UNITED STATES DEPARTMENT OF INTERIOR

GEOLOGICAL SURVEY

A TELEMETRY-BASED DATA-ACQUISITION SYSTEM

PART 2:

THE DATA-ACQUISITION RECEIVER

by

Gregory K. Miller

Open-File Report 89 -553-B

Woods Hole, Mass.

1989

TABLE OF CONTENTS

INTRODUCTION

The design requirements for this data-acquisition system were established for a U.S. Geological Survey study of coastal erosion on a remote sand island in the Gulf of Mexico. The primary functions of this system are to acquire hourly data concerning sand height, wave conditions, overwash processes, and weather conditions and to provide immediate access to that data through a telephone link. The complete system consists of (1) a remotely sited battery-powered, computer-based data-acquisition package and the equipment to transmit that data to a second computer system using a VHF (very high frequency) radio link. (2) The second computer, located onshore, records the data in permanent storage that is connected to a phone line to allow off-site access to the recorded data.

The basic design was developed in 1986, utilizing off-the-shelf hardware for as much of the system as possible to minimize development time. The data-acquisition portion is described in detail in Gregory K. Miler, 1989, A Telemetry-Based Data-Acquisition System Pat 1: The Remote Data-Acquisition Transmitter: USGS Open File Report 89- . This report describes the second computer, which is used to collect and store the data.

This second system consists of an IBM personal computer(PC), or clone, that has a minimum of 640 Kilobytes of memory and a hard disk for the storage of data. A Tape backup system is attached to this PC to backup any data recorded on the system's hard disk, a printer to record the times of data-acquisition, and a modem to permit remote user access to the data. The operating system for the computer is a commercially available software package that permits two separate programs to run independently. This was necessary to simultaneously provide continuous data-acquisition via the radio link and remote access through the modem. The software package used in this system allows for one of the programs to control the other on command. This allows a user to call into the system, shut down the acquisition software, backup the data, and erase the hard disk used to store the data without visiting the field site.

The tape backup system was set up to operate via the software provided by the manufacturer, allowing the system to utilize any device designed to backup data on a hard disk. Data backup can be accomplished by running the software on site, using a shell program (see listing for BACKUP) that runs the software at selected times of the day, or through the modem communications program.

The modem communications software can be any commercially available package. The software package used for this system will perform data transfers in background mode. This permits a user to retrieve data through the modem link at the same time data are being acquired through the radio link. The communications package also has the capability of running script files, which are programs designed to run within the communication program. With this option, the communication

software will check the time of day and run the tape backup program at the scheduled time.

The main data-acquisition program (TELGET) was written to acquire data received through the radio link. This software has the responsibility of recording any data received through the radio link onto hard disk: it prints out the record time and file name on a printer, and monitors the data quality.

Also included are two programs that do not make up any part of the data-acquisition system. They are used to analyze the data at the field site. TELPLOT is a graphic display program for monitoring data files acquired by the system. TELDUMP is a text display of data files acquired by the system.

DESCRIPTION OF PRIMARY PROGRAMS

BACKUP

This program checks the internal time, maintained by the computer, to determine the time scheduled for backup of data recorded on the hard disk (5 minutes after midnight). At this preselected time, the backup software is run, and its success or failure is displayed on the screen.

TELGET

This program is written to record data acquired through the radio link on disk drive D. The radio link is attached to the computer via the RS232 serial port identified as COM1 by the computer operating system. The functions used to communicate with COM1 are from a standard software package called C ASYNCH MANAGER (Blaise Computing, Berkley, Cal.). The program TELGET simply monitors a data buffer in the computer memory until a data set is completely received through the radio link. After the data are checked to determine the type of data file, and a file label is generated. (The file labeling procedure is dependent on the data format, listed in Gregory K. Miller, 1989, A Telemetry-based Data-Acquisition System, Part 1: The Remote Data-Acquisition Transmitter: USGS Open File Report 89- .) The primary data sets are labeled with the extension ".PRI", and the intermediate data sets are labeled ".INT". If the program is unable to correctly read the data's header information, the file extension is ".DAT". The file name is a number from 1 to 999999. The program checks previous data files to ensure that each data file has a unique number. The data are then written to a file on disk drive D in a directory called "\DATA". If this directory is not found, the program will create it. Once the data have been written to disk, the program prints the file name, its size, and the time data was written to file on the system's printer. This information and the data's header information is also written to disk drive E: in a file called DATA.TXT. This file permits a user accessing the system via the modem link with a means of viewing the progress of the data-acquisition.

TELDUMP

This program displays the data values recorded by the TELGET program. The functions used in this software to create pop-up windows and graphic boxes are a software package called CTOOLS PLUS (Blaise Computing, Berkley, Cal.). The hexadecimal value or voltage value of each channel can be displayed on the display screen, the printer, or a disk file. The user also has the option of creating a plot file. This is used to repair files that have been garbled during the radio transmission.

TELPLOT

This program plots the data values recorded by the TELGET program and requires either an EGA or VGA graphics terminal to run. The functions used in this software to create pop-up windows and graphics are a software package called CTOOLS PLUS (Blaise Computing, Berkley, Cal.). The voltage values of a selected range of data channels is first plotted for the entire sample period. The vertical axis is the voltage from +5 volts to -5 volts; the horizontal axis represents time in seconds. The program can replot this data to any scale by entering the maximum and minimum voltages and the maximum and minimum time values. Any data points not within these windowed limits are not plotted.

BACKUP.C LISTING

```
/*----- THIS PROGRAM BACKS UP THE HARD DISK AT THE SET TIME --*/

/* REV 1.00 code written 14 apr 1987 by gkm  */

#include <stdio.h>
#include <time.h>

int error,minute;
long ltime;
struct tm *newtime;

main()
{

/* THIS LOOP WILL CONTINUELY CHECK THE CLOCK UNTIL THE TIME
   MATCHES THE PRESET TIME */

     for (;;)                                /* Loop until ^B */
          {
          time(&ltime);                      /* Read the clock */
          newtime = localtime(&ltime);
          if(kbhit() != 0)          /* Test for ^B to exit */
               {
               if (getch() == 0x02)
                    exit(0);
               }
          else if ((newtime->tm_hour == 00) && (newtime->tm_min == 05))
               tapeb();
          else
               ;
          }
}

/* ----------------- RUN TBACKUP PROGRAM -------------------- */

tapeb()
{
   error = system("tbackup /m/a/y"); /* Run program */
   if (error < 0) printf("Unable to run tape backup\n");
   if (error > 0) printf ("System command error %d\n",error);
   printf("Last tape backup at %s\n",asctime(newtime));

   /* DELAY PAST CURRENT MINUTE TO PREVENT MULTIPLE BACKUPS  */

   minute = newtime->tm_min;  /* Delay past current minute. */
   while(minute == newtime->tm_min)
           {
           time(&ltime);                     /* Read the clock */
           newtime = localtime(&ltime);
           }
}
```

TELGET.C LISTING

```
/*                      TELGET.H                    */
/* rev L written by gkm 10/5/88 */
#define TEL_TRUE 0xF2      /* True flag */
#define TEL_FALSE 0x81     /* False flag */
#define I_BSIZ 32000       /* Input buffer size is 32000 bytes */
#define O_BSIZ 24          /* Output buffer is 24 bytes */

          /* DATA BUFFER DEFINES */

char *buffer,*malloc();
unsigned buff_siz = 32767;    /* Total buffer size */

          /* BLAISE COMMAND DEFINES */

int lstatus,mstatus,error;    /* Returned status and errors */
int port_no = COM1;       /* Port used is COM1 */
int baud_rate = 4;        /* Baud rate is 1200 */
int parity = 0;           /* No parity */
int stop_bits = 0;        /* 1 stop bit */
int data_bits = 3;        /* 8 data bits */
int int_level = 0;        /* Use default interrupt levels */
int port_addr = 0;        /* Use default port address */
int written;              /* No. bytes sent return word */
int status,iq_size;       /* status and # characters in buffer */
int isize,data;           /* Read command returns */

          /* DATA FILE DEFINES */

FILE *f2,*f1,*fopen();
int samdno = 1;           /* Initial sample no.- default file */
int samlno = 1;           /* Initial sample no.- long file*/
int samsno = 1;           /* Initial sample no.- short file*/
char *ext[50];            /* Default file label */
int label;                /* File extension flag storage */
long length,filelength();    /* File length functions */

          /* TIME OF DAY DEFINES */

int minute;               /* Minutes storage */
long ltime;               /* Pointer to time */
struct tm *newtime;       /* Time structure */

char msg[80];                /* Storage string for general use */
unsigned char header[50];    /* Storage for header values */

/*------------------ FUNCTION DECLARATIONS ----------------*/

extern void main(void);
extern int datack(void);
extern void datacol(void);
extern void timept(void);
extern void lprnt(char *);
extern void shutdown(void);
extern void head_stor(void);
extern int file_ext(void);
```

```
/* TELGET.C - THIS PROGRAMS RECEIVES DATA FROM THE RF MODEM,  */
/*   SAVES IT IN A FILE, PRINTS THE TIME OF RECORDING TO THE   */
/*   PRINTER */

/* --- REV L ---*/

/* code written  5/28/87 by gkm */
/* code modified to eliminate average files 11/18/87 */
/* rev I update 2/17/88 by gkm - program now records the data in
   a single file, buffers changed to fix a pointer error */
/* rev J update 2/25/88 by gkm - data is stored on drive D: in
   subdirectory \DATA.  A file is created on the default drive
   to save all information that goes to the printer */
/* rev K update 2/25/88 by gkm - the header information is read
   from the data and saved in the directory file on drive E: */
/* rev L update by gkm 10/04/88 - if data file opening fails,
   the program will attempt to create the \data directory */

#include "asynch_1.h"      /* Header file is Blaise Product  */
#include <stdio.h>
#include <time.h>
#include "telget.h"


void main()
{

     /* --- ALLOCATE BUFFER SPACE --- */

     if((buffer = malloc(buff_siz)) == NULL)
          {
          puts("No memory available");
          exit(0);
          }

     /* --- PRINT START TIME OF PROGRAM --- */

     lprnt("Program started at \0");
     timept();

     /* --- INITIALIZE AND OPEN THE COM1 RS232 PORT --- */

     /* SET COM 1 TO 1200 BAUD, NO PARITY 1 STOP BIT, 8 DATA BITS */

     init_a1(port_no,baud_rate,parity,stop_bits,data_bits,&lstatus,
          &mstatus);
```

```
/* --- OPEN PORT AS COM 1, EXIT IF ERROR --- */

error = open_a1(port_no,I_BSIZ,O_BSIZ,int_level,port_addr,buffer);
if (error == NULL)
     puts("Port is successfully opened.");
else
     {
     printf("Port open error # %d.\n",error);
     exit(1);
     }

/* --- LOOP FOR THE DATA --- */

iflsh_a1(port_no);      /* Flush input buffer*/

for (;;)
     {
     while(datack() > 37)
          {

          /* --- WRITE DATA INTO A TEMPORARY FILE --- */

          if((f1 = fopen("D:\\DATA\\TEMP.DAT\0","wb")) != NULL)
               {
               datacol();
               iflsh_a1(port_no);

               /* DELAY ONE MINUTE TO BE SURE FILE IS COMPLETE */

              time(&ltime);
               newtime = localtime(&ltime);
               minute = newtime->tm_min +2;
               while(minute > newtime->tm_min)
                    {
                    time(&ltime);
                    newtime = localtime(&ltime);
                    qsize_a1(port_no,&iq_size,&status);

                    /* --- IF MORE DATA, COLLECT IT --- */

                    if(iq_size >5)
                         {
                         datack();
                         datacol();
                         time(&ltime);
                         newtime = localtime(&ltime);
                         minute = newtime->tm_min +2;
                         iflsh_a1(port_no);
                         }
                    }

          /* --- CLOSE THE TEMPORARY DATA FILE --- */

          fclose(f1);
          }
```

```
/* CREATE THE DATA DIRECTORY IF FILE CREATION FAILED */

else
     {
     mkdir("D:\\DATA");
     puts("Can't write data to file!");
     }

/* --- LABEL THE DATA FILE --- */

if((f1 = fopen("D:\\DATA\\TEMP.DAT\0","r")) != NULL);
     {

     /* --- GET DATA FILE INFORMATION --- */

     fread((char *)header,1,50,f1);
     length = filelength(fileno(f1));
     fclose(f1);

     /* --- LABEL PRIMARY DATA FILE --- */

     if(file_ext()== TEL_TRUE)
          {
          sprintf(ext,"D:\\DATA\\%d.PRI\0",samlno);

          /*--- DO NOT OVERWRITE EXISTING DATA --- */

          while((f1 = fopen(ext,"r+")) != NULL)
               {
               fclose (f1);
               samlno = samlno + 1;
               sprintf(ext,"D:\\DATA\\%d.PRI\0",samlno);
               }

          /* --- INCREMENT FILE NUMBER --- */

          samlno = samlno + 1;
          fclose(f1);
          }

     /* --- LABEL AS AN INTERMEDIATE DATA FILE --- */

     else if(file_ext() == TEL_FALSE)
          {
          sprintf(ext,"D:\\DATA\\%d.INT\0",samsno);

          /* --- DO NOT OVERWRITE EXISTING DATA --- */

          while((f1 = fopen(ext,"r+")) != NULL)
               {
               fclose (f1);
               samsno = samsno + 1;
               sprintf(ext,"D:\\DATA\\%d.INT\0",samsno);
               }
```

```
                    /* --- INCREMENT FILE NUMBER --- */

                    samsno = samsno + 1;
                    fclose(f1);
                    }

               /* IF DATA TYPE CANNOT BE DETERMINED, LABEL AS .DAT */

               else
                    {
                    sprintf(ext,"D:\\DATA\\%d.dat\0",samdno);

                    /* --- DO NOT OVERWRITE EXISTING DATA --- */

                    while((f1 = fopen(ext,"r+")) != NULL)
                         {
                         fclose (f1);
                         samdno = samdno + 1;
                         sprintf(ext,"D:\\DATA\\%d.dat\0",samdno);
                         }

                    /* --- INCREMENT FILE NUMBER --- */

                    samdno = samdno + 1;
                    fclose(f1);
                    }

               /* --- RENAME THE FILE WITH NEW LABEL --- */

               rename("D:\\DATA\\TEMP.DAT",ext);
               }

          /* --- PRINT DATA FILE INFORMATION --- */

          sprintf(msg,"%ld bytes written in file %s ",length,ext);
          lprnt(msg);
          timept();
          head_stor();
          }
     }
}
```

```
/*------------------ LOOK FOR DATA COMING IN --------------*/

int datack()
{
     int i,buffsiz;

     /* --- LOOP UNTIL AT LEAST TWO BYTES IN DATA BUFFER --- */

     do
          {
          qsize_a1(port_no,&iq_size,&status);

          /* --- CHECK KEYBOARD FOR EXIT COMMAND (^B) --- */

          if(kbhit() != 0 && getch() == 0x2)
               shutdown();
          }
     while(iq_size < 2);

     /* --- LOOP UNTIL ALL DATA IS IN BUFFER --- */

     while ((iq_size < 32000)&&(iq_size != buffsiz))
          {
          buffsiz = iq_size;

          /* --- DELAY FOR 1200 BAUD SPEED --- */

          for (i = 0;i < 24000;i++)
               i = i;
          qsize_a1(port_no,&iq_size,&status);
          }
}

/*------------------ GET THE DATA AND STORE IT -------------*/

void datacol()
{
     int j,isize;
     char data;

     /* READ THE DATA IN THE BUFFER AND WRITE IT TO A FILE  */

     for(j = 1; j < iq_size; j++)
          {
          error = rdch_a1(port_no,&data,&isize,&status);
          if(error != 0)
               break;
          else
               fputc(data,f1);
          }
}
```

```
/*---------------------- PRINT THE TIME -------------------*/

void timept()
{
   time(&ltime);
   newtime = localtime(&ltime);
   sprintf(msg,"%s",asctime(newtime));
   lprnt(msg);
}

/*----------------- PRINT MESSAGE ON LINE PRINTER ---------*/

void lprnt(msg)
char *msg;
{
     FILE *fp,*fopen();

     if((fp = fopen("LPT1","w")) != NULL)
          fprintf(fp,msg);
     if(fclose(fp) == -1)
          puts("Cannot close printer");

     /* --- WRITE INFORMATION TO INFORMATION FILE --- */

     if((f2 = fopen("E:DATA.TXT","a")) != NULL)
          {
          fprintf(f2,msg);
          if(fclose(f2) == -1)
          puts("Cannot close information file on drive E:");
          }
}

/*----------------- CLOSE THE COM1 PORT AND EXIT ----------*/

void shutdown()
{
     (close_a1(port_no) == PORT_NOT_OPEN)? printf("Port not open.\n"):
     printf(" Port closed.\n");
     exit(0);
}
```

```
/* ----------- SAVE THE HEADER VALUES OF THE DATA  --------*/

void head_stor()
{
     int i,adval,sampl,min_off,year;
     float volt;

     if((f2 = fopen("E:DATA.TXT","a")) == NULL)
          {

          /* --- USE CURRENT YEAR TO FIND START OF VALID DATA --- */

          time(&ltime);
          newtime = localtime(&ltime);
          year = newtime->tm_year;
          i = 0;
          while(header[i] != year)
               {
               i++;
               if(i > 48)
                    return;
               }
          /* IF POWER CONTROL FLAG WAS TRUE, ADD 1 MINUTE TO
             TIME OF SAMPLE */

          if(header[i+15] != TEL_FALSE)
               min_off = header[i+4] + 1;
          else
               min_off = header[i+4];

          /* --- DISPLAY SAMPLE NUMBER AND TIME --- */

          sampl = header[i+8]*256 + header[i+7];
          fprintf(f2,"SAMPLE # %d at %d:%02d:%02d.%d %d/%d/19%d ",
          sampl,header[i+5],min_off,header[i+3],header[i+6],
          header[i+2],header[i+1],header[i+0]);

          /* --- CALCULATE VOLTAGE FOR CHANNEL 16 --- */

          adval = header[i+10]*256 + header[i+9];
          volt = (adval-2048)*0.00244;
          fprintf(f2,"CH 16 %5.2fv REMOTE A %2d REMOTE B %2d\n",
          volt,header[i+13],header[i+14]);
          }

     fclose(f2);
}
```

```
/* ----------- DETERMINE FILE EXTENSION ---------------------*/

int file_ext()
{
     int i,year;

     /* --- USE CURRENT YEAR TO DETERMINE START OF VALID DATA --- */

     time(&ltime);
     newtime = localtime(&ltime);
     year = newtime->tm_year;
     i = 0;
     while(header[i] != year)
          {
          i++;
          if(i > 48)
               return;
          }
     return(header[i+12]);
}
```

