      }
70      else if(strcmp(keyword, "thresh1") == 0)
71      {
72        fscanf(pfile, "%d %d", &usr->thresh[1], &s3->thresh[1]);
73      }
74      else if(strcmp(keyword, "thresh2") == 0)
75      {
76        fscanf(pfile, "%d %d", &usr->thresh[2], &s3->thresh[2]);
77      }
78      else if(strcmp(keyword, "thresh3") == 0)
79      {
80        fscanf(pfile, "%d %d", &usr->thresh[3], &s3->thresh[3]);
81      }
82      else if(strcmp(keyword, "thresh4") == 0)
83      {
84        fscanf(pfile, "%d %d", &usr->thresh[4], &s3->thresh[4]);
85      }
86      else if(strcmp(keyword, "minarea") == 0)
87      {
88        fscanf(pfile, "%d %d", &usr->minarea, &s3->minarea);
89      }
90      else if(strcmp(keyword, "nointstd") == 0)
91      {
92        fscanf(pfile, "%d %d", &usr->nstd, &s3->nstd);
93      }
94      else if(strcmp(keyword, "numunks") == 0)
95      {
96        fscanf(pfile, "%d %d", &usr->nunks, &s3->nunks);
97      }
98      else if(strcmp(keyword, "nlevels") == 0)
99      {
100       fscanf(pfile, "%d %d", &usr->nlevels, &s3->nlevels);
101     }
102     else
103     {
104       badparm = TRUE;
105       for(j = 1; j <= &usr->nstd; j++)
106       {
107         sprintf(string, "stdRT%d", j);
108         sprintf(str2, "stuarea%d", j);
109         if(strcmp(keyword, string) == 0)
110         {
```

```
111          fscanf(pfile, "%d %d", &usr->stdrt[j], &s3->stdrt[j]);
112          badparm = FALSE;
113          break;
114          }
115        else if(strcmp(keyword, str2) == 0)
116        {
117          fscanf(pfile, "%d %d", &usr->stdarea[j], &s3->stdarea[j]);
118          badparm = FALSE;
119          break;
120          }
121        }  /* end for j */
122      if(badparm)
123        fprintf(stderr, "\nParameter %s is not used\n", keyword);
124      }  /* end else */
125    }  /* end while */
126  fclose(pfile);
127  usr->period = 1000000/(usr->freq * 2.5 * 8.0);
128  usr->numpt = 60 * usr->freq * usr->runtime;
129  usr->nconv = usr->numpt * 8;
130  for (j = 1; j <= usr->nstd; j++)
131  {
132    usr->sumarea[j] = 0;
133    usr->sumrt[j] = 0;
134  }
135 }  /* end g_struct */
136
137
138  logstr(string)
139  char *string;
140 {
141  if(strcmp(string, "ON") == 0)
142    return(TRUE);
143  else
144    return(FALSE);
145 }
```

```
1   /*********************************************************************/
2   /* g_sspec.c                                                         */
3   /* g_sspec.c reads text file containing target parameters into       */
4   /* structure "samp"                                                  */
5   /* created 4/1/85  B. Lentz                                          */
6   /*                                                                   */
7   /*********************************************************************/
8   #include <stdio.h>
9   #include "parm.h"
10  #include "targ.h"
11
12  g_sspec(samp)
13  struct sample samp[];
14
15  {
16  FILE *pfile;          /* file pointer to text file containing parameters */
17  int i, j;             /* counter */
18  int badparm;          /* flag incorrect parameter */
19  char keyword[16];     /* name from text file */
20  char string[20];      /* holds parameter name */
21  extern struct value usr;
22
23  for(i = 1; i <= usr.nunks; i++)
24  {
25   if ((pfile = fopen(targfile[i], "r")) == 0)
26   {
27    fprintf(stderr, "\nERROR opening parameters file %s\n", targfile[i]);
28    exit(0);
29   }
30   while((fscanf(pfile, "%s", keyword)) != 0)
31   {
32    if(strcmp(keyword, "rettime") == 0)
33    {
34     fscanf(pfile, "%d %*d", &samp[i].rt);
35    }
36    else if(strcmp(keyword, "name") == 0)
37    {
38     fscanf(pfile, "%s %*s", &samp[i].name);
39    }
40    else if(strcmp(keyword, "units") == 0)
41    {
42     fscanf(pfile, "%s %*s", &samp[i].units);
43    }
44    else if(strcmp(keyword, "spike") == 0)
45    {
46     fscanf(pfile, "%f %*f", &samp[i].spike);
47    }
48    else if(strcmp(keyword, "precisn") == 0)
49    {
50     fscanf(pfile, "%f %*f", &samp[i].precisn);
51    }
52    else if(strcmp(keyword, "accurcy") == 0)
53    {
54     fscanf(pfile, "%f %*f", &samp[i].accurcy);
55    }
```

```
56      else if(strcmp(keyword, "cutoff") == 0)
57      {
58        fscanf(pfile, "%f %*f", &samp[i].cutoff);
59      }
60      else if(strcmp(keyword, "qtlimit") == 0)
61      {
62        fscanf(pfile, "%f %*f", &samp[i].qtlimit);
63      }
64      else if(strcmp(keyword, "dilution") == 0)
65      {
66        fscanf(pfile, "%f %*f", &samp[i].dilution);
67      }
68      else
69      {
70        badparm = TRUE;
71        for(j = 1; j <= usr.nstd; j++)
72        {
73          sprintf(string, "stdconc%d", j);
74          if(strcmp(keyword, string) == 0)
75          {
76            fscanf(pfile, "%f %*f", &samp[i].conc[j]);
77            badparm = FALSE;
78            break;
79          }
80          sprintf(string, "slope%d", j);
81          if(strcmp(keyword, string) == 0)
82          {
83            fscanf(pfile, "%f %*f", &samp[i].slope[j]);
84            badparm = FALSE;
85            break;
86          }
87          sprintf(string, "intercp%d", j);
88          if(strcmp(keyword, string) == 0)
89          {
90            fscanf(pfile, "%f %*f", &samp[i].intercp[j]);
91            badparm = FALSE;
92            break;
93          }
94        }
95        if(badparm)
96          fprintf(stderr, "parameter %s is not used\n", keyword);
97      }
98      }
99      if(fclose(pfile) == -1)
100       fprintf(stderr, "\nERROR closing %s\n", targfile[i]);
101     samp[i].sumurt = 0;
102   }
103 }
104
```

```
/*********************************************************************/
/* initdio.c                                                         */
/* set digital I/O bit 0 of port 0 for output and initialize  to open */
/* relay switch (set to 1)                                           */
/* this state will cause the Tekmar to wait for the switch closure to */
/* begin desorb                                                      */
/* created 2/7/86    B. Lentz                                        */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "bitset.h"     /* bit setting macros for data acquisition board */

initdio()
{
unsigned char status;   /* used for error check */

if ((STAT_REG & 0x70) != 0)
  {
  fprintf(stderr, "\nFATAL ERROR-Illegal status register value\n");
  fprintf(stderr, "\nStatus Register value is %o\n", STAT_REG);
  exit(0);
  }

COMM_REG(CSTOP);
status = DATA_OUT;
while(!(STAT_REG & COMM_WAIT));
COMM_REG(CCLEAR);

while(!(STAT_REG & COMM_WAIT));
COMM_REG(CSOUT);

while(STAT_REG & WRITE_WAIT);
DATA_IN(DIOPORT);

while(!(STAT_REG & COMM_WAIT));
status = STAT_REG;
if(status & 0x80)
  ioerr();

setdio(0, CDIOOUT);
}
```

```
/*********************************************************************/
/* ioerr.c                                                           */
/* read status registers for error reporting from data translation   */
/* board                                                             */
/* created 2/6/86    B.Lentz                                         */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "bitset.h"

ioerr()
{
unsigned char temp;
char error1;              /* first byte of error code */
char error2;              /* second byte of error code */

fprintf(stderr, "\nFATAL BOARD ERROR \n");
fprintf(stderr, "\nStatus Register Value is %x\n", STAT_REG);
   /* read error register */
COMM_REG(CSTOP);
temp = DATA_OUT;
while(!(STAT_REG & COMM_WAIT));
COMM_REG(CERROR);
while(!(STAT_REG & READ_WAIT));
error1 = DATA_OUT;
while(!(STAT_REG & READ_WAIT));
error2 = DATA_OUT;
fprintf(stderr, "Error Register values are: \n");
fprintf(stderr, "byte1: %x \tbyte2: %x", error1, error2);
exit(0);
}
```

```
/*********************************************************************/
/* match.c                                                           */
/* routine to locate standard peaks (internal and target)            */
/* within standard samples and determine actual retention times and  */
/* response factors for each standard.                               */
/* created 7/11/85              B. Lentz                             */
/*                                                                   */
/*********************************************************************/

#include "parm.h"
#include "targ.h"
#include <stdio.h>

float match(npeaks, rawrt, rwarea, finarea, rtime)
int npeaks;          /* total number of peaks found */
int rawrt[];         /* retention times of all peaks */
float rwarea[];      /* areas of all peaks */
long finarea[];      /* areas of selected peaks */
int rtime[];         /* retention times of selected peaks */

{
extern struct sample samp[];  /* target cmp parameters */
extern struct value usr;      /* contains parameters */
int window1;            /* range in which to search for std peak */
int window2;            /* range in which to search for std peak */
int start[2];           /* beginning of search range */
int end[2];             /* end of search range */
int index;              /* index to start of target parameters */
int i, j, k, m;         /* loop counters */
int rrt;                /* relative retention times based on first std */
float factor;           /* ratio of actual retention time to estimate */
float num;              /* numerator of match factor */
double sqrt();          /* square root function */
float fstarea;          /* std area expressed as floating pt/1000 */
float farea;            /* actual area expressed as floating pt/1000 */
float den;              /* denominator of match factor */
float uden;             /* portion of den containing sample contributions */
float rden;             /* reference denominator */
float match;            /* factor indicating quality of match */
float hit;              /* best match factor found */
float totpts;           /* total number of data points collected */
int foundpk;            /* boolean used to flag potential match */
int temp[MAXSTDS];      /* temporary storage for index to peak */

hit = 0.0;
totpts = usr.runtime * usr.freq * 60;
window2 = (int) (totpts * 0.03);   /* 3% window */
window1 = (int) (totpts * 0.40);   /* 40% window */

start[0] = usr.stdrt[1] - window1;
if(start[0] < 0)
  start[0] = 0;
end[0] = usr.stdrt[1] + window1;
for(i = 1; i <= npeaks; i++)
{
```

```
56    if((rawrt[i] > start[0]) && (rawrt[i] < end[0]))
57    {
58      if(rwarea[i] > usr.minarea)
59      {
60        temp[1] = i;
61        factor = (float) rawrt[i]/(usr.stdrt[1]);
62
63        for(k=2; k <= usr.nstd; k++)
64        {
65          temp[k] = 0;
66          rrt = (int) (usr.stdrt[k] * factor);
67          start[1] = rrt - window2;
68          if(start[1] < 0)
69            start[1] = 0;
70          end[1] = rrt + window2;
71
72          for(j = 1; j <= npeaks; j++)
73          {
74            if((rawrt[j] > start[1]) && (rawrt[j] < end[1]))
75            {
76              if(rwarea[j] > usr.minarea)
77                temp[k] = j;
78            }
79          }
80        }
81
82        foundpk = TRUE;
83        for(m = 1; m <= usr.nstd; m++)
84        {
85          if(temp[m] == 0)
86            foundpk = FALSE;
87        }
88        if(foundpk)
89        {
90          num = 0.0;
91          uden = 0.0;
92          rden = 0.0;
93          for(m = 1; m <= usr.nstd; m++)
94          {
95            fstarea = (usr.stdarea[m])/1000.;
96            farea = rwarea[temp[m]]/1000.;
97            num += (fstarea * farea);
98            uden += (farea * farea);
99            rden += (fstarea * fstarea);
100         }
101         if((den = sqrt(uden * rden)) == 0.0)
102         {
103           printf("\n\n ERROR - denominator is zero\n );
104           match = 0.0;
105         }
106         else
107         {
108           match = num/den;
109         }
110         if(match > hit)
```