TELDUMP.C LISTING

```
/*                              DUMP.H                                  */

/* rev 2.21 7/13/88 by gkm */
/* rev 2.23 9/29/88 by gkm - added plot file option */

#define SCREEN 1
#define PRINTER 2
#define DISK 3
#define PLOT 4



/*----------------------- MESSAGE DEFINES -----------------*/

char *prgm_ttl = "                        DATA DUMP UTILITY\0";
char *clnt_ttl = "                     USGS TELEMETRY PROGRAM\0";
char *ver_ttl = "                         REV 2.23 9/29/88\0";
char *year_msg = "Enter the year (0 - 99) for the data: \0";
char *chan1_msg = "Enter base channel to process(1 - 16): \0";
char *chan2_msg = "Enter number of channels to process(1 - \0";
char *form_msg = "Enter data format(1=voltage 2=hex): \0";
char *out_msg = "Enter data output(1=screen,2=printer,3=file,
                 4=plot file): ";
char *cont_msg = "Do you wish to view another?(Y/N) \0";
char gen_msg[50];

/*--------------------- FUNCTION DECLARATIONS -------------*/

extern void main(void);
```

```
/*                              DUMP.C                                  */

/*------- MONITOR DATA FROM A TELEMETRY REV-I,J FILES -----*/
/* REV 2.21 7/13/88 by gkm */
/* REV 2.22 9/28/88 by gkm - Display header after entering
                      file name */
/* REV 2.23 9/29/88 by gkm - added plot file function */

#include <stdio.h>
#include <math.h>
#include <bscreen.h>
#include "dump.h"
#include "gen.h"
#include "tel_data.h"

/*--------------------- DATA FILE DEFINES -------------------*/

FILE *fopen();


void main()
{

int k,no_scan,no_ch,year,output;
int end_ch,start,end,form,base_ch,count;
int start_byte,end_byte,no_a,no_i,no_f,st_cha,st_chi,st_chf;
unsigned char scan_d[35];
long position;

for(;;)
   {
   /* --- DISPLAY PROGRAM TITLE --- */

   signon_box(prgm_ttl,clnt_ttl,ver_ttl);

   /* --- INITIALIZE START CHANNELS --- */

   st_cha = 0;
   st_chi = 0;
   st_chf = 0;
   no_a = 0;
   no_i = 0;
   no_f = 0;

   /* --- OPEN THE DATA FILE --- */

   data_fil = fil_open_wn();

   /* --- READ THE HEADER --- */

   year = get_no_wn(year_msg,0,99);
   head_box(year);
```

```
/* --- GET BASE CHANNEL TO VIEW --- */

base_ch = get_no_wn(chan1_msg,1,16);

/* --- GET NUMBER OF CHANNELS TO VIEW --- */

sprintf(gen_msg,"%s%d) ",chan2_msg,(17-base_ch));
count = get_no_wn(gen_msg,1,(17-base_ch));

/* --- GET DATA FORMAT (HEXADECIMAL OR VOLT) --- */

form = get_no_wn(form_msg,1,2);

/* --- GET OUTPUT FOR DATA --- */

output = get_no_wn(out_msg,1,4);

/* --- OPEN ANY OUTPUT FILE --- */

if(output == PRINTER)
   {
   if((out_fil = fopen("PRN","w")) == NULL)
      {
      err_wn("Cannot open the printer\n");
      break;
      }
   }
if(output == DISK)
   out_fil = fil_creat_wn();

/* --- PRINT HEADER INFORMATION TO PRINTER (IF USED) --- */

if(output == PRINTER)
   head_read(year);
```

```
/* --- SET START AND # OF CHANNELS FOR EACH DATA SET --- */

end_ch = base_ch + count - 1;

if((head.frst_ch != 0) && (base_ch < (head.frst_ch+head.frst_bch+1)))
   {
   if(end_ch < (head.frst_bch + 1))
      st_cha = 0;
   else if(base_ch < head.frst_bch +1)
      st_cha = head.frst_bch + 1;
   else
      st_cha = base_ch;
   no_a = count;
   if((st_cha + count) > (head.frst_ch + head.frst_bch+1))
      no_a = head.frst_ch + head.frst_bch + 1 - st_cha;
   }
if(head.int_ch != 0 && base_ch < (head.int_ch+head.int_bch+1))
   {
   if(end_ch < (head.int_bch + 1))
      st_chi = 0;
   else if(base_ch < head.int_bch +1)
      st_chi = head.int_bch + 1;
   else
      st_chi = base_ch;
   no_i = count - (st_chi - base_ch);
   if((st_chi + count) > (head.int_ch + head.int_bch + 1))
      no_i = head.int_ch + head.int_bch + 1 - st_chi;
   }
if(head.full_ch != 0 && base_ch < (head.full_ch+head.full_bch+1))
   {
   if(end_ch < (head.full_bch + 1))
      st_chf = 0;
   else if(base_ch < head.full_bch + 1)
      st_chf = head.full_bch + 1;
   else
      st_chf = base_ch;
   no_f = count - (st_chf - base_ch);
   if((st_chf + count) > (head.full_ch+head.full_bch+1))
      no_f = head.full_ch + head.full_bch + 1 - st_chf;
   }

/* --- IF PLOT FILE, RESET ALL PARAMETERS --- */

if(output == PLOT)
   {
   out_fil = fil_creat_wn();

   /* --- REWRITE HEADER INTO NEW FILE --- */

   fputc(year,out_fil);
   k = head.scan_flg;
   head.scan_flg = TEL_FALSE;
   fwrite(&head,1,33,out_fil);
   head.scan_flg = k;
```

```
   /* --- SET CHANNELS --- */

   st_cha = head.frst_bch + 1;
   no_a = head.frst_ch;
   st_chi = head.int_bch + 1;
   no_i = head.int_ch;
   st_chf = head.full_bch + 1;
   no_f = head.full_ch;
   }

/* --- IF ANY FIRST SCAN DATA, PROCESS IT --- */

if(head.frst_ch != 0 && st_cha != 0)
   {
   no_ch = (head.frst_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
      no_ch = no_ch +2;
   no_scan = scan_a;
   end_byte = (head.frst_ch - st_cha + head.frst_bch + 1)*2;
   start_byte = end_byte - (no_a)*2;

   /* --- SCREEN DUMP FOR HEX AND VOLT DATA --- */

   if(output == SCREEN)
      {
      if(form == HEX)
         {
         hex_ttl_box(st_cha,no_a);
         hex_scrll(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      else
         {
         volt_ttl_box(st_cha,no_a);
         volt_scrll(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      hit_cont_wn();
      }

   /* --- DISK WRITE FOR HEX AND VOLT DATA --- */

   else if(output == DISK)
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);

      if(form == HEX)
         {
         bin_dat(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      else
         {
         volt_dat(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      msg_wn_off();
      }
```

```
   /* --- DISK WRITE FOR PLOT DATA --- */

   else if(output == PLOT)
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);
      plot_dat(no_scan,no_ch,start_byte,end_byte,no_a);
      msg_wn_off();
      }

   /* --- PRINTER DUMP FOR HEX AND VOLT DATA --- */

   else
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);

      if(form == HEX)
         {
         hex_ttl(st_cha,no_a);
         hex_dat(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      else
         {
         volt_ttl(st_cha,no_a);
         volt_dat(no_scan,no_ch,start_byte,end_byte,no_a);
         }
      msg_wn_off();
      }
   }
/* --- IF FIRST SCAN THERE AND NOT USED, SKIP THIS DATA --- */

if(head.frst_ch != 0 && st_cha == 0)
   {
   no_ch = (head.frst_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
      no_ch = no_ch +2;
   no_scan = scan_a;

   for(k = 1;k < no_scan + 1;k++)
      {
      if(fread((char *)scan_d,1,no_ch,data_fil)==0)
         {
         err_wn("ERROR:data read failure");
         exit(1);
         }
      }
   }
```

```
/* --- IF PRIMARY DATA, PROCESS IT --- */

if(head.full_ch != 0 && head.data_flg != TEL_FALSE && st_chf !=0)
   {
   no_ch = (head.full_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
      no_ch = no_ch +2;
   no_scan = scan_f;
   end_byte = (head.full_ch - st_chf + head.full_bch + 1)*2;
   start_byte = end_byte - (no_f)*2;

   /* --- SCREEN DUMP FOR HEX AND VOLT DATA --- */

   if(output == SCREEN)
      {
      if(form == HEX)
         {
         hex_ttl_box(st_chf,no_f);
         hex_scrll(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      else
         {
         volt_ttl_box(st_chf,no_f);
         volt_scrll(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      }

   /* --- DISK WRITE FOR HEX AND VOLT DATA --- */

   else if(output == DISK)
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);

      if(form == HEX)
         {
         bin_dat(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      else
         {
         volt_dat(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      msg_wn_off();
      }

   /* --- DISK WRITE FOR PLOT DATA --- */

   else if(output == PLOT)
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);
      plot_dat(no_scan,no_ch,start_byte,end_byte,no_f);
      msg_wn_off();
      }
```

```
   /* --- PRINTER DUMP FOR HEX AND VOLT DATA --- */

   else
      {
      sprintf(gen_msg,"Please wait while writing data...\0");
      msg_wn_on(gen_msg);

      if(form == HEX)
         {
         hex_ttl(st_chf,no_f);
         hex_dat(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      else
         {
         volt_ttl(st_chf,no_f);
         volt_dat(no_scan,no_ch,start_byte,end_byte,no_f);
         }
      msg_wn_off();
      }
   }

/* --- IF INTERMEDIATE DATA, PROCESS IT --- */

if(head.int_ch != 0 && head.data_flg == TEL_FALSE && st_chi !=0)
   {
   no_ch = (head.int_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
      no_ch = no_ch +2;
   no_scan = scan_i;
   end_byte = (head.int_ch - st_chi + head.int_bch + 1)*2;
   start_byte = end_byte - (no_i)*2;

   /* --- SCREEN DUMP FOR HEX AND VOLT DATA --- */

   if(output == SCREEN)
      {
      if(form == HEX)
         {
         hex_ttl_box(st_chi,no_i);
         hex_scrll(no_scan,no_ch,start_byte,end_byte,no_i);
         }
      else
         {
         volt_ttl_box(st_chi,no_i);
         volt_scrll(no_scan,no_ch,start_byte,end_byte,no_i);
         }
      }
```

```
/* --- DISK WRITE FOR HEX AND VOLT DATA --- */

else if(output == DISK)
   {
   sprintf(gen_msg,"Please wait while writing data...\0");
   msg_wn_on(gen_msg);

   if(form == HEX)
      {
      bin_dat(no_scan,no_ch,start_byte,end_byte,no_i);
      }
   else
      {
      volt_dat(no_scan,no_ch,start_byte,end_byte,no_i);
      }
   msg_wn_off();
   }

/* --- DISK WRITE FOR PLOT DATA --- */

else if(output == PLOT)
   {
   sprintf(gen_msg,"Please wait while writing data...\0");
   msg_wn_on(gen_msg);
   plot_dat(no_scan,no_ch,start_byte,end_byte,no_i);
   msg_wn_off();
   }


/* --- PRINTER DUMP FOR HEX AND VOLT DATA --- */

else
   {
   sprintf(gen_msg,"Please wait while writing data...\0");
   msg_wn_on(gen_msg);

   if(form == HEX)
      {
      hex_ttl(st_chi,no_i);
      hex_dat(no_scan,no_ch,start_byte,end_byte,no_i);
      }
   else
      {
      volt_ttl(st_chi,no_i);
      volt_dat(no_scan,no_ch,start_byte,end_byte,no_i);
      }
   msg_wn_off();
   }
}
```

```
   /* --- CLOSE THE DATA FILE --- */

   fclose(data_fil);
   if(output != SCREEN)
      fclose(out_fil);

   /* --- IF DONE, CLEAR THE SCREEN AND EXIT --- */

   if(get_yn_wn(cont_msg) == 'N')
      {
      system("cls");
      exit(0);
      }
   }
}
```

TELPLOT.C LISTING

```
/*                          TELPLOT.H                          */


#define SCREEN 1
#define PRINTER 2
#define DISK 3
#define PLOT 4

#define AFLAG 1
#define IFLAG 2
#define FFLAG 3


/*----------------------- MESSAGE DEFINES -----------------*/

char *prgm_ttl = "                        DATA PLOT UTILITY\0";
char *clnt_ttl = "                  USGS TELEMETRY PROGRAM\0";
char *ver_ttl = "                        REV 1.13 9/29/88\0";
char *year_msg = " Enter year (00 - 99) of data sample: \0";
char *chan1_msg = "Enter base channel to process(1 - 16): \0";
char *chan2_msg = "Enter number of channels to process(1 - \0";
char *cont_msg = "Do you wish to view another?(Y/N) \0";
char gen_msg[50];

float data[512][17];
float max_volt,min_volt;
float start_tim,end_tim;

/*------------------ FUNCTION DECLARATIONS ----------------*/

extern void main(void);
extern long data_proc(FILE *,long,int,int,int,int,int,int);
extern void plot_init(void);
extern void dat_plt(int,int,int,float,float,int,int);
extern void legend(int,int,int,int);
extern int draw_yaxis(float,float,int,int);
extern void draw_xaxis(float,float,int,int,int,int);
extern void read_array(FILE *,int,int,int,int,int,int,int,int);
extern void par_get(int,int,int,int);
extern int re_plot(int,int);
```

```
/*                              TELPLOT.C                              */

/* REV 1.11 by gkm 7/22/88 - added x_axis scale expansion  */
/* REV 1.12 by gkm 7/25/88 - converted x_axis to time      */
/* REV 1.13 by gkm 9/29/88 - now requires a return after replot
                             question, fixed zero axis
                             calculation */
#include <stdio.h>
#include <math.h>
#include <bscreen.h>
#include <bgraph.h>
#include "telplot.h"
#include "gen.h"
#include "tel_data.h"


/*-------------------- DATA FILE DEFINES ------------------*/

FILE *of,*fopen();


void main()
{

int k,no_scan,no_ch,year,output;
int end_ch,start,end,form,base_ch,count;
int no_a,no_i,no_f,st_cha,st_chi,st_chf;
char msg[5];
int fore,back;
unsigned char scan_d[MAX_SCAN];
long ptr;

for(;;)
   {
   /* --- DISPLAY PROGRAM TITLE --- */

   signon_box(prgm_ttl,clnt_ttl,ver_ttl);

   /* --- INITIALIZE START CHANNELS --- */

   st_cha = 0;
   st_chi = 0;
   st_chf = 0;
   no_a = 0;
   no_i = 0;
   no_f = 0;

   /* --- OPEN THE DATA FILE --- */

   data_fil = fil_open_wn();
```

```
/* --- READ THE HEADER --- */

year = get_no_wn(year_msg,0,99);
head_box(year);
ptr = ftell(data_fil);

/* --- GET BASE CHANNEL TO VIEW --- */

base_ch = get_no_wn(chan1_msg,1,16);

/* --- GET NUMBER OF CHANNELS TO VIEW --- */

sprintf(gen_msg,"%s%d) ",chan2_msg,(17-base_ch));
count = get_no_wn(gen_msg,1,(17-base_ch));

/* --- SET START AND # OF CHANNELS FOR EACH DATA SET --- */

end_ch = base_ch + count - 1;

if((head.frst_ch != 0) && (base_ch < (head.frst_ch+head.frst_bch+1)))
   {
   if(end_ch < (head.frst_bch + 1))
      st_cha = 0;
   else if(base_ch < head.frst_bch +1)
      st_cha = head.frst_bch + 1;
   else
      st_cha = base_ch;
   no_a = count;
   if((st_cha + count) > (head.frst_ch + head.frst_bch+1))
      no_a = head.frst_ch + head.frst_bch + 1 - st_cha;
   }
if(head.int_ch != 0 && base_ch < (head.int_ch+head.int_bch+1))
   {
   if(end_ch < (head.int_bch + 1))
      st_chi = 0;
   else if(base_ch < head.int_bch +1)
      st_chi = head.int_bch + 1;
   else
      st_chi = base_ch;
   no_i = count - (st_chi - base_ch);
   if((st_chi + count) > (head.int_ch + head.int_bch + 1))
      no_i = head.int_ch + head.int_bch + 1 - st_chi;
   }
if(head.full_ch != 0 && base_ch < (head.full_ch+head.full_bch+1))
   {
   if(end_ch < (head.full_bch + 1))
      st_chf = 0;
   else if(base_ch < head.full_bch + 1)
      st_chf = head.full_bch + 1;
   else
      st_chf = base_ch;
   no_f = count - (st_chf - base_ch);
   if((st_chf + count) > (head.full_ch+head.full_bch+1))
      no_f = head.full_ch + head.full_bch + 1 - st_chf;
   }
```

```
/* --- IF ANY FIRST SCAN DATA, PROCESS IT --- */

if(head.frst_ch != 0 && st_cha != 0)
   {
   ptr = data_proc(data_fil,ptr,AFLAG,st_cha,no_a,scan_a,
      head.frst_sr,head.frst_flg);
   screset(3);
   }

/* --- IF FIRST SCAN THERE AND NOT USED, SKIP THIS DATA --- */

if(head.frst_ch != 0 && st_cha == 0)
   {
   no_ch = (head.frst_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
      no_ch = no_ch +2;
   no_scan = scan_a;

   for(k = 1;k < no_scan + 1;k++)
      {
      if(fread((char *)scan_d,1,no_ch,data_fil)==0)
         {
         err_wn("ERROR:data read failure");
         exit(1);
         }
      }
   ptr = ftell(data_fil);
   }

/* --- IF PRIMARY DATA, PROCESS IT --- */

if(head.full_ch != 0 && head.data_flg != TEL_FALSE && st_chf !=0)
   {
   ptr = data_proc(data_fil,ptr,FFLAG,st_chf,no_f,scan_f,
      head.full_sr,head.full_flg);
   screset(3);
   }

/* --- IF INTERMEDIATE DATA, PROCESS IT --- */

if(head.int_ch != 0 && head.data_flg == TEL_FALSE && st_chi !=0)
   {
   ptr = data_proc(data_fil,ptr,IFLAG,st_chi,no_i,scan_i,
      head.int_sr,head.int_flg);
   screset(3);
   }

/* --- CLOSE THE DATA FILE --- */

fclose(data_fil);
if(output != SCREEN)
   fclose(of);
```

```
   /* --- IF DONE, CLEAR THE SCREEN AND EXIT --- */

   if(get_yn_wn(cont_msg) == 'N')
      {
      system("cls");
      exit(0);
      }
   }
}

/*---------------------- PROCESS A DATA SET -----------------*/

long data_proc(data_fil,fil_ptr,flag,start_ch,count,no_scan,smpl_rat,sr_flag)
FILE *data_fil;
int flag,start_ch,count,no_scan,smpl_rat,sr_flag;
long fil_ptr;
{
   int i,no_byt,start_byte,end_byte,end_ch;
   int fore,back;
   int max_scan,min_scan,no_avg,base_ln,t_scan;
   float x_scale,y_scale;
   char msg[5];


   /* --- GET TOTAL NUMBER OF BYTES PER SCAN --- */

   if(flag == AFLAG)
      no_byt = (head.frst_ch*2) + 1;
   else if(flag == IFLAG)
      no_byt = (head.int_ch*2) + 1;
   else
      no_byt = (head.full_ch*2) + 1;
   if(head.scan_flg != TEL_FALSE)
         no_byt = no_byt +2;
```

```
/* --- GET CHANNEL STARTS AND ENDS --- */

end_ch = start_ch + count - 1;
if(flag == AFLAG)
   end_byte = (head.frst_ch - start_ch + head.frst_bch + 1)*2;
else if(flag == IFLAG)
   end_byte = (head.int_ch - start_ch + head.int_bch + 1)*2;
else
   end_byte = (head.full_ch - start_ch + head.full_bch + 1)*2;
start_byte = end_byte - (count)*2;

if(no_scan < 512)
   x_scale = 512/no_scan;
else
   x_scale = 1.00;
min_scan = 0;
max_scan = no_scan;
y_scale = 1.00;
max_volt = 5.00;
min_volt = -5.00;
t_scan = no_scan;
start_tim = 0.00;
end_tim = no_scan/smpl_rat;
for(;;)
   {

   /* --- READ THE DATA --- */

   if(no_scan <= 512)
      no_avg = 1;
   else
      {
      for(i = no_scan;i > 1;i--)
         {
         no_avg = i/512;
         if(fmod((double)i,(double)512) == (double)0.0)
            break;
         }
      max_scan = min_scan + i;
      no_scan = i;
      }
   end_tim = max_scan/smpl_rat;
   arry_clr();
   read_array(data_fil,start_byte,end_byte,end_ch,no_byt,
      no_avg,min_scan,max_scan,t_scan);
```

```
        /* --- DRAW THE AXIS AND LABELS --- */

        plot_init();
        fore = BLUE;
        back = BLACK;
        base_ln = draw_yaxis(max_volt,min_volt,fore,back);
        draw_xaxis(end_tim,start_tim,sr_flag,fore,back,base_ln);
        legend(start_ch,end_ch,fore,back);
        if(base_ln == 3)
           base_ln = 3 + (int)((max_volt*16)*y_scale);
        if(base_ln == b_maxy - 36)
           base_ln = base_ln + (int)((min_volt*16)*y_scale);
        dat_plt(start_ch,end_ch,no_scan,x_scale,y_scale,base_ln,no_avg);

        fore = WHITE;
        if(re_plot(fore,back) == 'Y')
           {
           par_get(t_scan,smpl_rat,fore,back);
           scclear();
           fore = BLUE;
           min_scan = (int)(smpl_rat*start_tim);
           max_scan = (int)(end_tim*smpl_rat);
           no_scan = max_scan - min_scan;
           if(no_scan < 512)