```
111          {
112            hit = match;
113            for(j = 1; j <= usr.nstd; j++)
114            {
115              finarea[j] = rwarea[temp[j]];
116              rtime[j] = rawrt[temp[j]];
117            }
118            for(k=1; k <= usr.nunks; k++)
119            {
120              rrt = (int) (samp[k].rt * factor);
121              start[1] = rrt - window2;
122              if(start[1] < 0)
123                start[1] = 0;
124              end[1] = rrt + window2;
125              index = usr.nstd + k;
126              finarea[index] = 0;
127              rtime[index] = 0;
128              foundpk = FALSE;
129
130              for(j = 1; j <= npeaks; j++)
131              {
132                if((rawrt[j] > start[1]) && (rawrt[j] < end[1]))
133                {
134                  finarea[index] = rwarea[j];
135                  rtime[index] = rawrt[j];
136                  foundpk = TRUE;
137                }
138              }  /* end for j */
139              if(!foundpk)
140              {
141                finarea[index] = 0;
142                rtime[index] = 0;
143              }
144            }
145          }
146        }
147      }
148    }
149  }
150  return(hit);
151 }
152
153
```

```
 1  /*********************************************************************/
 2  /* menutest.c        User menu                                       */
 3  /* This is the main menu routine to control sample automation on the gc */
 4  /* Created 3/28/85      last revision 4/21/86                        */
 5  /* B. Lentz                                                          */
 6  /*********************************************************************/
 7
 8  #include <stdio.h>
 9  #define TRUE   (0==0)
10  #define FALSE  !TRUE
11
12  main(argc, argv)
13  int argc;
14  char *argv[];
15  {
16  int instrmo;          /* menu selection */
17
18  while(0==0)           /* infinite loop */
19  {
20    system("cls");            /* clear screen */
21    printf("\n\n\n\n\n\n\t\tSelect one of the following options:\n");
22    printf("\n\n\t\t 1) Analyze for TCE \n");
23    printf("\t\t 2) Calibration run for TCE\n");
24    printf("\t\t 3) Prepare new Data diskette(for drive B)\n");
25    printf("\t\t 4) Prepare new VOA Data System diskette(for drive A)\n");
26    printf("\t\t 5) Retrieve QC data\n");
27    printf("\t\t 6) Retrieve archived data\n");
28    printf("\t\t 7) Change operating parameters\n");
29    printf("\t\t 9) Exit Menu\n");
30    printf("\n\n\t ENTER OPTION NUMBER:\t");
31        /* get input from keyboard */
32    instrmo = select();
33    while(instrmo < 1 || instrmo > 9)
34    {
35      fprintf(stderr, "\nInvalid Input.  Enter NUMBER of selected option:\t");
36      instrmo = select();
37    }
38    system("cls");
39    switch(instrmo)
40    {
41      case 1:   /* sample run */
42      {
43        printf("Analyze for TCE\n");
44        system("analyz1 0");
45        break;
46      }  /* end case 2 */
47      case 2:   /* calibration run */
48      {
49        printf("Calibration run for TCE\n");
50        system("analyz1 1");
51        break;
52      }
53      case 3:  /* format data diskette */
54      {
55        printf("Prepare data diskette for use\n");
```

```
56          system("fmtdis");
57          break;
58          }
59        case 4:    /* Format system floppy disk */
60        {
61          printf("\nPrepare new VOA Data System diskette\n");
62          system("copsys");
63          break;
64          }
65        case 5:   /* QC */
66        {
67          printf("Retrieve QC Data\n");
68          system("qcexec");
69          break;
70          }
71        case 6:   /* retrieve data from floppy */
72        {
73          printf("Retrieve archived data from diskette\n");
74          system("xretrv");
75          break;
76          }
77        case 7:    /* Modify operating parameters */
78        {
79          printf("Change operating parameters\n");
80          system("change");
81          break;
82          }
83        case 9:   /* exit to DOS */
84        {
85          printf("\nReturning to operating system\n");
86          printf("Type \"demo\" to return to data analysis system\n");
87          exit(0);
88          break;
89          }
90        default:
91        {
92          printf("\nNot a valid selection\n");
93          break;
94          }
95        }
96      }
97    }
98
```

```
/*********************************************************************/
/* pkstart.c                                                         */
/* Routine called by pk_det.c to determine if a new peak             */
/* has been found.                                                   */
/* created 5/14/85        B. Lentz                                   */
/*                                                                   */
/*********************************************************************/
#define TRUE (0 == 0)
#define FALSE !TRUE

pkstart(np, thr1, thr2)
int np[];
int thr1;
int thr2;
{
int i;

if((np[1] - 2*np[2] + np[3]) > 0)
{
  if((np[1] - 3*np[3] + 2*np[4]) > 0)
    if((np[1] - 4*np[4] + 3*np[5]) > 0)
            /* check for false peak start */
      if((np[5] - np[1]) > thr1)
      {
        for(i = 1; i < 4; i++)
        {
          if(((np[i+2] - np[i+1]) - (np[i+1] - np[i])) >= thr2)
            return(TRUE);
        }
        if(((np[5] - np[4]) - (np[2] - np[1])) >= thr2)
          return(TRUE);
        else
          return(FALSE);
      }  /*end false peak start */
}
return(FALSE);
}
```

```
/*********************************************************************/
/* pk_det.c                                                          */
/* Main section of peak detection software.  Initializes data        */
/* segments and calls subroutines to establish peak start, peak end  */
/* and peak max.                                                     */
/* created 5/14/85        B. Lentz                                   */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include "parm.h"

#define DERIV1(a, b, c)  ((-3*a + 4*b - c)/2)
#define DERIV2(a, b, c)  (a - 2*b + c)

static int xst[NPEAKS];  /* starting x value for peak */
static int yst[NPEAKS];  /* starting y value for peak */
static int xend[NPEAKS]; /* value of x at peak end */
static int yend[NPEAKS]; /* value of y at peak end */
static float baseb;          /* intercept of baseline segment */
static float basem;          /* slope of baseline segment */

pk_det(npts, data, usr, xmax, area)
unsigned npts;           /* number of data points */
unsigned int *data;  /* data array */
int xmax[NPEAKS];    /* value of x at peak max */
float area[NPEAKS];   /* total peak area */
struct value *usr;   /* paramters including threshold values */

{
int i, j;
int np[8];        /* subset of data */
int pos;          /* current x-value of data */
int status;       /* was conditional satisfied? */
int ct;           /* count of times condition is met */
int peak;         /* was peak identified? */
int pkstart();    /* subroutine to locate start of peak */
int integr();     /* integrates peak areas */
int pkcnt;        /* number of peaks detected */
int ymax[NPEAKS]; /* value of y at peak max */
int rd_data();    /* subroutine to read next segment of data */
int pk_end();     /* subroutine to predict peak end */
float TT;         /* distance to predicted peak end */
int end;          /* reached peak end? */
int baseline;     /* current value of baseline */
long base;        /* sum baseline points */
int bcnt;         /* # of pts used to determine baseline */
int bm;           /* distance from peak start to peak max */
int mp;           /* distance from peak max to predicted peak end */

pkcnt = 0;        /* number of peaks in cluster */
npts -= 5;        /* examine 5 point segments */
   /* initialize data segment */
for (i=2; i<6; i++)
{
  np[i] = data[i-2];
```

```
  xst[i] = 0;
  yst[i] = 0;
  xmax[i] = 0;
  ymax[i] = 0;
  xend[i] = 0;
  yend[i] = 0;
  area[i] = 0.0;
 }
pos=0;
j=4;
while(pos<npts)
{
 base = 0;
 bcnt = 0;
 while(!(peak = pkstart(np, usr->thresh[1], usr->thresh[2])))
 {
  pos = rd_data(np, pos, data);
  if(pos > npts)
  {
   if (pkcnt < 1)
    return(pkcnt);
   integr(pkcnt, area, data);
   if(usr->verbose)
    pk_prt(pkcnt, xmax, ymax, area);
   return(pkcnt);
  }
  bcnt++;
  base += np[1];
  baseline = base/bcnt;
 }

 newpk: pkcnt++;
 if(pkcnt > NPEAKS)
 {
  pkcnt--;
  printf("\n exceeded max number of peaks\n");
  integr(pkcnt, area, data);
  if(usr->verbose)
   pk_prt(pkcnt, xmax, ymax, area);
  return(pkcnt);
 }
 area[pkcnt] = (float) np[1];
 yst[pkcnt] = np[1];
 ymax[pkcnt] = np[1];
 xmax[pkcnt] = pos;
 xst[pkcnt] = pos;

 ct = 1;
 status = FALSE;
 do {
  if(DERIV1(np[1], np[2], np[3]) < 0)
  {
   if(DERIV2(np[1], np[2], np[3]) < 1)
   {
    if(usr->verbose)
```

```
111            printf("shoulder at pos %d \n", pos);
112          status = TRUE;
113          if(ct > 1)
114          {
115            ct = pos - ct/2;
116            xmax[pkcnt] = ct;
117            ymax[pkcnt] = data[ct];
118          }
119        }
120      }
121      pos = rd_data(np, pos, data);
122      if(pos > npts)
123      {
124        pkcnt--;
125        integr(pkcnt, area, data);
126        if(usr->verbose)
127          pk_prt(pkcnt, xmax, ymax, area);
128        return(pkcnt);
129      }
130      if(np[1] == ymax[pkcnt])          /* flat area */
131        ct++;
132      else if(np[1] > ymax[pkcnt])
133      {
134        ct = 1;
135        ymax[pkcnt] = np[1];
136        xmax[pkcnt] = pos;
137      }
138      if((np[2] < np[1]) && (np[3] < np[2]) && (np[4] < np[3]) && (np[5] < np[4]))
139      {
140        status = TRUE;
141        if(ct > 1)
142        {
143          ct = pos - ct/2;
144          xmax[pkcnt] = ct;
145          ymax[pkcnt] = data[ct];
146        }
147      }
148      area[pkcnt] += (float) np[1];
149    }  while (!status);
150
151    bm = xmax[pkcnt] - xst[pkcnt];      /* peak start to apex */
152    do {
153      if(DERIV1(np[1], np[2], np[3]) > 0)
154      {
155        if(DERIV2(np[1], np[2], np[3]) > 0)
156        {
157          if(usr->verbose)
158            printf("shoulder at pos %d \n", pos);
159          xend[pkcnt] = pos;
160          yend[pkcnt] = np[1];
161          goto newpk;
162        }
163      }
164      pos = rd_data(np, pos, data);
165      if(pos > npts)
```