              x_scale = (float)(512.0/no_scan);
           else
              x_scale = 1.0;
           y_scale = 10.0/(max_volt - min_volt);

           if(fseek(data_fil,fil_ptr,SEEK_SET) != NULL)
              puts("Could not rewind the file");
           }
        else
           {
           fil_ptr = ftell(data_fil);
           return fil_ptr;
           }
        }
}
```

```
/*------------------------ PLOT THE DATA ------------------*/

void plot_init()
{
   int gr_mode;
   int mode,columns,act_page,color;

   /* --- INITIALIZE THE GRAPHICS MODE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      gr_mode = 6;
   else
      {
      if (mode == 3 || mode == 14)
         gr_mode = 14;
      else
         gr_mode = 6;
      }
   if(grinit(gr_mode,0,0) == -1)
      {
      err_wn("Graphics could not be set");
      return;
      }
}
```

```
/*---------------------- PLOT THE DATA --------------------*/

void dat_plt(first,last,no_scan,x_scale,y_scale,zero_axis,no_avg)
int first,last,zero_axis,no_scan,no_avg;
float x_scale,y_scale;
{
   struct{
      PT now;
      PT last;
      }data_pt[17];
   int i,k,color,old_color;

   for(i = 0; i < 17;i++)
      {
      data_pt[i].now.x = b_home.x;
      data_pt[i].now.y = b_home.y;
      data_pt[i].last.x = b_home.x;
      data_pt[i].last.y = b_home.y;
      }
   for(k = 0;k < (no_scan/no_avg);k++)
      {
      color = first;
      for(i = first;i <= last;i++)
         {
         data_pt[i].now.y = zero_axis - ((data[k][i]*16)*y_scale);
         data_pt[i].now.x = (k*x_scale) + 64;
         if(data_pt[i].now.y < 0)
            data_pt[i].now.y = 0;
         if(data_pt[i].now.y > b_maxy)
            data_pt[i].now.y = b_maxy;
         old_color = color;
         if(color == 16)
            color = RED;
         if(k == 0)
            color = BLUE;
         grline(&data_pt[i].last,&data_pt[i].now,color);
         data_pt[i].last.y = data_pt[i].now.y;
         data_pt[i].last.x = data_pt[i].now.x;
         color = old_color;
         color++;
         }
      }
}
```

```
/*----------------- DISPLAY THE CHANNEL COLORS ------------*/

void legend(frst_ch,lst_ch,fore,back)
int frst_ch,lst_ch,fore,back;
{
   int i,x_axis,color,old_color;
   char msg[5];

   x_axis = 0;
   color = frst_ch;
   for(i = frst_ch;i <= lst_ch; i++)
      {
      old_color = color;
      if(color == 16)
         color = RED;
      sprintf(msg," CH%2d",i);
      scdspmsg(24,x_axis,color,back,msg);
      x_axis = x_axis + 5;
      color = old_color;
      color++;
      }
}
/*---------------------- DRAW THE Y-AXIS --------------------*/

int draw_yaxis(y_max,y_min,fore,back)
float y_max,y_min;
int fore,back;
{
   PT pos,pos_nxt;
   char msg[6];
   float y_tic,label,zero;
   int zero_axis,x_axis,y_axis;

   /* --- LABEL THE Y-AXIS --- */

   y_tic = (y_max - y_min)/10;
   label = y_max;
   y_axis = 0;
   x_axis = 0;
   while(label >= y_min)
      {
      sprintf(msg,"%+5.3f",label);
      scdspmsg(y_axis,x_axis,fore,back,msg);
      y_axis = y_axis + 2;
      label = label - y_tic;
      }
   scdspmsg(y_axis-1,x_axis,fore,back,"volts");
```

```
    /* --- DRAW THE Y - AXIS --- */

    pos.x = 64;
    pos.y = 3;
    pos_nxt.x = 64;
    pos_nxt.y = b_maxy - 37;
    grline(&pos,&pos_nxt,fore);
    pos.x = 54;
    pos.y = 3;
    pos_nxt.x = 64;
    pos_nxt.y = 3;
    while(pos.y < b_maxy -36)
       {
       grline(&pos,&pos_nxt,fore);
       pos.y = pos.y +16;
       pos_nxt.y = pos.y;
       }

    /* --- SET THE ZERO AXIS --- */

    if(y_max <= 0)
       zero_axis = 3;
    else if(y_min >= 0)
       zero_axis = b_maxy - 36;
    else
       {
       zero = (y_max - y_min)/160;
       zero_axis = 3 + (int)(y_max/zero);
       }
    b_home.x = 64;
    b_home.y = zero_axis;
    return zero_axis;
}
```

```
/*--------------------- DRAW THE X-AXIS -------------------*/

void draw_xaxis(x_max,x_min,flag,fore,back,zero_axis)
int flag,fore,back,zero_axis;
float x_max,x_min;
{
   PT pos,pos_nxt;
   char msg[6];
   float x_tic,label,zero;
   int x_axis,y_axis;

   pos.x = 64;
   pos.y = zero_axis;
   pos_nxt.x = b_maxx- 64;
   pos_nxt.y = zero_axis;
   grline(&pos,&pos_nxt,fore);

   pos.x = 64;
   pos.y = zero_axis - 5;
   pos_nxt.x = 64;
   pos_nxt.y = zero_axis + 5;
   while(pos.x < b_maxx -60)
      {
      grline(&pos,&pos_nxt,fore);
      pos.x = pos.x + 64;
      pos_nxt.x = pos.x;
      }

   /* --- LABEL THE X-AXIS --- */

   x_tic = (x_max - x_min)/8;
   if(x_min == 1)
      label = x_min;
   else
      label = x_min;
   y_axis = (zero_axis/8) + 1;
   x_axis = 5;
   while(x_axis < 76)
      {
      sprintf(msg,"%5.1f",label);
      scdspmsg(y_axis,x_axis,fore,back,msg);
      x_axis = x_axis + 8;
      label = label + x_tic;
      }
   if(flag == TEL_TRUE)
      scdspmsg((zero_axis/8),74,fore,back,"min.");
   else
      scdspmsg((zero_axis/8),74,fore,back,"sec.");
}
```

```
/*------------ READ PLOT DATA INTO ARRAY --------------*/

void read_array(read_fil,frst_byt,lst_byt,lst_ch,scn_lgth,avg_scn,min_scan,
max_scan,total_scan)
FILE *read_fil;
int frst_byt,lst_byt,lst_ch,scn_lgth,avg_scn,min_scan,max_scan,total_scan;
{
   int i,j,k,adval,avg_ct,chnl;
   float volt;
   unsigned char scan_s[MAX_SCAN];

   for (k = 0;k < min_scan;k++)
         {
         if(fread((char *)scan_s,1,scn_lgth,read_fil) == NULL)
            {
            puts("Read failure");
            exit;
            }
         }
   k = min_scan;
   j = 0;
   while(k < max_scan)
      {
      for (avg_ct = 0;avg_ct < avg_scn;avg_ct++)
         {
         if(fread((char *)scan_s,1,scn_lgth,read_fil) == NULL)
            {
            puts("averaging read failure");
            exit;
            }
         else
            {
            for(i = frst_byt; i < lst_byt;i++)
               {
               adval = scan_s[i+1]*256 + scan_s[i];
               volt = (adval-2048)*0.00244;
               avg[i/2] = avg[i/2] + volt;
               i++;
               }
            }
         k++;
         }

      /* --- SAVE AVERAGES IN DATA PLOT BUFFER --- */

      avg_get(avg_scn,frst_byt/2,lst_byt/2);
      chnl = lst_ch;
      for(i = frst_byt/2;i < lst_byt/2;i++)
         {
         data[j][chnl] = avg[i];
         chnl--;
         }
      j++;
```

```
    /* --- CLEAR THE AVERAGE ARRAY --- */

    for(i=0;i<17;i++)
        avg[i] = 0;
    }

    for (k = max_scan;k < total_scan;k++)
        {
        if(fread((char *)scan_s,1,scn_lgth,read_fil) == NULL)
           {
           puts("Read failure");
           return;
           }
         }
}

/*---------------- GET NEW PLOT PARAMETERS ----------------*/

void par_get(scans,smpl_rat,fore,back)
int scans,smpl_rat,fore,back;
{
   char msg[20];

   do
      {
      scdspmsg(23,0,fore,back,"Enter maximum voltage: ");
      sccurset(23,23);
      gets(msg);
      max_volt = atof(msg);
      }
   while(max_volt > 5);
   scclrmsg(23,0,75);
   do
      {
      scdspmsg(23,0,fore,back,"Enter minimum voltage: ");
      sccurset(23,23);
      gets(msg);
      min_volt = atof(msg);
      }
   while(min_volt < -5 || min_volt > max_volt);
   scclrmsg(23,0,75);
   do
      {
      scdspmsg(23,0,fore,back,"Enter start time: ");
      sccurset(23,28);
      gets(msg);
      start_tim = atof(msg);
      }
   while(start_tim >= (scans/smpl_rat));
   scclrmsg(23,0,75);
```

```
   do
      {
      scdspmsg(23,0,fore,back,"Enter stop time: ");
      sccurset(23,28);
      gets(msg);
      end_tim = atof(msg);
      }
   while(end_tim <= start_tim || end_tim > (scans/smpl_rat));
}

/*-------------------- ASK FOR REPLOT OF DATA --------------*/

int re_plot(fore,back)
int fore,back;
{
   int c;
   char msg[5];

   do
      {
      scdspmsg(23,0,fore,back,"REPLOT DATA(Y/N) ");
      sccurset(23,18);
      c = toupper(getche());
      }
   while(c != 'N' && c != 'Y');
   gets(msg);

   return c;
}
```