```
166        {
167          pkcnt--;
168          integr(pkcnt, area, data);
169          if(usr->verbose)
170            pk_prt(pkcnt, xmax, ymax, area);
171          return(pkcnt);
172        }
173      area[pkcnt] += (float) np[1];
174      if(peak = pkstart(np, usr->thresh[1], usr->thresh[2]))
175        {
176          xend[pkcnt] = pos;
177          yend[pkcnt] = np[1];
178          goto newpk;
179        }
180
181      if((np[4] - np[5]) < usr->thresh[4])
182        end = TRUE;
183      else
184        end = pk_end(np, usr->thresh[3]);
185      if((np[5] - baseline) > (ymax[pkcnt] - baseline)*0.75)
186        end = FALSE;
187      if(np[5] < baseline)
188        end = TRUE;
189      mp = (pos+4) - xmax[pkcnt];
190      if(mp > 2*bm)
191        end = TRUE;
192    } while(!end);
193
194    if(mp*2 < bm)
195      TT = bm*2;
196    else
197      TT = mp*2;
198    end = FALSE;
199    ct = 0;
200    do
201      {
202        if(DERIV1(np[1], np[2], np[3]) > 0)
203          {
204            if(DERIV2(np[1], np[2], np[3]) > 0)
205              {
206                if(usr->verbose)
207                  printf("shoulder at pos %d\n", pos );
208                xend[pkcnt] = pos;
209                yend[pkcnt] = np[1];
210                goto newpk;
211              }
212          }
213        pos = rd_data(np, pos, data);
214        if(pos > npts)
215          {
216            pkcnt--;
217            integr(pkcnt, area, data);
218            if(usr->verbose)
219              pk_prt(pkcnt, xmax, ymax, area);
220            return(pkcnt);
```

```
221        }
222        area[pkcnt] += (float) np[1];
223        if(peak = pkstart(np, usr->thresh[1], usr->thresh[2]))
224        {
225          if(usr->verbose)
226            printf("valley detected at end %d\n", pos);
227          xend[pkcnt] = pos;
228          yend[pkcnt] = np[1];
229          goto newpk;
230        }
231        TT -= 1.0;
232        if((np[4] - np[5] < usr->thresh[4]) && (np[5] < baseline))
233          TT -= 0.5;
234        if (TT <= 0.0)
235        {
236          end = TRUE;
237          if((np[2] < np[1]) && (np[3] < np[2]) && (np[4] < np[3]) && (np[5] < np[4]))
238          {
239            end = FALSE;
240            if(pk_end(np, usr->thresh[3]))
241            {
242              ct++;
243              if(ct >= 4)
244                end = TRUE;
245            }
246            else
247              ct = 0;
248          }
249        }
250
251        if((np[2] >= np[1]) && (np[3] >= np[2]))
252          if((np[4] >= np[3]) && (np[5] >= np[4]))
253            end = TRUE;
254      } while (!end);
255      xend[pkcnt] = pos;
256      yend[pkcnt] = np[1];
257    }
258  }
259
260  /* rd_data reads next segment of data */
261  rd_data(np, pos, data)
262  int np[];
263  int pos;
264  unsigned int *data;
265  {
266  int k;
267
268  for(k=1; k<5; k++)
269    np[k] = np[k+1];
270  np[5] = data[pos+4];
271  pos++;
272  return(pos);
273  }
274
275  /* print peak detection results */
```

```
276
277     pk_prt(pkcnt, xmax, ymax, area)
278     int pkcnt;        /* number of peaks detected */
279     int xmax[];       /* value of x at peak max */
280     int ymax[];       /* value of y at peak max */
281     float area[];       /* total peak area */
282
283     {
284     int k;
285
286     printf("\n\t start\t\t center\t\t end \t\t area \n");
287     for(k=1; k <= pkcnt; k++)
288     {
289       printf(" x %d y %d\t", xst[k], yst[k]);
290       printf(" x %d y %d\t", xmax[k], ymax[k]);
291       printf(" x %d y %d\t", xend[k], yend[k]);
292       printf(" %f\n", area[k]);
293     }
294     }
295
296     /*************************************************************************/
297     /*    integr.c     used to integrate peak areas                          */
298     /* created 5/28/85                                                        */
299     /* calls: segm                                                            */
300     /* B. Lentz                                                               */
301     /*************************************************************************/
302
303     integr(pkcnt, area, data)
304     int pkcnt;          /* number of peaks detected */
305     float area[];        /* total peak area */
306     unsigned int *data;
307
308     {
309     int start;         /* array position in peak cluster */
310     int end;           /* array position at end of cluster */
311     int y;             /* y value from baseline */
312     int pos;           /* current array position in peak cluster */
313     int j;
314     long y1, y2;        /* baseline values at start and end of peak */
315     long sum;          /* area below peak baseline */
316     int segm();
317     extern struct value usr;
318
319     start = 1;
320     end = pkcnt;
321     while(start <= pkcnt)
322     {
323       end = segm(start, end, data);
324       if(usr.verbose)
325         printf("\tsegst %d segend %d\n", xst[start], xend[end]);
326       for(j = start; j <= end; j++)
327       {
328         if((y1 = (xst[j] * basem) + baseb) == 0)
329           y1 = 0;
330         if((y2 = (xend[j] * basem) + baseb) == 0)
```

```
331         y2 = 0;
332       sum = (xend[j] - xst[j] + 1) * ((y1 + y2)/2);
333       area[j] -= (float) sum;
334     }
335     start = end + 1;
336     end = pkcnt;
337   }
338 }
339
340
341 /*************************************************************************/
342 /*    segm.c     recursive routine used to set local baselines          */
343 /* created 6/12/85                                                      */
344 /* B. Lentz                                                             */
345 /*************************************************************************/
346
347 segm(start, end, data)
348 int start;        /* array element to start baseline */
349 int end;          /* array element to end baseline */
350 unsigned int *data;
351
352 {
353 float mck;    /* check slope value */
354 float fx1;
355 float fy1;
356 float fx2;
357 float fy2;
358 int a;         /* current peak position */
359 int y1, y2;    /* values from baseline */
360 int x;         /* current baseline position */
361
362 fx1 = xst[start];
363 fx2 = xend[end];
364 fy1 = yst[start];
365 fy2 = yend[end];
366
367 baseb = ((fx2 * fy1) - (fx1 * fy2))/(fx2 - fx1);
368 basem = (fy1 - baseb)/fx1;
369 mck = (fy2 - baseb)/fx2;
370 for(a = end; a >= start; a--)
371 {
372   y1 = (xst[a] * basem) + baseb;
373   y2 = (xend[a] * basem) + baseb;
374   if((y1 > yst[a]) || (y2 > yend[a]))
375   {
376     a--;
377     if(a <= start)
378     {
379       fx2 = xend[start];
380       fy2 = yend[start];
381       baseb = ((fx2 * fy1) - (fx1 * fy2))/(fx2 - fx1);
382       basem = (fy1 - baseb)/fx1;
383       return(start);
384     }
385     end = segm(start, a, data);
```

```
386        }
387      else if(a > start)
388        if(xst[a] > xend[a-1])     /* baseline between peaks */
389          for(x = xend[a-1]; x < xst[a]; x++)
390          {
391                  /* check for points below chromatographic baseline */
392            y1 = (x*basem) + baseb;
393            if(y1 > data[x])
394            {
395              a--;
396              if(a <= start)
397              {
398                      /* segment only includes one peak */
399                fx2 = xend[start];
400                fy2 = yend[start];
401                baseb = ((fx2 * fy1) - (fx1 * fy2))/(fx2 - fx1);
402                basem = (fy1 - baseb)/fx1;
403                return(start);
404              }
405              end = segm(start, a, data);
406            }
407          }
408      }
409      return(end);
410    }
411
412
413
```

```
/*********************************************************************/
/* pk_end.c                                                          */
/* Predict peak end for pk_det.c                                     */
/* created 5/15/85        B. Lentz                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
#define TRUE (0 == 0)
#define FALSE !TRUE

pk_end(np, thr)
int np[];
int thr;

{
int i;

for(i = 1; i < 4; i++)
  {
  if(((np[i+2] - np[i+1]) - (np[i+1] - np[i])) >= thr)
    return(FALSE);
  }
if(((np[5] - np[4]) - (np[2] - np[1])) >= thr)
  return(FALSE);
else
  return(TRUE);
}
```

```
1   /*************************************************************************/
2   /* prnparm.c                                                             */
3   /* Print current user defined parameters                                 */
4   /* created 8/14/85              B. Lentz                                 */
5   /*                                                                       */
6   /*************************************************************************/
7
8   #include "parm.h"
9
10  prnparm(usr)
11  struct value *usr;
12  {
13  int i;
14  float convrt;        /* convert from array position to minutes */
15
16  system("cls");
17  convrt = usr->freq * 60.0;
18  if(usr->verbose == TRUE)
19    printf("\n\n\na)  Verbose option is ON\n");
20  else
21    printf("\n\n\na)  Verbose option is OFF\n");
22  printf("b)  Chromatography sample time is %.3f minutes\n",
23  usr->runtime);
24  printf("c)  Retention time of internal standards\n");
25  for(i = 1; i <= usr->nstd; i++)
26    printf("\tstandard #%d at %.3f\n", i, (usr->stdrt[i])/convrt);
27  printf("d)  Areas of internal standards (used for match calculation)\n");
28  for(i = 1; i <= usr->nstd; i++)
29    printf("\tstandard #%d at %d\n", i, usr->stdarea[i]);
30  printf("e)  Minimum area required for standards %d\n", usr->minarea);
31  printf("t)  Change target compound parameters\n");
32  printf("x)  Return to main menu\n");
33  }
34
```

```
/*********************************************************************/
/* prnres.c                                                          */
/* send sample results to printer                                    */
/* created 6/25/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

prnres(prnptr, position)
FILE *prnptr;
int position;
{
extern struct value usr;   /* run parameters */

fprintf(prnptr, "\n\n\n");
fprintf(prnptr, "\n\tSAMPLE POSITION #%d\n", position+1);
switch(usr.type[position])
{
 case BLANK:
 {
  fprintf(prnptr, "\n\tBlank ");
  break;
 }
 case STANDRD1:
 {
  fprintf(prnptr, "\n\tStandard#A ");
  break;
 }
 case STANDRD2:
 {
  fprintf(prnptr, "\n\tStandard#B ");
  break;
 }
 case STANDRD3:
 {
  fprintf(prnptr, "\n\tStandard#C ");
  break;
 }
 case SAMPLE:
 {
  fprintf(prnptr, "\n\tWater Sample # %s ", usr.number[position]);
  break;
 }
 case SPIKE:
 {
  fprintf(prnptr, "\n\tSpike ");
  break;
 }
 case SPIKDUP:
 {
  fprintf(prnptr, "\n\tDuplicate Spike ");
  break;
 }
```

```
56      default:
57      {
58        fprintf(prnptr, "\n\tUnknown_Type ");
59        break;
60      }
61    } /* end switch */
62    fprintf(prnptr, "\n\tAnalyst %s", usr.analyst);
63    fprintf(prnptr, "\n\tDate %s\t\tTime %s\n", usr.date, usr.time);
64    fflush(prnptr);
65  }
```

```
/*********************************************************************/
/* prnseq.c                                                          */
/* print sample sequence to screen or printer                        */
/* created 7/24/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

prnseq(filename)
char *filename;      /* device name for redirection of output */
{
extern struct value usr;
extern char label[];  /* date and time information used in name */
int pos;          /* sample position on purge and trap */
FILE *fptr;

if((fptr = fopen(filename, "w")) == 0)
{
fprintf(stderr, "\nError sending information to output device\n");
return;
}
fprintf(fptr, "\n\n\n\t\tASSIGNED SAMPLE NUMBERS");
fprintf(fptr, "\n\tSamples MUST be in the order designed \n\n");
for(pos = 0; pos < usr.sampcnt; pos++)
{
fprintf(fptr, "\n\tPosition %d\t", pos+1);
switch(usr.type[pos])
{
case BLANK:
{
sprintf(usr.number[pos], "%d%s.blk", pos, label);
fprintf(fptr, "BLANK\n");
break;
}
case STANDRD1:
{
sprintf(usr.number[pos], "%d%s.stA", pos, label);
fprintf(fptr, "STANDARD A\n");
break;
}
case STANDRD2:
{
sprintf(usr.number[pos], "%d%s.stB", pos, label);
fprintf(fptr, "STANDARD B\n");
break;
}
case STANDRD3:
{
sprintf(usr.number[pos], "%d%s.stC", pos, label);
fprintf(fptr, "STANDARD C\n");
break;
}
case SAMPLE:
```

```
56          {
57            fprintf(fptr, "WATER SAMPLE\t %s\n", usr.number[pos]);
58            break;
59          }
60        case SPIKE:
61          {
62            sprintf(usr.number[pos], "%d%s.spk", pos, label);
63            fprintf(fptr, "SPIKE of sample %s\n", usr.number[pos-1]);
64            break;
65          }
66        case SPIKDUP:
67          {
68            sprintf(usr.number[pos], "%d%s.dsp", pos, label);
69            fprintf(fptr, "Duplicate SPIKE of sample %s\n", usr.number[pos-2]);
70            break;
71          }
72        default:
73          {
74            fprintf(fptr, "Unasigned position\n");
75            break;
76          }
77      }
78    }
79  fclose(fptr);
80 }
81
```

```
1    /*********************************************************************/
2    /* prntype.c                                                         */
3    /* print sample types available to the screen                        */
4    /* created 8/13/86 B. Lentz                                          */
5    /*                                                                   */
6    /*********************************************************************/
7
8    #include <stdio.h>
9
10   prntype()
11   {
12   printf("\n\t\t\tSAMPLE TYPE NUMBERS\n\n\t   0) BLANK\t\t\t4) WATER SAMPLE\n");
13   printf("\t   1) STANDARD #A\t\t5) SPIKE\n");
14   printf("\t   2) STANDARD #B\t\t6) END OF SAMPLES\n\t   3) STANDARD #C\n\n");
15   }
```

```
/*********************************************************************/
/* qcexec.c                                                          */
/* calls QC data retrieval routines                                  */
/* created 7/18/86   B. Lentz                                        */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>

main()
{
printf("\n\nInsert QC Archive diskette into Drive B\nReady? (y or n): ");
while(!reply())
  printf("QC Archive Diskette in Drive B? (y or n): ");
printf("\nUpdating QC Archive Diskette.  Please wait...");
system("copy a:qc????.dat b: > NUL");
printf("\n\nThis program will retrieve QC spike concentrations \n");
printf("from the month and year selected\n");
printf("date format month/year (eg. 7/86)\n");
do
{
  qcretrv();
  printf("\nDo you wish to retrieve more QC data? (y or n): ");
} while(reply());
printf("\n\nRemove QC Archive diskette from Drive B\nReady? (y or n): ");
while(!reply())
  printf("QC Archive Diskette removed from Drive B? (y or n): ");
}
```

```
1   /*********************************************************************/
2   /* calib.c                                                           */
3   /* calib.c     Used to calibrate chromatographic response based on the  */
4   /* ratio of the response of the target compound to the standard.  All  */
5   /* three standards are used in the final concentration calculations in  */
6   /* order to improve the results (the three values are averaged).      */
7   /* created 7/24/85                                                   */
8   /*                                                                   */
9   /* revised 6/13/86  set calibration date                             */
10  /* B. Lentz                                                          */
11  /*********************************************************************/
12
13  #include <stdio.h>
14  #include "parm.h"
15  #include "targ.h"
16  #define CORR   0.90  /* minimum required correlation */
17
18  calib(unknwn)
19  int unknwn;
20  {
21  extern struct sample samp[]; /* pointer to target structure */
22  extern struct value usr;   /* set parameters */
23  int i, j, k;               /* counters for  loops */
24  float xmean;               /* mean of x values */
25  float ymean;               /* mean of y values */
26  float xresid;              /* x residuals */
27  float yresid[MAXSTDS];     /* y residuals */
28  float ratio[MAXSTDS];      /* area ratios */
29  float num;                 /* numerator */
30  float den;                 /* denominator */
31  float ycorr;               /* sum of squares of y residuals */
32  float var1;                /* intermediate value of corr coef */
33  float r;                   /* correlation coefficient */
34  float intercp;             /* calculated intercept */
35  float slope;               /* calculated slope */
36  double sqrt();             /* square root function */
37  int gudcalib;              /* flag good calibration */
38
39  ymean = 0.0;
40  gudcalib = TRUE;
41  for(k = 1; k <= usr.nlevels; k++)
42    ymean += samp[unknwn].conc[k];
43  ymean = (float) ymean/(usr.nlevels);
44  for(k = 1; k <= usr.nlevels; k++)
45  {
46    yresid[k] = (float) samp[unknwn].conc[k] - ymean;
47    ycorr += yresid[k] * yresid[k];
48  }
49  for (j = 1; j <= usr.nstd; j++)
50  {
51    xmean = 0.0;
52    for(k = 1; k <= usr.nlevels; k++)
53    {
54      ratio[k] = (float) samp[unknwn].area[k]/usr.sarea[k][j];
55      xmean += ratio[k];
```

```
    }
    xmean = xmean/(usr.nlevels);
    num = 0.0;
    den = 0.0:
    for(k = 1; k <= usr.nlevels; k++)
    {
      xresid = ratio[k] - xmean;
      num += xresid * yresid[k];
      den += xresid * xresid;
    }
    slope = num/den;
    intercp = ymean - slope * xmean;
    varl = sqrt(den/ycorr);
    r = varl * slope;
    if(usr.verbose)
    {
      printf("\n correlation coeficient = %f\n", r);
      printf("slope %f\tintercept %f\n", slope, intercp);
    }
    if(r < CORR)
    {
      gudcalib = FALSE;
    }
    else
    {
      samp[unknwn].slope[j] = slope;
      samp[unknwn].intercp[j] = intercp;
    }
  }
  return(gudcalib);
}
```

```
1   /*********************************************************************/
2   /* qcretrv.c                                                         */
3   /* retrieve spike data from floppy disk files                        */
4   /* created 7/17/86   B. Lentz                                        */
5   /*                                                                   */
6   /* printed                                                           */
7   /*********************************************************************/
8
9   #include <stdio.h>
10  #include "smonth.h"
11
12  qcretrv()
13  {
14  int flag;           /* code to indicate type of response(valid, exit) */
15  char filename[30];  /* name of qc file to access */
16  int sday;           /* starting day of month to access data */
17  int eday;           /* end day of month to access data */
18  int day;            /* day of month retrieved from file */
19  FILE *fptr;         /* archived file */
20  FILE *prnptr;       /* output to printer */
21  float conc;         /* concentration of spike sample */
22  int month;
23  int year;
24
25  do
26  {
27    printf("\n\tEnter month/year (or x to exit): ");
28    if((flag = readdate(filename, &month, &year)) == 0)
29      printf("\nIllegal date\tRe-enter month/year:");
30    if(flag == -1)
31      return;
32  }  while(!valfile(filename));
33  do
34  {
35    printf("\nEnter first analysis date to view (day of month): ");
36    while((sday = readday()) == 0);
37    printf("\nEnter last analysis date to view (day of month): ");
38    while((eday = readday()) == 0);
39  }  while(sday > eday);
40
41  if((fptr = fopen(filename, "r")) == 0)
42    printf("\n Error opening qc file\n");
43  else
44  {
45      /* search for starting date */
46    do
47    {
48      if(fscanf(fptr, "%d %f", &day, &conc) == EOF)
49      {
50        printf("\nNo QC data was recorded between %d and %d\n", sday, eday);
51        fclose(fptr);
52        return;
53      }
54    }  while(day < sday);
55    if((prnptr = fopen("PRN:", "wb")) == 0)
```

```
    {
    fprintf(stderr, "\nError transmitting data to printer file \n");
    return;
    }
  if(day > eday)
    {
    printf("\nNo QC data was recorded between %d and %d\n", sday, eday);
    fclose(fptr);
    fclose(prnptr);
    return;
    }
  fprintf(prnptr, "\nQC spike results from %s %d\n", smonth[month], year);
  fprintf(prnptr, "\tDate \tConc\n");
  do
    {
    fprintf(prnptr, "\t%d \t%.3f\n", day, conc);
    if(fscanf(fptr, "%d %f", &day, &conc) == EOF)
      break;
    }  while(day < (eday+1));
  fclose(fptr);
  fclose(prnptr);
  }
}
```

```
/*********************************************************************/
/* qcwrite.c                                                         */
/* opens archive file for writing spike concentrations for QC analysis  */
/* qc filename format: qc<month><year>.dat                           */
/* created 6/11/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

qcwrite(conc)
float conc;        /* concentration of spiked sample */
{
extern struct value usr;
char filename[20];       /* name of QC archive file */
FILE *fptr, *fopen();
int day;                 /* day sample was collected */
int month;
int year;

sscanf(usr.date, "%2d%*c%2d%*c%2d", &month, &day, &year);
sprintf(filename, "a:QC%d%d.dat", month, year);

if((fptr = fopen(filename, "a")) == 0)
{
fprintf(stderr, "ERROR opening control archive file\n");
return;
}
fprintf(fptr, "%d %f ", day, conc);
fclose(fptr);
}

```

```
1   /*********************************************************************/
2   /* readdate.c                                                        */
3   /* wait for user to enter in date string and check for valid entry   */
4   /* string should be in format: month/year                            */
5   /* created 7/18/86      B. Lentz                                     */
6   /*                                                                   */
7   /*********************************************************************/
8
9   #include <stdio.h>
10
11  readdate(filename, month, year)
12  char *filename;          /* name of qc file */
13  int *month;
14  int *year;
15  {
16  char string[20];     /* temporary string storage */
17
18  fgets(string, 15, stdin);
19  if(strncmp(string, "x", 1) == 0)
20    return(-1);    /* exit */
21  sscanf(string, "%2d%*c%2d", month, year);
22  if(*month < 13 && *month > 0)
23  {
24   sprintf(filename, "b:QC%d%d.dat", *month, *year);
25   return(1);
26  }
27  return(0);
28  }
29
```

```
1   /*********************************************************************/
2   /* readday.c                                                         */
3   /* wait for user input and check for valid day                       */
4   /* created 7/18/86   B. Lentz                                        */
5   /*                                                                   */
6   /*********************************************************************/
7
8   #include <stdio.h>
9
10  readday()
11  {
12  char string[15];
13  int day;
14
15  fgets(string, 15, stdin);
16  if(strncmp(string, "x", 1) == 0)
17    return(-1);    /* exit */
18  sscanf(string, "%2d", &day);
19  if(day > 0 && day < 32)
20    return(day);
21  else
22  {
23  printf("\nIllegal value for date\n");
24  return(0);
25  }
26  }
27
```