TELDATA.C LISTING

```
/*------------------------- TEL_DATA.H --------------------*/

/*------ GENERAL DEFINES FOR TELEMETRY DATA FILES ---------*/

/* REV 2.00 by gkm 7/8/88 - written for screen display only*/
/* REV 2.10 by gkm 7/11/88 - added printer output          */
/* REV 2.20 by gkm 7/12/88 - added disk output             */
/* REV 2.21 by gkm 7/13/88 - added error checks and display.
                             Altered binary output to data
                             only ( no scan count. )       */
/* REV 2.21 by gkm 9/28/88 - redefined the data and output
                             files as global pointers */

#define HEAD_SIZ 34         /* Header is 34 bytes long */
#define MAX_SCAN 35         /* Max. scan size is 35 bytes */
#define TEL_TRUE 0xF2       /* True value used in telemetry */
#define TEL_FALSE 0x81      /* False value used in telemetry */
#define SEPARATOR 0xFF      /* Scan separator */

#define BIN 0
#define VOLT 1
#define HEX 2
#define PLT 3

/*------------------- HEADER FILE DEFINES -----------------*/

struct dat_head{
   unsigned char mnth;      /* Sample month */
   unsigned char day;       /* Sample day */
   unsigned char sec;       /* Sample second */
   unsigned char mnt;       /* Sample minute */
   unsigned char hr;        /* Sample hour */
   unsigned char t_sec;     /* Sample tenths of seconds */
   unsigned char l_samno;   /* Low-byte sample number */
   unsigned char h_samno;   /* High-byte sample number */
   unsigned char l_ch16;    /* Low-byte channel 16 data */
   unsigned char h_ch16;    /* High-byte channel 16 data */
   unsigned char scan_flg;  /* Scan count in data flag */
   unsigned char data_flg;  /* Primary data set flag */
   unsigned char rem_a;     /* Remote event channel A data */
   unsigned char rem_b;     /* Remote event channel B data */
   unsigned char pwr_flg;   /* Power control flag */
   unsigned char frst_bch;  /* Base channel for 1st scan set */
   unsigned char int_bch;   /* Base channel for int. data */
   unsigned char full_bch;  /* Base channel for primary data */
   unsigned char frst_ch;   /* No. channels for 1st scan set */
   unsigned char int_ch;    /* No. channels for int. data */
   unsigned char full_ch;   /* No. channels for primary data */
   unsigned char frst_sr;   /* Scan rate for 1st scan set */
   unsigned char int_sr;    /* Scan rate for int. data */
   unsigned char full_sr;   /* Scan rate for primary data */
   unsigned char frst_flg;  /* Time base flag for 1st scan set */
   unsigned char int_flg;   /* Time base flag for int. data */
   unsigned char full_flg;  /* Time base flag for primary data */
   unsigned char frst_lscn; /* Low-byte # scans for 1st scan*/
```

```
   unsigned char frst_hscn; /* High-byte # scans for 1st scan*/
   unsigned char int_lscn;  /* Low-byte # scans for int. data */
   unsigned char int_hscn;  /* High-byte # scans for int. data*/
   unsigned char full_lscn; /* Low-byte # scans for prim. data */
   unsigned char full_hscn; /* High-byte # scans for prim. data */
   } head;

/*----- COMMON STORAGE PARAMETERS USED BY ALL FUNCTIONS ---*/


int scan_a;     /* Storage for # scans of 1st scan data */
int scan_i;     /* Storage for # scans of intermediate data */
int scan_f;     /* Storage for # scans of full data */
int base_cha;   /* Storage for base channel of 1st scan data */
int base_chi;   /* Storage for base channel of intermediate data */
int base_chf;   /* Storage for base channel of full data */
int end_cha;    /* Storage for end channel of 1st scan data */
int end_chi;    /* Storage for end channel of intermediate data */
int end_chf;    /* Storage for end channel of full data */

float avg[17];  /* Storage for averaging values */
float mn[17];   /* Storage for minimum values */
float mx[17];   /* Storage for maximum values */

FILE *data_fil; /* Data file pointer */
FILE *out_fil;  /* Output file pointer */

/*------------------- FUNCTION DECLARATIONS ---------------*/

extern int head_read(int);
extern void head_box(int);
extern int hex_data(int,int,unsigned,unsigned,unsigned);
extern int bin_data(int,int,unsigned,unsigned,unsigned);
extern int hex_scrll(int,int,unsigned,unsigned,unsigned);
extern int volt_data(int,int,unsigned,unsigned,unsigned);
extern int volt_scrll(int,int,unsigned,unsigned,unsigned);
extern void maxmin(int,float);
extern void avg_get(int,int,int);
extern void arry_clr(void);
extern void volt_ttl(unsigned,unsigned);
extern void volt_ttl_box(unsigned,unsigned);
extern void hex_ttl(unsigned,unsigned);
extern void hex_ttl_box(unsigned,unsigned);
```

```
/*                          TEL_DATA.C                          */

/* ROUTINES TO HANDLE DATA FROM REV I & J TELEMETRY FILES  */


/* REV 2.00 by gkm 7/8/88 - written for screen display only*/
/* REV 2.10 by gkm 7/11/88 - added printer output          */
/* REV 2.20 by gkm 7/12/88 - added disk output             */
/* REV 2.21 by gkm 7/13/88 - added error checks and display.
                      Altered binary output to data
                      only ( no scan count. )       */
/* REV 2.22 by gkm 9/28/88 - moved header block to middle of
                      screen, moved title bar below the
                      header block, moved data section
                      to lower part of screen, and added
                      routines that will look for data
                      errors(missing data) and fill in
                      the missing sections with zeros */
/* REV 2.23 by gkm 9/29/88 - added plot file functions */

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <bscreen.h>
#include "gen.h"
#include "tel_data.h"

/*--------------- READ AND WRITE HEADER INFORMATION -------*/

head_read(yr)
int yr;
{
   int i,minute,adval,sam_no;
   float volt;
   char year[1];

   /* --- SEARCH FOR START OF DATA --- */

   i = 0;
   while(year[0] != yr)
      {
      fread((char *)year,1,1,data_fil);
      i++;
      if(i > 48)
         return;
      }

   /* --- READ THE HEADER --- */

   if(fread(&head,1,33,data_fil) == NULL)
      {
      err_wn("Read error on data");
      return;
      }
```

```
/* IF POWER CONTROL FLAG WAS SET, ADD 1 MINUTE TO TIME */

if(head.pwr_flg != TEL_FALSE)
   minute = head.mnt + 1;
else
   minute = head.mnt;

/* --- WRITE SAMPLE NUMBER AND TIME --- */

sam_no = head.h_samno*256 + head.l_samno;
fprintf(out_fil,"\nSample %d at %d:%02d:%02d.%d %d/%d/19%d\n",
   sam_no,head.hr,minute,head.sec,head.t_sec,head.day,
   head.mnth,year[0]);

/* --- DETERMINE VOLTAGE VALUE FOR CHANNEL 16 DATA --- */

adval = head.h_ch16*256 + head.l_ch16;
volt = (adval-2048)*0.00244;
fprintf(out_fil,"Channel 16 %5.2f volts  ",volt);

/* --- WRITE REMOTE COUNTER VALUES --- */

fprintf(out_fil,"Remote Counter A %2d  ",head.rem_a);
fprintf(out_fil,"Remote Counter B %2d\n",head.rem_b);

/* --- DETERMINE SCAN COUNT FOR EACH DATA SET --- */

scan_a = head.frst_hscn*256 + head.frst_lscn;
scan_i = head.int_hscn*256 + head.int_lscn;
scan_f = head.full_hscn*256 + head.full_lscn;

/* --- DETERMINE BASE CHANNEL FOR EACH DATA SET --- */

base_cha = head.frst_bch + 1;
base_chi = head.int_bch + 1;
base_chf = head.full_bch + 1;

/* --- DETERMINE THE END CHANNEL FOR EACH DATA SET --- */

end_cha = head.frst_ch + head.frst_bch;
end_chi = head.int_ch + head.int_bch;
end_chf = head.full_ch + head.full_bch;

/* --- WRITE DATA FORMAT FOR 1ST SCAN SET DATA --- */

fprintf(out_fil,"%4d scans of channels %d to %d at %d ",
   scan_a,base_cha,end_cha,head.frst_sr);
if(head.frst_flg == TEL_FALSE)
   fprintf(out_fil,"scans/sec\n");
else
   fprintf(out_fil,"scans/hr\n");
```

```
   /* --- WRITE DATA FORMAT FOR INTERMEDIATE DATA --- */

   fprintf(out_fil,"%4d scans of channels %d to %d at %d ",
      scan_i,base_chi,end_chi,head.int_sr);
   if(head.int_flg == TEL_FALSE)
      fprintf(out_fil,"scans/sec\n");
   else
      fprintf(out_fil,"scans/hr\n");

   /* --- WRITE DATA FORMAT FOR PRIMARY DATA --- */

   fprintf(out_fil,"%4d scans of channels %d to %d at %d ",
      scan_f,base_chf,end_chf,head.full_sr);
   if(head.full_flg == TEL_FALSE)
      fprintf(out_fil,"scans/sec\n");
   else
      fprintf(out_fil,"scans/hr\n");
}

/*-- READ AND DISPLAY HEADER INFORMATION (BOX DISPLAY) ----*/

void head_box(yr)
int yr;
{
   int i,minute,adval,sam_no;
   float volt;
   char year[1],msg[80];
   int mode,fore,back,columns,act_page;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = BLACK;
      back = WHITE;
      }
   else
      {
      fore = WHITE;
      back = RED;
      }
```

```
/* --- DRAW DISPLAY BOX ON THE SCREEN --- */

scbox(5,0,10,79,0,-1,NORMAL);
scatrect(5,0,10,79,fore,back);


i = 0;
while(year[0] != yr)
   {
   fread((char *)year,1,1,data_fil);
   i++;
   if(i > 48)
      return;
   }

/* --- READ THE HEADER DATA --- */

if(fread(&head,1,33,data_fil) == NULL)
   {
   err_wn("Read error on data");
   return;
   }

/* IF POWER CONTROL FLAG WAS SET, ADD 1 MINUTE TO TIME */

if(head.pwr_flg != TEL_FALSE)
   minute = head.mnt + 1;
else
   minute = head.mnt;

/* --- DISPLAY SAMPLE NUMBER AND TIME --- */

sam_no = head.h_samno*256 + head.l_samno;
sprintf(msg,"Sample %d%d at %d:%02d:%02d.%d %d/%d/19%d",
   head.h_samno,head.l_samno,head.hr,minute,head.sec,
   head.t_sec,head.day,head.mnth,year[0]);
scdspmsg(5,24,fore,back,msg);

/* --- DISPLAY CHANNEL 16 DATA AS VOLTAGE --- */

adval = head.h_ch16*256 + head.l_ch16;
volt = (adval-2048)*0.00244;
sprintf(msg,"Channel 16 %5.2f volts  ",volt);
scdspmsg(6,1,fore,back,msg);

/* --- DISPLAY REMOTE COUNTERS DATA --- */

sprintf(msg,"Remote Counter A %2d  Remote Counter B %2d",
   head.rem_a,head.rem_b);
scdspmsg(6,25,fore,back,msg);
```

```
   /* --- DETERMINE SCAN COUNT FOR EACH DATA SET --- */

   scan_a = head.frst_hscn*256 + head.frst_lscn;
   scan_i = head.int_hscn*256 + head.int_lscn;
   scan_f = head.full_hscn*256 + head.full_lscn;

   /* --- DETERMINE BASE CHANNEL FOR EACH DATA SET --- */

   base_cha = head.frst_bch + 1;
   base_chi = head.int_bch + 1;
   base_chf = head.full_bch + 1;

   /* --- DETERMINE THE END CHANNEL FOR EACH DATA SET --- */

   end_cha = head.frst_ch + head.frst_bch;
   end_chi = head.int_ch + head.int_bch;
   end_chf = head.full_ch + head.full_bch;

   /* --- DISPLAY DATA FORMAT FOR 1ST SCAN SET DATA --- */

   sprintf(msg,"%4d scans of channels %2d to %2d at %3d ",
      scan_a,base_cha,end_cha,head.frst_sr);
   scdspmsg(7,1,fore,back,msg);

   if(head.frst_flg == TEL_FALSE)
      sprintf(msg,"scans/sec");
   else
      sprintf(msg,"scans/hr");
   scdspmsg(7,40,fore,back,msg);

   /* --- DISPLAY DATA FORMAT FOR INTERMEDIATE DATA --- */

   sprintf(msg,"%4d scans of channels %2d to %2d at %3d ",
      scan_i,base_chi,end_chi,head.int_sr);
   scdspmsg(8,1,fore,back,msg);

   if(head.int_flg == TEL_FALSE)
      sprintf(msg,"scans/sec");
   else
      sprintf(msg,"scans/hr");
   scdspmsg(8,40,fore,back,msg);

   /* --- DISPLAY DATA FORMAT FOR PRIMARY DATA --- */

   sprintf(msg,"%4d scans of channels %2d to %2d at %3d ",
      scan_f,base_chf,end_chf,head.full_sr);
   scdspmsg(9,1,fore,back,msg);

   if(head.full_flg == TEL_FALSE)
      sprintf(msg,"scans/sec");
   else
      sprintf(msg,"scans/hr");
   scdspmsg(9,40,fore,back,msg);
}
```

```
/*----------- READ AND WRITE HEXADECIMAL SCAN DATA --------*/

hex_dat(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
   char msg[80];
   unsigned char scan_s[MAX_SCAN];