```
1   /*********************************************************************/
2   /* reply.c                                                           */
3   /* get yes or no reply from user                                     */
4   /* returns true on yes and false on no                               */
5   /* created 7/2/86    B.Lentz                                         */
6   /*                                                                   */
7   /*********************************************************************/
8
9   #define YES 1
10  #define NO  0
11  #define EXIT -1
12
13  reply()
14  {
15  char string[6];
16  char rspns;          /* first letter of response */
17  int value;           /* return value */
18
19  if(!(gets(string, 5)));
20  sscanf(string, "%c", &rspns);
21  if((rspns == 'y') || (rspns == 'Y'))
22    return(YES);
23  else if((rspns == 'x') || (rspns == 'X'))
24    return(EXIT);
25  else if((rspns == 'n') || (rspns == 'N'))
26    return(NO);
27  else
28  {
29  printf("\n\tIllegal response.  Enter 'y' or 'n': ");
30  value = reply();
31  return(value);
32  }
33  }
34
```

```
/*********************************************************************/
/* resflpy.c                                                         */
/* store results in floppy disk file                                 */
/* created 6/27/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "targ.h"

resflpy(code, position, targ, value1, value2)
int code;               /* type of results to report */
int position;           /* sample position */
int targ;               /* index of target compound */
float value1, value2;   /* concentration results */
{
extern struct value usr;
extern struct sample samp[];  /* parameters for target cmpds */
FILE *flppy;
char filename[20];      /* floppy disk filename */

sprintf(filename, "b:%s", usr.number[position]);
if((flppy = fopen(filename, "a")) == 0)
{
 fprintf(stderr, "\nError Opening %s\n", filename);
 return;
}
switch(code)
{
 case 1:     /* report concentration at quantitation limit */
 {
  fprintf(flppy, " %s < %.3f %s", samp[targ].name, samp[targ].qtlimit, samp[targ].units);
  break;
 }
 case 2:     /* sample result */
 {
  fprintf(flppy, " %s = %.3f %s", samp[targ].name, value1, samp[targ].units);
  break;
 }
 case 3:     /* spike result */
 {
  fprintf(flppy, " %s %.3f %.3f %s", samp[targ].name, value1, value2, samp[targ].units);
  break;
 }
 case 4:
 {
  fprintf(flppy, " Recalibration required");
  break;
 }
 case 5:     /* standard calibration sample */
 {
  fprintf(flppy, " Calibration");
  break;
 }
```

```
56      case 6:     /* contaminated blank */
57      {
58        fprintf(flppy, " BLANK CONTAMINATED for %s analysis", samp[targ].name);
59        break;
60      }
61      case 7:     /* chromatography problem, < 3 peaks */
62      {
63        fprintf(flppy, " %s = %.3f ", samp[targ].name, value1);
64        fprintf(flppy, " x Missing Internal Standards");
65        break;
66      }
67      case 8:     /* sample too complex */
68      {
69        fprintf(flppy, " %s = %.3f ", samp[targ].name, value1);
70        fprintf(flppy, " x Sample too complex for automated analysis ");
71        break;
72      }
73      default:
74        break;
75    }
76    fclose(flppy);
77  }
78
```

```
/*********************************************************************/
/* retriev.c                                                         */
/* retrieves information from data files on floppy disk              */
/* created 6/30/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#define  MINMATCH 0.7

retriev(filename)
char *filename;          /* name of data file */
{
FILE *prnptr;          /* send contents to printer */
FILE *stptr;           /* read values from archived file */
char analyst[10];      /* initials of analyst */
char date[10];         /* date of analysis */
char time[10];         /* time of analysis */
char type[20];         /* sample type(blank, std, etc) */
int npeaks;            /* number of peaks found in sample */
float rt;              /* retention time in minutes */
long area;             /* peak area */
int i;
char string[30];       /* character string */
char equil;            /* value is "<" or "=" relationship */
float value1, value2;  /* floating point values */
int ntargs;            /* number of target compounds */
char units[10];        /* units of concentration */

if((prnptr = fopen("PRN:", "w")) == 0)
{
 fprintf(stderr, "\nError transmitting data to printer \n");
 return;
}
fprintf(prnptr, "\n\n\t%s\n", filename);
sprintf(string, "b:%s", filename);
if((stptr = fopen(string, "r")) == 0)
{
 fprintf(stderr, "\nError Opening %s\n", filename);
 return;
}
fscanf(stptr, "%s %s %s ", analyst, date, time);
fprintf(prnptr, "\n\tAnalyst is %s\n\tDate %s\n\tTime %s\n", analyst, date, time);
fscanf(stptr, "%s %d", type, &npeaks);
fprintf(prnptr, "\tSample type %s\n", type);
fprintf(prnptr, "\n     Peak #\t    retention time\tarea\n");
for(i = 0; i < npeaks; i++)
{
 fscanf(stptr, "%f %D", &rt, &area);
 fprintf(prnptr, "\t%d\t\t%.3f\t\t%D\n", i+1, rt, area);
}
fscanf(stptr, "%f %d", &value1, &ntargs);
fprintf(prnptr, "\n\tMatch factor is %.3f\n", value1);
for(i = 0; i < ntargs; i++)
{
```

```
 56     if(strcmp("Blank", type) == 0)
 57     {
 58       break;
 59     }
 60     else if(strncmp("Standard", type, 8) == 0)
 61     {
 62       fscanf(stptr, "%s", string);
 63       if(strncmp(string, "calibration", 6) == 0)
 64       {
 65         fprintf(prnptr, "\tCalibration run\n");
 66       }
 67       else
 68       {
 69         fscanf(stptr, "%f %s", &value1, units);
 70         fprintf(prnptr, "\tConcentration of %s is %.3f %s\n", string, value1, units);
 71       }
 72       break;
 73     }
 74     else if(strcmp("Sample", type) == 0)
 75     {
 76       fscanf(stptr, "%s %is %f %s", string, &equil, &value1, units);
 77       if(value1 == -1.0)
 78         i = ntargs;
 79       else
 80         fprintf(prnptr, "\tConcentration of %s %c %.3f %s\n", string, equil, value1, units);
 81       break;
 82     }
 83     else if(strncmp("Spik", type, 4) == 0)
 84     {
 85       fscanf(stptr, "%s %f %f %s", string, &value1, &value2, units);
 86       fprintf(prnptr, "\tConcentration of %s is %.3f %s\n", string, value1, units);
 87       fprintf(prnptr, "\tAmount of spike %.3f %s\n", value2, units);
 88       break;
 89     }
 90   }
 91   fprintf(prnptr, "\n");
 92   while(fscanf(stptr, "%s", string) != EOF)
 93     fprintf(prnptr, "%s ", string);
 94   fprintf(prnptr, "\n\n\n\n");
 95   fclose(stptr);
 96   fclose(prnptr);
 97 }
```

```
/*********************************************************************/
/* sampler.c                                                         */
/* From sample type(std, blank, sample), performs                    */
/* necessary calculations and sends results to appropriate location. */
/* created 7/24/85                B. Lentz                           */
/*                                                                   */
/* revised 5/21/86 added spike duplicate                             */
/* revised 6/13/86 included quantitation limit                       */
/* revised 8/21/86 added dilution limit                              */
/* revised 9/8/86 added factor to indicate system response.  Actually */
/*    ratio of actual std area to value set in table                 */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "targ.h"

sampler(prmptr, position, rettm, area, calrun)
FILE *prmptr;        /* output to printer */
int position;        /* sample position */
int rettm[];         /* selected retention times for std & target peaks */
long area[];         /* selected areas for std & target peaks */
int *calrun;         /* calibration run */
{
extern struct value usr;      /* run parameters */
extern struct sample samp[];  /* parameters for target cmpds */
int i, j;
int index;                /* index value to retrieve target info */
float ratio;              /* ratio of target to int std */
float conct[MAXUNKS];     /* concentration of unknown */
float sumconc;            /* sum of conc used for averaging */
float spikamt;            /* amount of target due to spike */
float diff;               /* difference between expected and experimental */
float factor;             /* indication of system response */
float stddev();           /* calculates std dev and returns mean */
float quant();            /* calculate sample concentration */

switch(usr.type[position])
{
 case BLANK:
 {
      /* check for contamination */
  for(j = 1; j <= usr.nunks; j++)
  {
   conct[j] = quant(j, area);
        /* print out results */
   if(conct[j] >= samp[j].qtlimit)
   {
        /* contaminated blank */
    fprintf(prmptr, "\n\t*** Contaminated blank ***\n");
    fprintf(stderr, "\n\t*** Contaminated blank ***\n");
    resflpy(6, position, j, conct[j], 0.0);
    exit(0);
   }
  }
```

```
56          break;
57        }
58      case STANDRD1:
59      case STANDRD2:
60      case STANDRD3:
61      {
62        if(*calrun)
63        {
64          fprintf(prnptr, "\n\tCalibration run\n");
65          for(i = 1; i <= usr.nstd; i++)
66          {
67            usr.sumrt[i] += rettm[i];
68            usr.sumarea[i] += area[i];
69            usr.sarea[usr.type[position]][i] = area[i];
70          }
71          for(i = 1; i <= usr.nunks; i++)
72          {
73            index = usr.nstd + i;
74            samp[i].sumurt += rettm[index];
75            samp[i].area[usr.type[position]] = area[index];
76          }
77          if(position == 3)
78          {
79            if(usr.invalid)
80            {
81              fprintf(prnptr, "\nRecalibration is REQUIRED\n");
82              fprintf(stderr, "\nRecalibration is REQUIRED\n");
83              break;
84            }
85            for(j = 1; j <= usr.nstd; j++)
86            {
87              usr.stdrt[j] = usr.sumrt[j]/usr.nlevels;
88              usr.stdarea[j] = usr.sumarea[j]/usr.nlevels;
89                      /* calculate system response factor */
90              factor = (float) area[j]/usr.stdarea[j];
91              fprintf(prnptr, "\tColumn factor %d  %f\n", j, factor);
92            }
93            for(j = 1; j <= usr.nunks; j++)
94            {
95              samp[j].rt = samp[j].sumurt/(usr.nlevels);
96              if(calib(j))
97              {
98                stocalib(j);
99                printf("\nSuccessful Calibration for %s\n", samp[j].name);
100               fprintf(prnptr, "\n\tCalibration successful for %s\n", samp[j].name);
101             }
102             else
103             {
104               fprintf(prnptr, "\n\t*** WARNING *** Calibration UNSUCCESSFUL for %s\n", samp[j].name);
105               fprintf(stderr, "\n\t*** WARNING *** Calibration UNSUCCESSFUL for %s\n", samp[j].name);
106             }
107           }
108
109           *calrun = 0;
110         }
```