   int j,k,i,scan,numread;
   long position;

   for(k = 1;k < scans + 1;k++)
      {
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         {
         err_wn("ERROR:data read failure");
         break;
         }

      /* --- WRITE HEX VALUES WITH SCAN COUNT --- */


      else if(head.scan_flg != TEL_FALSE)
           {
           scan = scan_s[chans-2]*256 + scan_s[chans-3];
           if(scan != k)
               {
               scan = dat_corr(k,scans,chans,count,HEX);
               k = scan;
               position = ftell(data_fil) - chans;
               if(position == -1L)
               err_wn("ERROR:could not find file position");
               if(fseek(data_fil,position,SEEK_SET) != 0)
                    err_wn("ERROR:could not reposition file");
               if(fread((char *)scan_s,1,chans,data_fil) == NULL)
                    err_wn("ERROR:data read failure");
               }
           if(count != 16)
               j = sprintf(msg,"%4d ",scan);
           else
               j = 0;
           for(i = start; i < last;i++)
               {
               j += sprintf(msg+j,"%02x%02x ",scan_s[i+1],scan_s[i]);
               i++;
               }
          fprintf(out_fil,"%s\n\r",msg);
          }
```

```
   /* --- WRITE HEX VALUES AND USE LOOP VALUE FOR SCAN COUNT --- */

   else
     {
     if(count != 16)
          j = sprintf(msg,"%4d ",k);
     else
          j = 0;
     for(i = start; i < last;i++)
          {
          j += sprintf(msg+j,"%02x%02x ",scan_s[i+1],scan_s[i]);
          i++;
          }
     fprintf(out_fil,"%s\n\r",msg);
     }
   kbhit();
   }

}

/*------------ READ AND WRITE BINARY SCAN DATA --------*/

bin_dat(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
     unsigned char scan_s[MAX_SCAN];
     int k,i,scan;
     char msg[20];
     long position;

     for(k = 1;k < scans + 1;k++)
          {
          if(fread((char *)scan_s,1,chans,data_fil) == NULL)
               {
               err_wn("ERROR:data read failure");
               break;
               }
```

```
        /* --- WRITE HEX VALUES WITH SCAN COUNT --- */


        else if(head.scan_flg != TEL_FALSE)
             {
             scan = scan_s[chans-2]*256 + scan_s[chans-3];
             if(scan != k)
                  {
                  scan = dat_corr(k,scans,chans,count,BIN);
                  k = scan;
                  position = ftell(data_fil) - chans;
                  if(position == -1L)
                       err_wn("ERROR:could not find file position");
                  if(fseek(data_fil,position,SEEK_SET) != 0)
                       err_wn("ERROR:could not reposition file");
                  if(fread((char *)scan_s,1,chans,data_fil) == NULL)
                       err_wn("ERROR:data read failure");
                  }
             for(i = start; i < last;i++)
                  fputc(scan_s[i],out_fil);
             kbhit();

             }

        /* --- WRITE HEX VALUES AND USE LOOP VALUE FOR SCAN COUNT --- */

        else
             {
             for(i = start; i < last;i++)
                  fputc(scan_s[i],out_fil);
             }
   }

}
```

```
/*----------- READ AND WRITE PLOT SCAN DATA --------*/

plot_dat(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
   unsigned char scan_s[MAX_SCAN];
   int k,i,scan;
   char msg[20];
   long position;

   for(k = 1;k < scans + 1;k++)
      {
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         {
         err_wn("ERROR:data read failure");
         break;
         }

      /* --- WRITE HEX VALUES WITH SCAN COUNT --- */


      else if(head.scan_flg != TEL_FALSE)
        {
        scan = scan_s[chans-2]*256 + scan_s[chans-3];
      if(scan != k)
         {
         scan = dat_corr(k,scans,chans,count,PLT);
         k = scan;
         position = ftell(data_fil) - chans;
         if(position == -1L)
            err_wn("ERROR:could not find file position");
         if(fseek(data_fil,position,SEEK_SET) != 0)
            err_wn("ERROR:could not reposition file");
         if(fread((char *)scan_s,1,chans,data_fil) == NULL)
            err_wn("ERROR:data read failure");
         }
      fwrite((char *)scan_s,1,chans-3,out_fil);
      fputc(0xFF,out_fil);
      kbhit();

      }

   /* --- WRITE HEX VALUES AND USE LOOP VALUE FOR SCAN COUNT --- */

   else
      {
        for(i = start; i < last;i++)
         fputc(scan_s[i],out_fil);
      }
   }

}
```

```
/*---- READ AND WRITE HEXADECIMAL SCAN DATA (SCROLL ) ---------*/

hex_scrll(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
   unsigned char scan_s[MAX_SCAN];
   int chnl,j,k,i,scan;
   char c,msg[79];
   int mode,fore,back,columns,act_page;
   long position;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      }
   else
      {
      fore = WHITE;
      back = BLACK;
      }
   scatrect(12,0,24,79,fore,back);
   sccurset(24,0);
   scpgcur(1,12,13,0);
   scpscrol(0,NORMAL,12,1,24,79,SCR_UP);

   for(k = 1;k < scans + 1;k++)
      {
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         {
         err_wn("ERROR:data read failure");
         break;
         }

      /* --- DISPLAY HEX VALUES WITH SCAN COUNT --- */

      else if(head.scan_flg != TEL_FALSE)
        {
        scan = scan_s[chans-2]*256 + scan_s[chans-3];
      if(scan != k)
         {
         scan = dat_corr_scroll(k,scans,chans,count,HEX,fore,back);
         k = scan;
         position = ftell(data_fil) - chans;
         if(position == -1L)
            err_wn("ERROR:could not find file position");
         if(fseek(data_fil,position,SEEK_SET) != 0)
            err_wn("ERROR:could not reposition file");
         if(fread((char *)scan_s,1,chans,data_fil) == NULL)
            err_wn("ERROR:data read failure");
         }
      if(count != 16)
```

```
            j = sprintf(msg,"%4d ",scan);
        else
           j = 0;
          for(i = start; i < last;i++)
              {
              j += sprintf(msg+j,"%02x%02x ",scan_s[i+1],scan_s[i]);
              i++;
              }
        scdspmsg(24,1,fore,back,msg);
        scpscrol(1,NORMAL,12,1,24,79,SCR_UP);
        kbhit();
        }

    /* --- DISPLAY HEX VALUES AND USE LOOP VALUE FOR SCAN COUNT --- */

    else
        {
        if(count != 16)
            j = sprintf(msg,"%4d ",k);
        else
           j = 0;
          for(i = start; i < last;i++)
              {
              j += sprintf(msg+j,"%02x%02x ",scan_s[i+1],scan_s[i]);
              i++;
              }
        scdspmsg(24,1,fore,back,msg);
        scpscrol(1,NORMAL,12,1,24,79,SCR_UP);
        kbhit();
        }
    }

}
```

```
/*------------- READ AND WRITE VOLTAGE SCAN DATA ----------*/

volt_dat(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
   char msg[80];
   unsigned char scan_s[MAX_SCAN];
   int j,k,i,chnl,scan,adval;
   float volt;
   long position;

   /* --- CLEAR AVERAGE, MAX, MIN ARRAYS --- */

   arry_clr();

   /* --- READ EACH SCAN OF DATA --- */

   for(k = 1;k < scans + 1;k++)
      {
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         {
         err_wn("ERROR:data read failure");
         break;
         }

      /* --- WRITE VOLTAGES WITH SCAN COUNT IN DATA --- */

      else if(head.scan_flg != TEL_FALSE)
         {
         scan = scan_s[chans-2]*256 + scan_s[chans-3];
         if(scan != k)
            {
            scan = dat_corr(k,scans,chans,count,VOLT);
            k = scan;
            position = ftell(data_fil) - chans;
            if(position == -1L)
               err_wn("ERROR:could not find file position");
            if(fseek(data_fil,position,SEEK_SET) != 0)
               err_wn("ERROR:could not reposition file");
            if(fread((char *)scan_s,1,chans,data_fil) == NULL)
               err_wn("ERROR:data read failure");
            }
         j = sprintf(msg,"%4d",scan);
         for(i = start; i < last;i++)
            {
            adval = scan_s[i+1]*256 + scan_s[i];
            volt = (adval-2048)*0.00244;
            avg[i/2] = avg[i/2] + volt;
            maxmin(i/2,volt);
            if(count < 9)
               j += sprintf(msg+j," %7.4f",volt);
            else if (count <11)
               j += sprintf(msg+j," %6.3f",volt);
```

```
            else if(count < 13)
               j += sprintf(msg+j," %5.2f",volt);
            else
               j += sprintf(msg+j,"%4.1f",volt);
            i++;
            }
         fprintf(out_fil,"%s\n\r",msg);
         }

      /* WRITE VOLTAGES AND USE LOOP COUNT FOR SCAN COUNT */

      else
         {
         j = sprintf(msg,"%4d ",k);
         for(i = start; i < last;i++)
            {
            adval = scan_s[i+1]*256 + scan_s[i];
            volt = (adval-2048)*0.00244;
            avg[i/2] = avg[i/2] + volt;
            maxmin(i/2,volt);
            if(count < 9)
               j += sprintf(msg+j," %7.4f",volt);
            else if (count <11)
               j += sprintf(msg+j," %6.3f",volt);
            else if(count < 13)
               j += sprintf(msg+j," %5.2f",volt);
            else
               j += sprintf(msg+j,"%4.1f",volt);
            i++;
            }
         fprintf(out_fil,"%s\n\r",msg);
         }
      kbhit();
   }

   /* --- WRITE AVERAGES, MAXIMUMS, AND MINIMUMS --- */

   avg_get(scans,start/2,last/2);
   j = sprintf(msg,"AVG ");
   for(i = start/2; i < last/2;i++)
      {
      if(count < 9)
         j += sprintf(msg+j," %7.4f",avg[i]);
      else if (count <11)
         j += sprintf(msg+j," %6.3f",avg[i]);
      else if(count < 13)
         j += sprintf(msg+j," %5.2f",avg[i]);
      else
         j += sprintf(msg+j,"%4.1f",avg[i]);
      }
   fprintf(out_fil,"%s\n\r",msg);
```

```
   j = sprintf(msg,"MIN ");
   for(i = start/2; i < last/2;i++)
      {
      if(count < 9)
         j += sprintf(msg+j," %7.4f",mn[i]);
      else if (count <11)
         j += sprintf(msg+j," %6.3f",mn[i]);
      else if(count < 13)
         j += sprintf(msg+j," %5.2f",mn[i]);
      else
         j += sprintf(msg+j,"%4.1f",mn[i]);
      }
   fprintf(out_fil,"%s\n\r",msg);

   j = sprintf(msg,"MAX ");
   for(i = start/2; i < last/2;i++)
      {
      if(count < 9)
         j += sprintf(msg+j," %7.4f",mx[i]);
      else if (count <11)
         j += sprintf(msg+j," %6.3f",mx[i]);
      else if(count < 13)
         j += sprintf(msg+j," %5.2f",mx[i]);
      else
         j += sprintf(msg+j,"%4.1f",mx[i]);
      }
   fprintf(out_fil,"%s\n\r",msg);
}
```

```
/*---- READ AND WRITE VOLTAGE SCAN DATA (SCROLL ) ---------*/

volt_scrll(scans,chans,start,last,count)
unsigned start,last,count;
int scans,chans;
{
   unsigned char scan_s[MAX_SCAN];
   int chnl,j,k,i,scan,adval;
   float volt;
   char c,msg[79];
   int mode,fore,back,columns,act_page;
   long position;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      }
   else
      {
      fore = WHITE;
      back = BLACK;
      }
   scatrect(12,0,24,79,fore,back);
   sccurset(24,0);
   scpgcur(1,12,13,0);
   scpscrol(0,NORMAL,12,1,24,79,SCR_UP);

   /* --- CLEAR AVERAGE, MAX, MIN ARRAYS --- */

   arry_clr();

   /* --- READ EACH SCAN OF DATA --- */

   for(k = 1;k < scans + 1;k++)
      {
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         {
         err_wn("ERROR:data read failure");
         break;
         }
```

```
/* --- DISPLAY VOLTAGES WITH SCAN COUNT IN DATA --- */

else if(head.scan_flg != TEL_FALSE)
   {
   scan = scan_s[chans-2]*256 + scan_s[chans-3];
   if(scan != k)
      {
      scan = dat_corr_scroll(k,scans,chans,
         count,VOLT,fore,back);
      k = scan;
      position = ftell(data_fil) - chans;
      if(position == -1L)
         err_wn("ERROR:could not find file position");
      if(fseek(data_fil,position,SEEK_SET) != 0)
         err_wn("ERROR:could not reposition file");
      if(fread((char *)scan_s,1,chans,data_fil) == NULL)
         err_wn("ERROR:data read failure");
      }
   j = sprintf(msg,"%4d",scan);
   for(i = start; i < last;i++)
      {
      adval = scan_s[i+1]*256 + scan_s[i];
      volt = (adval-2048)*0.00244;
      avg[i/2] = avg[i/2] + volt;
      maxmin(i/2,volt);
      if(count < 9)
         j += sprintf(msg+j," %7.4f",volt);
      else if (count <11)
         j += sprintf(msg+j," %6.3f",volt);
      else if(count < 13)
         j += sprintf(msg+j," %5.2f",volt);
      else
         j += sprintf(msg+j,"%4.1f",volt);
      i++;
      }
   scdspmsg(21,1,fore,back,msg);
   scpscrol(1,NORMAL,12,1,21,79,SCR_UP);
   kbhit();
   }
```

```
   /* DISPLAY VOLTAGES AND USE LOOP COUNT FOR SCAN COUNT */

   else
      {
      j = sprintf(msg,"%4d ",k);
      for(i = start; i < last;i++)
         {
         adval = scan_s[i+1]*256 + scan_s[i];
         volt = (adval-2048)*0.00244;
         avg[i/2] = avg[i/2] + volt;
         maxmin(i/2,volt);
         if(count < 9)
            j += sprintf(msg+j," %7.4f",volt);
         else if (count <11)
            j += sprintf(msg+j," %6.3f",volt);
         else if(count < 13)
            j += sprintf(msg+j," %5.2f",volt);
         else
            j += sprintf(msg+j,"%4.1f",volt);
         i++;
         }
      scdspmsg(21,1,fore,back,msg);
      scpscrol(1,NORMAL,12,1,21,79,SCR_UP);
      kbhit();
      }
}
/* --- DISPLAY AVERAGES, MAXIMUMS, AND MINIMUMS --- */

avg_get(scans,start/2,last/2);
j = sprintf(msg,"AVG ");
for(i = start/2; i < last/2;i++)
   {
   if(count < 9)
      j += sprintf(msg+j," %7.4f",avg[i]);
   else if (count <11)
      j += sprintf(msg+j," %6.3f",avg[i]);
   else if(count < 13)
      j += sprintf(msg+j," %5.2f",avg[i]);
   else
      j += sprintf(msg+j,"%4.1f",avg[i]);
   }
scdspmsg(22,1,fore,back,msg);

j = sprintf(msg,"MIN ");
for(i = start/2; i < last/2;i++)
   {
   if(count < 9)
      j += sprintf(msg+j," %7.4f",mn[i]);
   else if (count <11)
      j += sprintf(msg+j," %6.3f",mn[i]);
   else if(count < 13)
      j += sprintf(msg+j," %5.2f",mn[i]);
   else
      j += sprintf(msg+j,"%4.1f",mn[i]);
   }
```

```
   scdspmsg(23,1,fore,back,msg);

   j = sprintf(msg,"MAX ");
   for(i = start/2; i < last/2;i++)
      {
      if(count < 9)
         j += sprintf(msg+j," %7.4f",mx[i]);
      else if (count <11)
         j += sprintf(msg+j," %6.3f",mx[i]);
      else if(count < 13)
         j += sprintf(msg+j," %5.2f",mx[i]);
      else
         j += sprintf(msg+j,"%4.1f",mx[i]);
      }
   scdspmsg(24,1,fore,back,msg);

scpgcur(0,12,13,0);
}


/*--------- CHECK FOR MAXIMUM AND MINIMUM VALUES -------------*/

void maxmin(chan,volt)
int chan;
float volt;
{
                        /* If voltage > than previous
                           maximum, save new value */
   if(volt > mx[chan])
      mx[chan] = volt;
                        /* If voltage < than previous
                           minimum, save new value */
   if(volt < mn[chan])
      mn[chan] = volt;
}

/*-------- DETERMINE THE AVERAGES FOR EACH CHANNEL --------*/

void avg_get(scans,start,last)
int scans,start,last;
{
   int i;

   for(i = start;i < last+1;i++)
      avg[i] = avg[i]/scans;
}
```

```
/*--------- CLEAR THE AVERAGE, MAX, AND MIN ARRAYS --------*/

void arry_clr()
{
   int i;

   for(i = 0;i < 17;i++)
      {
      avg[i] = 0.0;
      mn[i] = 5.0;
      mx[i] = -5.0;
      }
}

/*----------- PRINT VOLT TITLES ---------------------------*/

void volt_ttl(frst_ch,count)
unsigned frst_ch,count;
{
   int i;

   /* --- WRITES LABEL AND EACH CHANNEL TO BE WRITTEN --- */

   fprintf(out_fil,"\nSCAN");
   for(i = (count+frst_ch-1); i >= frst_ch;i--)
       {
       if(count < 9)
          fprintf(out_fil,"  %2d    ",i);
       else if(count < 11)
          fprintf(out_fil,"  %2d   ",i);
       else if(count < 13)
          fprintf(out_fil,"  %2d  ",i);
       else
          fprintf(out_fil," %2d ",i);
       }
   fprintf(out_fil,"\n");
}
```

```
/*----------- PRINT VOLT TITLES (BOX DISPLAY) -------------*/

void volt_ttl_box(frst_ch,count)
unsigned frst_ch,count;
{
   int i,j;
   char msg[79];
   int mode,fore,back,columns,act_page;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = BLACK;
      back = WHITE;
      }
   else
      {
      fore = WHITE;
      back = RED;
      }

   /* --- DRAW THE TITLE BOX --- */

   scbox(10,0,11,79,0,-1,NORMAL);
   scatrect(10,0,11,79,fore,back);

   /* DISPLAY SCAN LABEL AND EACH CHANNEL TO BE DISPLAYED */

   j = sprintf(msg,"SCAN");
   for(i = (count+frst_ch-1); i >= frst_ch;i--)
       {
       if(count < 9)
          j += sprintf(msg+j,"  %2d    ",i);
       else if(count < 11)
          j += sprintf(msg+j,"  %2d   ",i);
       else if(count < 13)
          j += sprintf(msg+j,"  %2d  ",i);
       else
          j += sprintf(msg+j," %2d ",i);
       }
   scdspmsg(11,1,fore,back,msg);
}
```

```
/*----------- PRINT HEXADECIMAL TITLES --------------------*/

void hex_ttl(frst_ch,count)
unsigned frst_ch,count;
{
   int i,j;
   char msg[80];

   /* DISPLAY SCAN LABEL AND EACH CHANNEL TO BE DISPLAYED */

   if(count != 16)
      j = sprintf(msg,"SCAN ");
   else
      j = 0;
   for(i = (count+frst_ch-1); i >= frst_ch;i--)
       {
       j += sprintf(msg+j," %2d  ",i);
       }
   fprintf(out_fil,"%s\n",msg);
}
```

```
/*----------- PRINT HEXADECIMAL TITLES (BOX DISPLAY) ------*/

void hex_ttl_box(frst_ch,count)
unsigned frst_ch,count;
{
   int i,j;
   char msg[80];
   int mode,fore,back,columns,act_page;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = BLACK;
      back = WHITE;
      }
   else
      {
      fore = WHITE;
      back = RED;
      }

   /* --- DRAW THE TITLE BOX --- */

   scbox(10,0,11,79,0,-1,NORMAL);
   scatrect(10,0,11,79,fore,back);

   /* DISPLAY SCAN LABEL AND EACH CHANNEL TO BE DISPLAYED */

   if(count != 16)
      j = sprintf(msg,"SCAN ");
   else
      j = 0;
   for(i = (count+frst_ch-1); i >= frst_ch;i--)
       {
       j += sprintf(msg+j," %2d  ",i);
      }
   scdspmsg(11,1,fore,back,msg);
}
```

```
/*-------------- CORRECT ANY MISSING DATA -----------------*/

dat_corr(no_scan,t_scan,no_byt,count,flag)
int no_scan,t_scan,no_byt,count,flag;
{
   unsigned char c,hi,lo;
   int i,j,k,scan;
   float volt;
   char msg[80];
   long position;

   /* --- DISPLAY WARNING MESSAGE --- */

   sprintf(msg,"WARNING:data lost at scan %d",no_scan);
   err_wn(msg);
   scan = 0;

   /* --- BACKUP FILE ONE SCAN LENGTH --- */

   position = ftell(data_fil) - no_byt;
   if(position == -1L)
      err_wn("ERROR:file position failure");
   if(fseek(data_fil,position,SEEK_SET) != 0)
      err_wn("ERROR:file reposition failure");

   /* --- SEARCH FOR THE NEXT END-OF-SCAN SEPARATOR --- */

   while(scan < no_scan || scan > t_scan)
      {
      do
         {
         c = fgetc(data_fil);
         }
      while(c != 0xFF);
      position = ftell(data_fil) - 3;
      if(position == -1L)
         err_wn("ERROR:file position failure");
      if(fseek(data_fil,position,SEEK_SET) != 0)
         err_wn("ERROR:file reposition failure");

      /* --- READ THE SCAN NUMBER --- */

      lo = fgetc(data_fil);
      hi = fgetc(data_fil);
      scan = hi*256 + lo;
      fgetc(data_fil);
      }
```

```
   /* IF VALID SCAN NUMBER, PAD THE DATA WITH ZEROS */

   for(k = no_scan;k < scan;k++)
      {
      if(flag == HEX)
         {
         if(count != 16)
              j = sprintf(msg,"%4d ",k);
         else
            j = 0;
           for(i = 0; i < count;i++)
               {
               j += sprintf(msg+j,"0800 ");
               }
         fprintf(out_fil,"%s\n\r",msg);
         }
      else if(flag == BIN)
         {
         for(j = 0; j < count;j++)
            {
            fputc(0x00,out_fil);
            fputc(0x08,out_fil);
            }
         }
      else if(flag == PLT)
         {
         for(j = 0; j < count;j++)
            {
            fputc(0x00,out_fil);
            fputc(0x08,out_fil);
            }
         fputc(0xFF,out_fil);
         }
      else
         {
         j = sprintf(msg,"%4d",k);
         for(i = 0; i < count;i++)
            {
            volt = 0.00;
            avg[i/2] = avg[i/2] + volt;
            maxmin(i/2,volt);
            if(count < 9)
               j += sprintf(msg+j," %7.4f",volt);
            else if (count <11)
               j += sprintf(msg+j," %6.3f",volt);
            else if(count < 13)
               j += sprintf(msg+j," %5.2f",volt);
            else
               j += sprintf(msg+j,"%4.1f",volt);
            }
         fprintf(out_fil,"%s\n\r",msg);
         }
      }
   return scan;
}
```

```
/*---------- CORRECT ANY MISSING DATA (SCROLL) -------------*/

dat_corr_scroll(no_scan,t_scan,no_byt,count,flag,fore,back)
int no_scan,t_scan,no_byt,count,flag,fore,back;
{
   unsigned char c,hi,lo;
   int i,j,k,scan;
   float volt;
   char msg[80];
   long position;


   /* --- DISPLAY WARNING MESSAGE --- */

   sprintf(msg,"WARNING:data lost at scan %d",no_scan);
   err_wn(msg);
   scan = 0;

   /* --- BACKUP FILE ONE SCAN LENGTH --- */

   position = ftell(data_fil) - no_byt;
   if(position == -1L)
      err_wn("ERROR:file position failure");
   if(fseek(data_fil,position,SEEK_SET) != 0)
      err_wn("ERROR:file reposition failure");

   /* --- SEARCH FOR THE NEXT END-OF-SCAN SEPARATOR --- */

   while(scan < no_scan || scan > t_scan)
      {
      do
         {
         c = fgetc(data_fil);
         }
      while(c != 0xFF);
      position = ftell(data_fil) - 3;
      if(position == -1L)
         err_wn("ERROR:file position failure");
      if(fseek(data_fil,position,SEEK_SET) != 0)
         err_wn("ERROR:file reposition failure");

      /* --- READ THE SCAN NUMBER --- */

      lo = fgetc(data_fil);
      hi = fgetc(data_fil);
      scan = hi*256 + lo;
      fgetc(data_fil);
      }
```

```
   /* IF VALID SCAN NUMBER, PAD THE DATA WITH ZEROS */

   for(k = no_scan;k < scan;k++)
      {
      if(flag == HEX)
         {
         if(count != 16)
              j = sprintf(msg,"%4d ",k);
         else
            j = 0;
           for(i = 0; i < count;i++)
               {
               j += sprintf(msg+j,"0800 ");
               }
         scdspmsg(24,1,fore,back,msg);
         scpscrol(1,NORMAL,12,1,24,79,SCR_UP);
         kbhit();
         }
      else
         {
         j = sprintf(msg,"%4d",k);
         for(i = 0; i < count;i++)
            {
            volt = 0.00;
            avg[i/2] = avg[i/2] + volt;
            maxmin(i/2,volt);
            if(count < 9)
               j += sprintf(msg+j," %7.4f",volt);
            else if (count <11)
               j += sprintf(msg+j," %6.3f",volt);
            else if(count < 13)
               j += sprintf(msg+j," %5.2f",volt);
            else
               j += sprintf(msg+j,"%4.1f",volt);
            }
         scdspmsg(21,1,fore,back,msg);
         scpscrol(1,NORMAL,12,1,21,79,SCR_UP);
         kbhit();
         }
   }
   return scan;
}
```