```
111          resflpy(5, position, 1, 0.0, 0.0);
112        }
113      else
114      {
115        for(j = 1; j <= usr.nunks; j++)
116        {
117          conct[j] = quant(j, area);
118          if(conct[j] < samp[j].qtlimit)
119          {
120            fprintf(prnptr, "\n\tConcentration of %s is less than %.3f %s\n", samp[j].name, samp[j].qtlimit, samp[j].
               units);
121            resflpy(1, position, j, 0.0, 0.0);
122          }
123          else
124          {
125            fprintf(prnptr, "\n\tConcentration of %s is %.3f %s", samp[j].name, conct[j], samp[j].units);
126            resflpy(2, position, j, conct[j], 0.0);
127          }
128          if((diff = conct[j] - samp[j].conc[usr.type[position]]) < 0.0)
129            diff = -diff;
130          if(usr.verbose)
131            printf("\naccuracy diff %f resid %f\n", diff, diff-samp[j].accurcy);
132          if((diff - samp[j].accurcy) > 0.0)
133          {
134            setcalib();
135            fprintf(prnptr, "\n*** WARNING *** System out of calibration, recalibration is required\n");
136            fprintf(stderr, "\n*** WARNING *** System out of calibration, recalibration is required\n");
137          }
138        }
139      }
140      break;
141    }
142    case SAMPLE:
143    {
144      for(j = 1; j <= usr.nunks; j++)
145      {
146        conct[j] = quant(j, area);
147        samp[j].lastconc = conct[j];
148        if(conct[j] < samp[j].qtlimit)
149        {
150          fprintf(prnptr, "\n\tConcentration of %s is less than %.3f %s\n", samp[j].name, samp[j].qtlimit, samp[j].
               units);
151          resflpy(1, position, j, 0.0, 0.0);
152        }
153        else
154        {
155          fprintf(prnptr, "\n\tConcentration of %s is %.3f %s", samp[j].name, conct[j], samp[j].units);
156          resflpy(2, position, j, conct[j], 0.0);
157          if(conct[j] > samp[j].cutoff)
158          {
159            fprintf(prnptr, "\nWARNING sample concentration exceeds maximum allowable water limits\n");
160            if(conct[j] > samp[j].dilution)
161            {
162              fprintf(prnptr, "\nWARNING sample results are beyond the linear range of the instrument\n");
163              fprintf(prnptr, "For accurate results, dilute sample and run again\n");
```

```
164                 }
165                 }
166               }
167             }
168             if(usr.calflag)
169             {
170               fprintf(prnptr, "\n*** WARNING *** These results are ONLY estimates\n");
171               fprintf(prnptr, "\nRecalibration is required for valid results\n");
172               resflpy(4, position, 1, 0.0, 0.0);
173             }
174             break;
175           }
176           case SPIKDUP:
177           case SPIKE:
178           {
179             for(j = 1; j <= usr.nunks; j++)
180             {
181               conct[j] = quant(j, area);
182               if((spikamt = conct[j] - samp[j].lastconc) < 0.0)
183                 spikamt = 0.0;
184               fprintf(prnptr, "\n\tConcentration of %s is %.3f %s", samp[j].name, conct[j], samp[j].units);
185               fprintf(prnptr, "\n\tspiked amount of %s is %.3f %s", samp[j].name, spikamt, samp[j].units);
186               resflpy(3, position, j, conct[j], spikamt);
187               qcwrite(spikamt);
188               if(usr.type[position] == SPIKDUP)
189               {
190                 if((diff = conct[j] - samp[j].frstspke) < 0)
191                   diff = -diff;
192                 if((diff - samp[j].precisn) > 0)
193                 {
194                   fprintf(prnptr, "\n*** WARNING *** poor reproducibility, recalibrate\n");
195                   setcalib();
196                 }
197               }  /* end SPIKDUP */
198               if((diff = spikamt - samp[j].spike) < 0)
199                 diff = -diff;
200               if(diff - samp[j].accurcy > 0)
201               {
202                 fprintf(prnptr, "\n*** WARNING *** poor sample recovery, recalibrate\n");
203                 fprintf(stderr, "\n*** WARNING *** poor sample recovery, recalibrate\n");
204                 setcalib();
205               }
206               samp[j].frstspke = conct[j];
207             }
208             break;
209           }
210           default:
211           {
212             fprintf(stderr, "\nIllegal sample type\n");
213             break;
214           }
215         }
216       }
217
218     /* quantitation step */
```

```
219    float quant(j, area)
220    int j;            /* target cmpd index */
221    long area[];      /* selected peak areas */
222    {
223    extern struct value usr;
224    extern struct sample samp[];
225    float mean;             /* sample concentration */
226    int i;
227    int index;              /* index value to retrieve target info */
228    float ratio;            /* ratio of target to int std */
229    float conct[MAXUNKS];   /* concentration of unknown */
230    float sumconc;          /* sum of conc used for averaging */
231
232    sumconc = 0.0;
233    for(i = 1; i <= usr.nstd; i++)
234    {
235      index = usr.nstd + j;
236      ratio = (float) area[index]/area[i];
237      if((conct[i] = ((samp[j].slope[i]) * ratio) + samp[j].intercp[i]) < 0.0)
238        conct[i] = 0.0;
239      sumconc += conct[i];
240      if(usr.verbose)
241        printf("unknown %d \tconc %f\n", j, conct[i]);
242    }
243    mean = stddev(usr.nstd, sumconc, conct, usr.verbose);
244    return(mean);
245    }
246
247
```

```
1   /*********************************************************************/
2   /* samptype.c                                                        */
3   /* sets sample numbers for all sample types                          */
4   /* created 4/1/86       B. Lentz                                     */
5   /*                                                                   */
6   /* modified 5/21/86                                                  */
7   /*********************************************************************/
8
9   #include <stdio.h>
10  #include "parm.h"
11
12  samptype(pos, type)
13  int *pos;          /* position number */
14  int type;          /* sample type */
15  {
16  extern struct value usr;
17  char filename[15];
18  int code;              /* error code from getname */
19  char string[100];
20  FILE *fptr;
21  int i;
22
23  switch(type)
24  {
25   case SAMPLE:
26   {
27    usr.type[*pos] = type;
28    printf("\t\t\tENTER sample IDENTIFICATION: ");
29    while((code = getname(filename)) != 0)
30    {
31     if(code == 2)
32     {
33            /* out of space on floppy */
34            /* delete files created on current disk */
35      for(i = 1; i < *pos; i++)
36      {
37       if(usr.type[i] == SAMPLE)
38       {
39        sprintf(string, "del b:%s > NUL", usr.number[i]);
40        system(string);
41       }
42      }
43      printf("\n\t\tThere is no space left on the data diskette in Drive B\n");
44      printf("\t\tPlease prepare a new data diskette\n");
45      system("pause");
46      system("fmtdis");
47      system("cls");
48      prntype();
49      printf("\n\tPosition %d\tRE-ENTER sample IDENTIFICATION: ", (*pos)+1);
50     }
51    }
52    strcpy(usr.number[*pos], filename);
53    usr.spikcnt++;
54    (*pos)++;
55    if((usr.spikcnt > 9) && (*pos < 8))
```

```
56          {
57            type = SPIKE;
58            samptype(pos, type);
59          }
60        break;
61      }
62      case SPIKE:
63      {
64        if(usr.type[(*pos)-1] == SAMPLE)
65        {
66          usr.type[*pos] = type;
67          printf("\tPosition %d\tSPIKE of sample %s\n", (*pos)+1, usr.number[(*pos)-1]);
68          if(*pos < 9)
69          {
70            (*pos)++;
71            usr.type[*pos] = SPIKDUP;
72            printf("\tPosition %d\tDuplicate SPIKE of sample %s\n", (*pos)+1, usr.number[(*pos)-2]);
73          }
74          (*pos)++;
75          usr.spikcnt = 0;
76          break;
77        }
78        else
79        {
80          fprintf(stderr, "\tSpike MUST follow sample\n");
81          return;
82          break;
83        }
84      }
85      case END:
86      {
87        usr.sampcnt = *pos;
88        *pos = 11;
89        break;
90      }
91      default:
92      {
93        usr.type[*pos] = type;
94        (*pos)++;
95        break;
96      }
97    }
98    return;
99  }
100
```

```
1    /*********************************************************************/
2    /* select.c                                                          */
3    /* make a numerical menu selection                                   */
4    /* created 7/10/86      B. Lentz                                     */
5    /*                                                                   */
6    /*********************************************************************/
7
8    #include <stdio.h>
9    #include <ctype.h>
10
11   select()
12   {
13   char string[20];
14   char c;
15   long atoi();
16
17   while(!(gets(string, 20)));
18   sscanf(string, "%c", &c);
19   if(isdigit(c))
20     return(atoi(string));
21   else
22     return(-1);
23   }
24
```

```
/*********************************************************************/
/* seqenc.c                                                          */
/* Prompts user for sample numbers, colletion date/time and          */
/* analyst.  It indicates which positions are to be filled with      */
/* standards, blanks, and samples. This information is stored in a   */
/* structure which is passed to the other routines.                  */
/* created 4/17/85            B. Lentz                               */
/*                                                                   */
/* modified 3/31/86  to allow more flexibity in loading seqence      */
/* modified 4/30/86  add spike samples every 10th sample             */
/* modified 6/10/86                                                  */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

seqenc(frstrun)
int frstrun;                /* is this the first run of the day? */
{.
extern struct value usr;
FILE * fptr;
int pos;                /* physical sample position */
int type;               /* sample type */
char string[20];        /* temporary string storage */
int nochg;              /* number of positions that cannot be changed */

printf("\nEnter analyst identification : ");
while(!(gets(usr.analyst, 10)))
  fprintf(stderr, "\nENTER ANALYST IDENTIFICATION: ");
printf("Enter information as requested below:\n");
if(fflush(stdin) < 0)
  printf("\nbuffer not cleared\n");
usr.sampcnt = 10;       /* number of samples in run */
prntype();      /* print sample types */
pos = 0;
usr.type[pos] = BLANK;
printf("\tPosition %d\tBLANK\n", pos+1);
pos++;


nochg = 2;
if(frstrun)
{
  usr.type[pos] = STANDRD1;
  printf("\tPosition %d\tSTANDARD#A\n",pos+1);
  pos++;
  nochg = 3;
}
while(pos < 10)
{
  printf("\tPosition %d\tENTER sample TYPE number: ", pos+1);
  type = select();
  while(type < 0 || type > 6)
  {
```

```
56          printf("\t\t\tENTER sample TYPE number: ", pos+1);
57          type = select();
58        }
59        samptype(&pos, type);
60      }
61      if((fptr = fopen(targfile[0], "rw")) == 0)
62      {
63        fprintf(stderr, "\nERROR opening parameter file\n");
64        return;
65      }
66      itoa(usr.spikcnt, string);
67      wrparm("spikcnt", string, fptr);
68      fclose(fptr);
69
70      fixseq(nochg, usr.sampcnt);
71      return;
72    }
73
```

```
/*********************************************************************/
/* setcalib.c                                                        */
/* set switches to require recalibration on the next run             */
/* created 6/24/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

setcalib()
{
extern struct value usr;
FILE *fptr;

   /* set switch for calibration run */
if((fptr = fopen(targfile[0], "rw")) == 0)
{
 fprintf(stderr, "error opening parameter file\n");
 return;
}
usr.calflag = TRUE;       /* calibration required */
wrparm("calflag", "1", fptr);
fclose(fptr);
}
```

```
/*********************************************************************/
/* setdate.c                                                         */
/* sets date and obtains operator id for run                         */
/* created 4/4/86    B. Lentz                                        */
/*                                                                   */
/* modified 6/6/86                                                   */
/* modified 7/3/86                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

char label[15];         /* part of std identification with date & time */

setdate()
{
extern struct value usr;
int ntime[4];
int ndate[4];
FILE *fptr;

sysdate(ndate, ntime);
sprintf(usr.date, "%2d/%02d/%2d", ndate[0], ndate[1], ndate[2]);
if((fptr = fopen(targfile[0], "rw")) == 0)
{
fprintf(stderr, "error opening parameter file\n");
return;
}
wrparm("date", usr.date, fptr);
fclose(fptr);
sprintf(usr.time, "%2d:%02d", ntime[0], ntime[1]);
sprintf(label, "%02d%02d%02d", ndate[0], ndate[1], ntime[0]);
}
```