GEN.C LISTING

```
/*--------------------------- GEN.H ----------------------*/

/* REV 2.00 7/6/88 by gkm - functions for printing titles,
                            file handling, and entry input */
/* REV 2.10 7/12/88 by gkm - added extended screen functions
                             and pop-up windows to all screen
                             functions                      */
/* REV 2.11 7/13/88 by gkm - added pop-up window messages  */

#define TRUE   1
#define FALSE   0

char scrn_buf[4][42][2];    /* Screen storage buffer */

/*------------------- FUNCTION DECLARATIONS ---------------*/

             /* FILE HANDLING FUNCTIONS */

extern FILE *fil_open(void);
extern FILE *fil_open_wn(void);
extern FILE *fil_creat(void);
extern FILE *fil_creat_wn(void);

               /* SCREEN FUNCTIONS */

extern void signon(char *,char *,char *);
extern void signon_box(char *,char *,char *);
extern void hit_cont(void);
extern void hit_cont_wn(void);
extern void msg_wn_on(char *);
extern void msg_wn_off(void);
extern void err_wn(char *);

               /* ENTRY FUNCTIONS */

extern char get_char(char,char);
extern int get_no(int,int);
extern int get_no_wn(char *,int,int);
extern char get_yn_wn(char *);
```

```
/*------------ GENERAL PURPOSE FUNCTIONS ------------------*/

/* REV 2.00 7/6/88 by gkm - functions for printing titles,
                           file handling, and entry input */
/* REV 2.10 7/12/88 by gkm - added extended screen functions
                             and pop-up windows to all screen
                             functions                      */
/* REV 2.11 7/13/88 by gkm - added pop-up window messages  */

/* NOTE: All simple functions display information at the
      current cursor location, working from the top of a
      cleared screen downwards.  All functions with _box
      or _wn in its label use a screen management system.
      The upper 5 lines of the screen are reserved for
      the program title.  The upper-left area is used to
      display status messages, which appear as pop-up
      windows.  The upper-right area is used to display
      error messages, which also appear as pop-up windows.
      The bottom 5 lines of the screen are reserved for
      program entry displays.  These appear as pop-up
      windows, saving any data written in that area.  This
      area is also used to display the operational
      parameters for the program that is running.  Lines
      6 through 19 are used as the data area for the
      program.                                          */

#include <stdio.h>
#include <dos.h>
#include <bscreen.h>
#include <bwindow.h>
#include <bfile.h>
#include "gen.h"
```

```
/****************** FILE HANDLING FUNCTIONS **************/

/*--------------- OPEN A FILE FOR READING -----------------*/

FILE *fil_open()
{
   /* --- RETURNS A POINTER TO THE OPENED FILE --- */

   FILE *read_fil,*fopen();
   char fil[67];

   do
      {
      /* --- ENTER FILE NAME  --- */

      printf("Enter file name for reading data: \0");
      gets(fil);

      /* --- SEE IF FILE EXISTS --- */

      if((read_fil = fopen(fil,"rb")) == NULL)
         printf("Cannot find %s",fil);
      }
   while(read_fil == NULL);

   return(read_fil);
}
```

```
/*-------- OPEN A FILE FOR READING (WINDOW MODE) ---------*/

FILE *fil_open_wn()
{
   /* --- RETURNS A POINTER TO THE OPENED FILE --- */

   FILE *read_fil,*fopen();
   char fil[67];
   int scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int w_row,w_col,scan,cursor_was_off,row,col,high,low;

   /* --- CHECK MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = BLACK;
      back = WHITE;
      scrn_mode = TRUE;
      }

   /* --- CREATE THE SCREEN WINDOW --- */

   pwin = wncreate(2,70,REVERSE);
   bord.type = 1;
   if(scrn_mode == TRUE)
      bord.attr = RED;
   else
      bord.attr = BLACK;

   location.dev = scmode(&mode,&columns,&act_page);
   location.page = act_page;
   location.corner.row = 22;
   location.corner.col = 5;

   scpage(act_page);
   cursor_was_off = sccurst(&row,&col,&high,&low);
   wndsplay(pwin,&location,&bord);
   wnwrap(0,"Enter file name for reading data: \0",-1,-1,CHARS_ONLY);
   wncurpos(&w_row,&w_col);
```

```
   do
      {

      /* --- ENTER THE FILE NAME --- */

      wnquery(fil,sizeof(fil),&scan);

      /* --- SEE IF FILE EXISTS --- */

      if((read_fil = fopen(fil,"rb")) == NULL)
         {
         if(scrn_mode == TRUE)
            fore = RED;
         scdspmsg(23,6,fore,back,"Cannot find ");
         if(scrn_mode == TRUE)
            fore = BLUE;
         scdspmsg(23,18,fore,back,fil);
         wncurmov(w_row,w_col);
         }
      }
   while(read_fil == NULL);

   /* --- REMOVE THE WINDOW --- */

   wnremove(pwin);
   sccurset(row,col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
   wndstroy(pwin);

   return(read_fil);
}
```

```
/*--------------- OPEN A FILE FOR WRITING -----------------*/

FILE *fil_creat()
{
   /* --- RETURNS A POINTER TO THE OPENED FILE  --- */

   FILE *writ_fil,*fopen();
   char c,fil[67];

   do
      {
      /* --- ENTER FILE NAME --- */

      printf("Enter file name for writing data: ");
      gets(fil);

      /* --- SEE IF FILE EXISTS --- */

      if(fopen(fil,"r") == NULL)   /* No, then open it */
         {
         if((writ_fil = fopen(fil,"wb")) == NULL)
            printf("Cannot open %s for writing\n",fil);
         }

      /* IF FILE EXISTS, ASK IF TO OVERWRITE EXISTING DATA */
      else
         {
         printf("%s already exists. Overwrite?[N]\b\b",fil);
         do
            {
            c = toupper(getch());
            }
         while(c != 'N' && c != 'Y');
         printf("%c]\n",c);

         if(c == 'Y')
            {
            if((writ_fil = fopen(fil,"wb")) == NULL)
               printf("Could not open %s for writing\n",fil);

       }
         else
            writ_fil = NULL;
         }
      }
   while(writ_fil == NULL);
   return(writ_fil);
}
```

```
/*-------- OPEN A FILE FOR WRITING (WINDOW MODE) ---------*/

FILE *fil_creat_wn()
{
   /* --- RETURNS A POINTER TO THE OPENED FILE --- */

   FILE *fopen(),*writ_fil;
   int handle;
   char c,fil[67];
   int error,scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int w_row,w_col,scan,cursor_was_off,row,col,high,low;

   /* --- CHECK MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = BLACK;
      back = WHITE;
      scrn_mode = TRUE;
      }

   /* --- CREATE THE SCREEN WINDOW --- */

   pwin = wncreate(2,70,REVERSE);
   bord.type = 1;
   if(scrn_mode == TRUE)
      bord.attr = RED;
   else
      bord.attr = BLACK;

   location.dev = scmode(&mode,&columns,&act_page);
   location.page = act_page;
   location.corner.row = 22;
   location.corner.col = 5;

   scpage(act_page);
   cursor_was_off = sccurst(&row,&col,&high,&low);
   wndsplay(pwin,&location,&bord);
   wnwrap(0,"Enter file name for writing data: ",-1,-1,CHARS_ONLY);
   wncurpos(&w_row,&w_col);
```

```
do
   {

   /* --- ENTER THE FILE NAME --- */

   wnquery(fil,sizeof(fil),&scan);

   /* --- SEE IF FILE EXISTS --- */

   error = flnew(fil,&handle,AT_GENERAL);
   switch(error)
      {
      case 0:              /* Successful opening */
         flclose(handle);
         if((writ_fil = fopen(fil,"wb")) == NULL)
            {
            if(scrn_mode == TRUE)
               fore = BLUE;
            scdspmsg(23,6,fore,back,fil);
            wncurmov(w_row,w_col);
            if(scrn_mode == TRUE)
               fore = RED;
            scdspmsg(23,21,fore,back,"cannot be opened.");
            }
         break;
      case 3:
         if(scrn_mode == TRUE)
            fore = RED;
         scdspmsg(23,6,fore,back,"Invalid path!");
         writ_fil = NULL;
         break;
      case 4:
         if(scrn_mode == TRUE)
            fore = RED;
         scdspmsg(23,6,fore,back,"Too many open files!");
         writ_fil = NULL;
         break;
      case 5:
         if(scrn_mode == TRUE)
            fore = BLUE;
         scdspmsg(23,6,fore,back,fil);
         wncurmov(w_row,w_col);
         if(scrn_mode == TRUE)
            fore = RED;
         scdspmsg(23,21,fore,back,"cannot be opened for writing");
         break;
      case 80:
         if(scrn_mode == TRUE)
            fore = BLUE;
         scdspmsg(23,6,fore,back,fil);
         wncurmov(w_row,w_col);
         if(scrn_mode == TRUE)
            fore = RED;
         scdspmsg(23,21,fore,back,"already exists. Overwrite?(y/n)");
```

```
            do
               {
               c = toupper(getch());
               }
            while(c != 'N' && c != 'Y');

            if(c == 'Y')
               {
               if((writ_fil = fopen(fil,"wb")) == NULL)
                  {
                  if(scrn_mode == TRUE)
                     fore = BLUE;
                  scdspmsg(23,6,fore,back,fil);
                  wncurmov(w_row,w_col);
                  if(scrn_mode == TRUE)
                     fore = RED;
                  scdspmsg(23,21,fore,back,"cannot be opened.");
                  }
               }
            else
               {
               scclrmsg(23,6,50);
               scclrmsg(22,39,15);
               writ_fil = NULL;
               }
            break;
         default:
            writ_fil = NULL;
            break;
         }
      }
   while(writ_fil == NULL);

   /* --- REMOVE THE WINDOW --- */

   wnremove(pwin);
   sccurset(row,col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
   wndstroy(pwin);

   return(writ_fil);
}
```

```
/********************** SCREEN FUNCTIONS *****************/

/*------------------- DISPLAY SIGNON MESSAGES --------------*/

void signon(prgm_ttl,clnt_ttl,ver_ttl)
char *prgm_ttl,*clnt_ttl,*ver_ttl;
{
   system("cls");
   puts(prgm_ttl);
   puts(clnt_ttl);
   puts(ver_ttl);
}

/* --- DISPLAY SIGNON MESSAGES (BOX DISPLAY) --- */

void signon_box(prgm_ttl,clnt_ttl,ver_ttl)
char *prgm_ttl,*clnt_ttl,*ver_ttl;
{
   int mode,fore,back,columns,act_page;

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = WHITE;
      back = BLACK;
      }
   else
      {
      fore = WHITE;
      back = BLUE;
      }

   scclear();
   scbox(0,0,4,79,0,-1,NORMAL);
   scatrect(0,0,4,79,fore,back);
   scdspmsg(1,1,fore,back,prgm_ttl);
   scdspmsg(2,1,fore,back,clnt_ttl);
   scdspmsg(3,1,fore,back,ver_ttl);
}

/*------------------- HIT ANY KEY TO CONTINUE ---------------*/

void hit_cont()
{
   puts("Hit any key to continue....");
   getch();
}
```

```
/*------- HIT ANY KEY TO CONTINUE (WINDOW) ----------------*/

void hit_cont_wn()
{
   int scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int w_row,w_col,scan,cursor_was_off,row,col,high,low;
   char c,response[5];

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = BLACK;
      back = WHITE;
      scrn_mode = TRUE;
      }
   /* --- CREATE THE WINDOW --- */

   pwin = wncreate(3,20,REVERSE);
   bord.type = 1;
   if(scrn_mode == TRUE)
      bord.attr = RED;
   else
      bord.attr = BLACK;

   location.dev = scmode(&mode,&columns,&act_page);
   location.page = act_page;
   location.corner.row = 2;
   location.corner.col = 5;

   /* --- DISPLAY THE MESSAGE --- */

   scpage(act_page);
   cursor_was_off = sccurst(&row,&col,&high,&low);
   wndsplay(pwin,&location,&bord);
   wnwrap(0,"Hit any key to continue",-1,-1,CHARS_ONLY);
   wncurpos(&w_row,&w_col);
   getch();

   /* --- REMOVE THE WINDOW --- */

   wnremove(pwin);
   sccurset(row,col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
   wndstroy(pwin);
}
```

```
/*--------- DISPLAY ERROR MESSAGE (WINDOW) ----------------*/

void err_wn(msg)
char *msg;
{

   int scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int w_row,w_col,scan,cursor_was_off,row,col,high,low;

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = RED;
      back = BLACK;
      scrn_mode = TRUE;
      }

   /* --- CREATE THE WINDOW --- */

   pwin = wncreate(3,20,REVERSE);
   bord.type = 1;
   if(scrn_mode == TRUE)
      bord.attr = RED;
   else
      bord.attr = BLACK;

   location.dev = scmode(&mode,&columns,&act_page);
   location.page = act_page;
   location.corner.row = 2;
   location.corner.col = 55;

   /* --- DISPLAY THE MESSAGE --- */

   scpage(act_page);
   cursor_was_off = sccurst(&row,&col,&high,&low);
   wndsplay(pwin,&location,&bord);
   wnwrap(0,msg,-1,-1,CHARS_ONLY);
   wncurpos(&w_row,&w_col);
   hit_cont_wn();
```

```
   /* --- REMOVE THE WINDOW --- */

   wnremove(pwin);
   sccurset(row,col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
   wndstroy(pwin);

}

/*------------- DISPLAY A MESSAGE (WINDOW) ----------------*/

void msg_wn_on(msg)
char *msg;
{
   int mode,fore,back,columns,act_page;

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = BLACK;
      back = WHITE;
      }
   else
      {
      fore = BLACK;
      back = GREEN;
      }
   scrdrect(10,19,12,59,&scrn_buf[0][0][0],CHAR_ATTR);
   scbox(10,19,12,59,0,-1,NORMAL);
   scatrect(10,19,12,59,fore,back);
   scdspmsg(11,20,fore,back,msg);
}

/*----------- REMOVE THE DISPLAYED MESSAGE (WINDOW) -------*/

void msg_wn_off()
{
   scwrrect(10,19,12,59,&scrn_buf[0][0][0],0,0,CHAR_ATTR);
}
```

```
/********************** ENTRY FUNCTIONS ******************/

/*---- GET A CHARACTER ENTRY AND CHECK FOR VALID OPTION ---*/

char get_char(lower,upper)
char upper,lower;
{
   char c;

   do
      {
      c = toupper(getch());
      }
   while(c < lower || c > upper);
   printf("%c\n",c);
   return (c);
}

/*-------------------- GET A NUMBER ENTRY ------------------*/

int get_no(lower,upper)
int lower,upper;
{
   int no;
   char msg[20];

   do
      {
      gets(msg);
      no = atoi(msg);
      }
   while(no < lower || no > upper);
   return no;
}
```

```
/*--------------- GET A NUMBER ENTRY (WINDOW MODE) --------*/

get_no_wn(msg,lower,upper)
char *msg;
int lower,upper;
{

   int scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int num,w_row,w_col,scan,cursor_was_off,row,col,high,low;
   char response[10];

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = BLACK;
      back = WHITE;
      scrn_mode = TRUE;
      }

   /* --- CREATE THE WINDOW --- */

   pwin = wncreate(2,70,REVERSE);
   bord.type = 1;
   if(scrn_mode == TRUE)
      bord.attr = RED;
   else
      bord.attr = BLACK;

   location.dev = scmode(&mode,&columns,&act_page);
   location.page = act_page;
   location.corner.row = 22;
   location.corner.col = 5;

   scpage(act_page);
   cursor_was_off = sccurst(&row,&col,&high,&low);
   wndsplay(pwin,&location,&bord);
   wnwrap(0,msg,-1,-1,CHARS_ONLY);
   wncurpos(&w_row,&w_col);
```

```
   /* --- GET THE CHECKED NUMBER ENTRY --- */

   do
      {
      wnquery(response,sizeof(response),&scan);
      num = atoi(response);
      wncurmov(w_row,w_col);
      }
   while(num < lower || num > upper);

   /* --- REMOVE THE WINDOW --- */

   wnremove(pwin);
   sccurset(row,col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
   wndstroy(pwin);

   return(num);
}



/*------- PROMPT USER FOR YES OR NO ------------------*/

char get_yn_wn(msg)
char *msg;
{

   int scrn_mode,mode,fore,back,columns,act_page;
   BWINDOW *pwin;
   BORDER bord;
   WHERE location;
   int w_row,w_col,scan,cursor_was_off,row,col,high,low;
   char c,response[5];

   /* --- DETERMINE MONITOR TYPE --- */

   if(scmode(&mode,&columns,&act_page) == MONO)
      {
      fore = -1;
      back = -1;
      scrn_mode = FALSE;
      }
   else
      {
      fore = BLACK;
      back = WHITE;
      scrn_mode = TRUE;
      }
```

```
    /* --- CREATE THE WINDOW --- */

    pwin = wncreate(3,20,REVERSE);
    bord.type = 1;
    if(scrn_mode == TRUE)
       bord.attr = RED;
    else
       bord.attr = BLACK;

    location.dev = scmode(&mode,&columns,&act_page);
    location.page = act_page;
    location.corner.row = 2;
    location.corner.col = 5;

    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);
    wndsplay(pwin,&location,&bord);
    wnwrap(0,msg,-1,-1,CHARS_ONLY);
    wncurpos(&w_row,&w_col);

    /* --- LOOP UNTIL A YES OR NO IS ENTERED --- */

    do
       {
       wnquery(response,sizeof(response),&scan);
       c = toupper(response[0]);
       wncurmov(w_row,w_col);
       }
    while(c != 'Y' && c != 'N');

    /* --- REMOVE THE WINDOW --- */

    wnremove(pwin);
    sccurset(row,col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
    wndstroy(pwin);

    return(c);
}
```