```
/*********************************************************************/
/* setdio.c                                                          */
/* sends signal to Tekmar to control desorb start(contact closure)   */
/* open relay switch (set to 1),or set to 0 to close relay switch.   */
/* this state will cause the Tekmar to wait for the switch closure to */
/* begin desorb                                                      */
/* created 2/7/86    B. Lentz                                        */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "bitset.h"     /* bit setting macros for data acquisition board */

setdio(value, port)
unsigned char value;  /* digital output */
unsigned char port;   /* set digital port (with or w/o trigger) */
{
unsigned char status;   /* used for error check */

if ((STAT_REG & 0x70) != 0)
{
  fprintf(stderr, "\nFATAL ERROR-Illegal status register value\n");
  fprintf(stderr, "\nStatus Register value is %o\n", STAT_REG);
  exit(0);
}

COMM_REG(CSTOP);
status = DATA_OUT;
while(!(STAT_REG & COMM_WAIT));
COMM_REG(CCLEAR);

while(!(STAT_REG & COMM_WAIT));
COMM_REG(port);

while(STAT_REG & WRITE_WAIT);
DATA_IN(DIOPORT);

while(STAT_REG & WRITE_WAIT);
DATA_IN(value);

while(!(STAT_REG & COMM_WAIT));
status = STAT_REG;
if(status & 0x80)
  ioerr();
}
```

```
/*********************************************************************/
/* stddev.c                                                          */
/* Calculate std deviation of concentrations and throw out           */
/* outliers.  Acceptance criteria - must be within 2 SD.             */
/* created 8/8/85              B. Lentz                              */
/*                                                                   */
/*********************************************************************/
#define  SD 2     /* value must be within 2 std dev */

float stddev(nvalues, sum, conc, verbose)
int nvalues;         /* number of concentration values */
float sum;           /* sum of concentration values */
float conc[];        /* concentration values */
int verbose;

{
float mean;       /* average concentration */
int i, k;         /* counters for loop */
float resid;      /* residuals */
float sqresid;    /* sum of squares of residuals */
float stdev;      /* standard deviation */
float start;      /* start of window for acceptable conc values */
float end;        /* end of window for acceptable conc values */
double sqrt();    /* square root function */

begin:
mean = sum/((float) nvalues);
sqresid = 0.0;
for(i = 1; i <= nvalues; i++)
  {
  resid = conc[i] - mean;
  sqresid += resid * resid;
  }
stdev = sqrt(sqresid/(nvalues - 1));
start = mean - (SD * stdev);
end = mean + (SD * stdev);
for(i = 1; i <= nvalues; i++)
  {
  if(conc[i] < start || conc[i] > end)
    {
    if(verbose)
      printf("outlier %f\n", conc[i]);
    sum = 0;
    for(k = 1; k < i; k++)
      sum += conc[k];
    for(k = i; k < nvalues; k++)
      {
      conc[k] = conc[k+1];
      sum += conc[k];
      }
    if(--nvalues < 3)
      return(mean);
    goto begin;
    }
  }
```

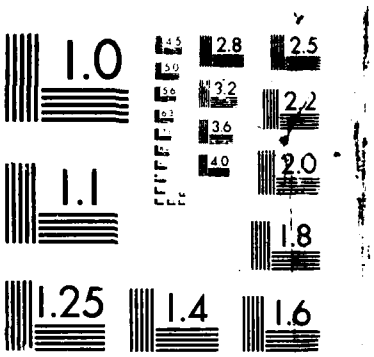
```
56 │  return(mean);
57 └─}
58
```

```
/*********************************************************************/
/* stdseq.c                                                          */
/* standard seqence is the mandatory sample seqence for the calibration */
/* run.                                                              */
/* created 3/13/86      B. Lentz                                     */
/*                                                                   */
/* modified 5/20/86                                                  */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "targ.h"

stdseq()
{
extern struct value usr;
extern struct sample samp[];
FILE *fptr;
int pos;        /* sample position on purge and trap */
int type;       /* sample type */
int i;
char string[20];

printf("\nEnter analyst identification :  ");
while(!(gets(usr.analyst, 10)))
  fprintf(stderr, "\nENTER ANALYST IDENTIFICATION  ");
printf("Enter information as requested below for the calibration run:\n");
   /* clear out input buffer */
if(fflush(stdin) < 0)
  printf("\nbuffer not cleared\n");
usr.sampcnt = 10;       /* number of samples in run */

prntype();
usr.type[0] = BLANK;
printf("\tPosition 1\tBLANK\n");
usr.type[1] = STANDRD1;
printf("\tPosition 2\tSTANDARD#A\n");
usr.type[2] = STANDRD2;
printf("\tPosition 3\tSTANDARD#B\n");
usr.type[3] = STANDRD3;
printf("\tPosition 4\tSTANDARD#C\n");

for(pos = 4; pos < 10;)
{
  printf("\tPosition %d\tENTER sample TYPE number: ", pos+1);
  type = select();
  while(type < 0 || type > 6)
  {
    printf("\t\t\tENTER sample TYPE number:  , pos+1);
    type = select();
  }
  samptype(&pos, type);
}
 if((fptr = fopen(targfile[0], "rw")) == 0)
 {
```

```
56      fprintf(stderr, "\nERROR opening parameter file\n");
57      exit(0);
58    }
59    itoa(usr.spikcnt, string);
60    wrparm("spikcnt", string, fptr);
61    fclose(fptr);
62
63    fixseq(5, usr.sampcnt);
64    for(i = 1; i <= usr.nstd; i++)
65    {
66      usr.sumrt[i] = 0;
67      usr.sumarea[i] = 0;
68    }
69    for(i = 1; i <= usr.nunks; i++)
70      samp[i].sumurt = 0;
71    return;
72  }
73
```

AD-A195 101 PROTOTYPE TECHNOLOGY FOR MONITORING VOLATILE ORGANICS 2/2
VOLUME 2(U) S-CUBED LA JOLLA CA V TAYLOR ET AL. MAR 88
SSS-R-87-8515-VOL-2 ESL-TR-88-01-VOL-2 F08635-84-C-0290

UNCLASSIFIED F/G 7/3 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963

```
/*********************************************************************/
/* stocalib.c                                                        */
/* change values in "permanent" parameter file for slope and intercept */
/* and change calibration flag to indicate that calibration was      */
/* successful                                                        */
/* created 8/6/86    B. Lentz                                        */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"
#include "targ.h"

stocalib(unknwn)
int unknwn;
{
extern struct sample samp[]; /* pointer to target structure */
extern struct value usr;   /* set parameters */
int j;
FILE *fptr;
char string[50];           /* string representation of number */
char parmname[20];         /* parameter name */

if((fptr = fopen(targfile[unknwn], "rw")) == 0)
{
fprintf(stderr, "\nERROR opening parameter file %s\n", targfile[unknwn]);
return;
}
for (j = 1; j <= usr.nstd; j++)
{
ftoa(samp[unknwn].slope[j], string, 4, 4);
sprintf(parmname, "slope%d", j);
wrparm(parmname, string, fptr);
ftoa(samp[unknwn].intercp[j], string, 4, 4);
sprintf(parmname, "intercp%d", j);
wrparm(parmname, string, fptr);
}
fclose(fptr);

if((fptr = fopen(targfile[0], "rw")) == 0)
{
fprintf(stderr, "\nERROR opening parameter file %s\n", targfile[0]);
return;
}
usr.calflag = FALSE;
wrparm("calflag", "0", fptr);
strcpy(usr.caldate, usr.date);
strcpy(string, usr.caldate);
wrparm("caldate", string, fptr);
fclose(fptr);
}
```

```
/*********************************************************************/
/* stoflpy.c                                                         */
/* store sample info, retention time and areas on floppy disk "b"    */
/* created 6/13/86      B. Lentz                                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

stoflpy(position, npeaks, rettm, area, hit)
int position;                /* sample position */
int npeaks;                  /* number of peaks found in sample */
int rettm[MAXSTDS];          /* sample retention times */
float area[MAXSTDS];         /* sample areas */
float hit;                   /* match factor */
{
extern struct value usr;   /* run parameters */
FILE *stptr;               /* to send data for archiving on floppy */
char filename[20];         /* filename on floppy */
char strtyp[20];           /* string containing sample type */
float rt;                  /* retention time in minutes */
int i;

switch(usr.type[position])
{
 case BLANK:
 {
  strcpy(strtyp, "Blank ");
  break;
 }
 case STANDRD1:
 {
  strcpy(strtyp, "Standard#A ");
  break;
 }
 case STANDRD2:
 {
  strcpy(strtyp, "Standard#B ");
  break;
 }
 case STANDRD3:
 {
  strcpy(strtyp, "Standard#C ");
  break;
 }
 case SAMPLE:
 {
  strcpy(strtyp, "Sample ");
  break;
 }
 case SPIKE:
 {
  strcpy(strtyp, "Spike ");
  break;
```

```
    }
    case SPIKDUP:
    {
      strcpy(strtyp, "Spike_Duplicate ");
      break;
    }
    default:
    {
      strcpy(strtyp, "Unknown_Type ");
      break;
    }
  }  /* end switch */

  sprintf(filename, "b:%s", usr.number[position]);
  if((stptr = fopen(filename, "w")) == 0)
  {
    fprintf(stderr, "\nError Opening %s\n", filename);
    return;
  }
  fprintf(stptr, "%s %s %s ", usr.analyst, usr.date, usr.time);
  fprintf(stptr, "%s ", strtyp);

  fprintf(stptr, "%d ", npeaks);
  for(i = 1; i <= npeaks; i++)
  {
    rt = rettm[i]/(usr.freq * 60.0);
    fprintf(stptr, "%.2f %.0f ", rt, area[i]);
  }
  fprintf(stptr, "%.3f %d ", hit, usr.nunks);
  fclose(stptr);
}

```

```
/*****************************************************************/
/* sysdate.c                                                     */
/* get current system date and time                              */
/* created 8/7/85                B. Lentz                        */
/*                                                               */
/*****************************************************************/

sysdate(date, time)
int date[];
int time[];
{
struct regval { int ax,bx,cx,dx,si,di,ds,es; } srv;
int ndat[4];
int year;          /* 4 digit listing of year */
char string[6];    /* string representation of year */

srv.ax = 0x2a00;
sysint21(&srv,&srv);
year = srv.cx;          /* year */
sprintf(string, "%d", year);
sscanf(string, "%*c%*c%2d", &(date[2]));
ndat[1] = srv.dx;

srv.ax = 0x2c00;
sysint21(&srv,&srv);
ndat[2] = srv.cx;

date[0] = ndat[1] >> 8;     /* month */
date[1] = ndat[1] & 0xff;   /* day */
time[0] = ndat[2] >> 8;     /* hour */
time[1] = ndat[2] & 0xff;   /* minutes */
}
```

```c
/* temp.c        used for testing remaining modules without the need   */
/* to collect data each time                                           */
/* 4/24/86                                                             */
/*********************************************************************/

#include <stdio.h>
#include "parm.h"

temp(position, smprt, areas)
int position;
int smprt[NPEAKS];       /* retention times of sample peaks */
float areas[NPEAKS];      /* areas of sample peaks */
{
extern struct value usr;
FILE *fptr;
int npeaks;     /* number of peaks found in sample */
int j;
float var1;    /* retention time in minutes */

static char *file[] = {
"blank.dat",
"standrd1.dat",
"standrd2.dat",
"standrd3.dat",
"sample.dat",
"sample1.dat",
"sample2.dat",
"sample3.dat",
"sample4.dat",
"sample5.dat"
};

if((fptr = fopen(file[position], "r")) == 0)
{
fprintf(stderr, "\nError opening %s datafile\n", file[position]);
exit(0);
}
fscanf(fptr, "%d ", &npeaks);
for(j = 1; j <= npeaks; j++)
{
fscanf(fptr, "%f ", &var1);
smprt[j] = var1 * 60.0 * usr.freq;
fscanf(fptr, "%f ", &areas[j]);
}
fclose(fptr);
return(npeaks);
}  /* end temp */
```

```
1    #include <stdio.h>
2
3    main()
4    {
5    char buffer[6];
6
7    printf("\n\nInsert a new diskette into Drive B\nPress Enter key when ready...");
8    fgetc(stdin);
9    fflush(stdin);
10   printf("\nPlease Wait...");
11   printf("Enter");
12   fgets(buffer, 5, stdin);
13   fflush(stdin);
14   printf("Enter");
15   fgetc(stdin);
16   }
17
```

```
1   /*********************************************************************/
2   /* umenu.c        User menu                                          */
3   /* This is the main menu routine to control sample automation on the gc */
4   /* Created 3/28/85      B. Lentz                                     */
5   /*                                                                   */
6   /* last revision 4/21/86                                             */
7   /*********************************************************************/
8
9   #include <stdio.h>
10  #define TRUE   (0==0)
11  #define FALSE  !TRUE
12
13  main(argc, argv)
14  int argc;
15  char *argv[];
16  {
17  int instrmo;         /* menu selection */
18
19  while(0==0)          /* infinite loop */
20  {
21   system("cls");            /* clear screen */
22   printf("\n\n\n\n\n\n\t\tSelect one of the following options:\n");
23   printf("\n\n\t\t 1) Analyze for TCE \n");
24   printf("\t\t 2) Calibration run for TCE\n");
25   printf("\t\t 3) Prepare new Data diskette(for drive B)\n");
26   printf("\t\t 4) Prepare new VOA Data System diskette(for drive A)\n");
27   printf("\t\t 5) Retrieve QC data\n");
28   printf("\t\t 6) Retrieve archived data\n");
29   printf("\t\t 7) Change operating parameters\n");
30   printf("\t\t 9) Exit Menu\n");
31   printf("\n\n\t ENTER OPTION NUMBER:\t");
32   instrmo = select();
33   while(instrmo < 1 || instrmo > 9)
34   {
35    fprintf(stderr, "Invalid Input.  Enter NUMBER of selected option:\t");
36    instrmo = select();
37   }
38   system("cls");            /* clear screen */
39   switch(instrmo)
40   {
41    case 1:   /* sample run */
42    {
43     printf("Analyze for TCE\n");
44     system("analyz 0");
45     break;
46    }  /* end case 2 */
47    case 2:   /* calibration run */
48    {
49     printf("Calibration run for TCE\n");
50     system("analyz 1");
51     break;
52    }
53    case 3:  /* format data diskette */
54    {
55     printf("Prepare data diskette for use\n");
```

```
56          system("fmtdis");
57          break;
58          }
59        case 4:    /* Format system floppy disk */
60          {
61          printf("\nPrepare new VOA Data System diskette\n");
62          system("copsys");
63          break;
64          }
65        case 5:   /* QC */
66          {
67          printf("Retrieve QC Data\n");
68          system("qcexec");
69          break;
70          }
71        case 6:   /* retrieve data from floppy */
72          {
73          printf("Retrieve archived data from diskette\n");
74          system("xretrv");
75          break;
76          }
77        case 7:    /* Modify operating parameters */
78          {
79          printf("Change operating parameters\n");
80          system("change");
81          break;
82          }
83        case 9:   /* exit to DOS */
84          {
85          printf("\nReturning to operating system\n");
86          printf("Type \"menu\" to return to data analysis system\n");
87          exit(0);
88          break;
89          }
90        default:
91          {
92          printf("\nNot a valid selection\n");
93          break;
94          }
95        }
96      }
97    }
98
99
```

```
1    /*********************************************************************/
2    /* valfile.c                                                         */
3    /* check for presence of file                                        */
4    /* created 7/18/86   B. Lentz                                        */
5    /*                                                                   */
6    /*********************************************************************/
7
8    #include <stdio.h>
9
10   valfile(filename)
11   char *filename;
12   {
13   FILE *fptr;
14
15   if((fptr = fopen(filename, "r")) == 0)
16   {
17     printf("\nFile %s does not exist\n", filename);
18     return(0);
19   }
20   fclose(fptr);
21   return(1);
22   }
23
```

```
/*****************************************************************/
/* wrparm.c                                                      */
/* Write new value of parameter to "parm.dat"                    */
/* created 8/16/85      B. Lentz                                 */
/*                                                               */
/* modified 6/9/86                                               */
/*****************************************************************/

#include <stdio.h>
#define BASE 0

wrparm(name, newval, fileptr)
char *name;        /* name of parameter to change */
char *newval;      /* new value of parameter */
FILE *fileptr;     /* pointer to file to change */
{
char string[20];      /* read parameter name from  "parm.dat" */
char oldval[20];      /* s3 parameters */
long pos;             /* position in file */

rewind(fileptr);
do
{
  if(fscanf(fileptr, "%s", string) == 0)
  {
    printf("\n%s not found in file\n", string);
    return;
  }
} while(strcmp(string, name) != 0);
pos = ftell(fileptr);
fscanf(fileptr, "%s %s", string, oldval);
if(fseek(fileptr, pos, BASE) == -1)
{
  printf("Address error - improper seek\n");
  return;
}
fprintf(fileptr, "  \t%-15s %-15s", newval, oldval);
if(fflush(fileptr) == -1)
  fprintf(stderr, "ERROR writing to file\n");
}
```

```
1    /*********************************************************************/
2    /* xretrv.c                                                          */
3    /* executive module for retrieving data from floppy disk             */
4    /* created 7/1/86    B. Lentz                                        */
5    /*                                                                   */
6    /*********************************************************************/
7
8    main()
9    {
10   filelist();
11   }
12
```

```
/*********************************************************************/
/* smooth.c                                                          */
/* smooth raw data for further processing using Savitzky-Golay least  */
/* squares fit.  5, 7, 9, 11, 13, 15, 17, 19, 21, 23, and 25 point    */
/* smoothing equations are available. (Anal. Chem. 1964, 36(8) 1627-39 */
/* created 5/1/85    B. Lentz                                        */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>

smooth(data, npts, factor)
int data[];          /* raw data */
unsigned npts;           /* number of raw data points */
int factor;          /* type of smooth: 5 point, 9 point */

{
int spts;            /* number of pts smoothed */
int a[14];           /* array of coefficients */
int norm;            /* normalizing factor */
int ctr;             /* center array position for coefficients */
int i, j, k, x;      /* counters */
long np[26];         /* current data pts being smoothed */
long nsum;           /* intermediate sum of least squares */

switch(factor)
{
 case 5:
 {
  norm = 35;
  a[0] = 17;
  a[1] = 12;
  a[2] = -3;
  ctr = 3;
  break;
 }
 case 7:
 {
  norm = 21;
  a[0] = 7;
  a[1] = 6;
  a[2] = 3;
  a[3] = -2;
  ctr = 4;
  break;
 }
 case 11:
 {
  norm = 429;
  a[0] = 89;
  a[1] = 84;
  a[2] = 69;
  a[3] = 44;
  a[4] = 9;
  a[5] = -36;
```

```
56          ctr = 6;
57          break;
58        }
59        case 13:
60        {
61          norm = 143;
62          a[0] = 25;
63          a[1] = 24;
64          a[2] = 21;
65          a[3] = 16;
66          a[4] = 9;
67          a[5] = 0;
68          a[6] = -11;
69          ctr = 7;
70          break;
71        }
72        case 15:
73        {
74          norm = 1105;
75          a[0] = 167;
76          a[1] = 162;
77          a[2] = 147;
78          a[3] = 122;
79          a[4] = 87;
80          a[5] = 42;
81          a[6] = -13;
82          a[7] = -78;
83          ctr = 8;
84          break;
85        }
86        case 17:
87        {
88          norm = 323;
89          a[0] = 43;
90          a[1] = 42;
91          a[2] = 39;
92          a[3] = 34;
93          a[4] = 27;
94          a[5] = 18;
95          a[6] = 7;
96          a[7] = -6;
97          a[8] = -21;
98          ctr = 9;
99          break;
100       }
101       case 19:
102       {
103         norm = 2261;
104         a[0] = 269;
105         a[1] = 264;
106         a[2] = 249;
107         a[3] = 224;
108         a[4] = 189;
109         a[5] = 144;
110         a[6] = 89;
```

```
111         a[7] = 24;
112         a[8] = -51;
113         a[9] = -136;
114         ctr = 10;
115         break;
116       }
117     case 21:
118       {
119         norm = 3059;
120         a[0] = 329;
121         a[1] = 324;
122         a[2] = 309;
123         a[3] = 284;
124         a[4] = 249;
125         a[5] = 204;
126         a[6] = 149;
127         a[7] = 84;
128         a[8] = 9;
129         a[9] = -76;
130         a[10] = -171;
131         ctr = 11;
132         break;
133       }
134     case 23:
135       {
136         norm = 805;
137         a[0] = 79;
138         a[1] = 78;
139         a[2] = 75;
140         a[3] = 70;
141         a[4] = 63;
142         a[5] = 54;
143         a[6] = 43;
144         a[7] = 30;
145         a[8] = 15;
146         a[9] = -2;
147         a[10] = -21;
148         a[11] = -42;
149         ctr = 12;
150         break;
151       }
152     case 25:
153       {
154         norm = 5175;
155         a[0] = 467;
156         a[1] = 462;
157         a[2] = 447;
158         a[3] = 422;
159         a[4] = 387;
160         a[5] = 342;
161         a[6] = 287;
162         a[7] = 222;
163         a[8] = 147;
164         a[9] = 62;
165         a[10] = -33;
```

```
166         a[11] = -138;
167         a[12] = -253;
168         ctr = 13;
169         break;
170       }
171     case 9:
172     default:
173       {
174         norm = 231;
175         a[0] = 59;
176         a[1] = 54;
177         a[2] = 39;
178         a[3] = 14;
179         a[4] = -21;
180         ctr = 5;
181         factor = 9;
182         break;
183       }
184     }
185   for(i = ctr; i < 13; i++)
186     a[i] = 0;
187   for(i = 2; i <= factor; i++)
188     np[i] = data[i-2];
189   spts = npts - ctr - 1;
190   for(i = ctr; i < spts; i++)
191     {
192       j = i + ctr - 1;
193       for(k = 1; k < factor; k++)
194         np[k] = np[k+1];
195       np[factor] = data[j];
196
197       nsum = a[0] * np[ctr];
198       for(x = 1; x < ctr; x++)
199         nsum += a[x] * (np[ctr+x] + np[ctr-x]);
200       data[i] =  nsum/norm;
201     }
202   return;
203 }
204
205
```

HQ AFESC/RDXI
TYNDALL AFB FL 32403-6001

OFFICIAL BUSINESS

FOURTH CLASS

END
DATE
FILMED
8-88
DTIC