Energy, Mines and Resources

Energie, Mines et Ressources

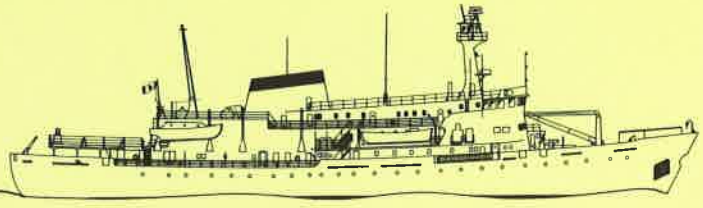
BEDFORD INSTITUTE OF OCEANOGRAPHY

INSTITUT OCÉANOGRAPHIQUE DE BEDFORD

MINAV:

Mini-Ranger III Position Logging Syst

Installation and Operating Guide



Canada

MINAV:

Mini-Ranger III Position Logging System

Installation and Operating Guide

by

B. D. Loncarevic,
Everett Coldwell,
Ross-Allen McKenna
and
Dave Hackett

Geological Survey of Canada
Bedford Institute of Oceanography
Dartmouth, N.S., Canada, B2Y 4A2

May 1, 1986
Atlantic Geoscience Centre
Regional Reconnaissance Internal Report

Table of Contents

Introduction

MINAV (MIni-Ranger NAVigation) is a software package written on the Compaq microcomputer in Turbo-Pascal (Version 3.01A with 8087 Math Support). It is used to calculate a geographic position from up to four ranges from known Mini-Ranger stations. The positions are displayed on the Compaq's screen and logged to its disc. A plot of the ship's track is also available.

The program is run on a COMPAQ portable microcomputer which is connected to a QUBIT interface, which in turn is connected to a Mini-Ranger III.

Project Organization

This project was initiated by Coastal Group of EMG (D. L. Forbes) and was carried out under the direction of B.D. Loncarevic. D. Hackett converted position calculations from Fortran to Pascal. R. McKenna interfaced QUBIT 2784 to the Compaq via GPIB and developed the overall structure of the program. E. Coldwell completed the user interface, developed the plot routines and wrote this manual.

1. Materials Required

Compaq Portable Computer with:
- National Instruments GPIB <-> PC IEEE 488 interface board
- 8087 Math Processor

Qubit 2784 with a Mini-Ranger III interface card

Mini-Ranger III with positioning stations set up

GPIB cable

4 code MRS III connector

Mini-Ranger cable

3 power cords

MINAV system disc

Formatted data discs

Graphics Card (Optional)

External Monitor (Optional)

2. System Description

When MINAV is started, QUBIT is initialized and its internal clock is set from the internal clock of the COMPAQ. Qubit then interrogates Mini-Ranger once per second. Once every ten seconds, COMPAQ reads data ranges from QUBIT. The ranges (in meters) to the reference stations are converted to geographic coordinates using routine HRFIX adopted from BIONAV and converted from FORTRAN to Turbo Pascal. These new positions are then either displayed on the screen as text, or displayed graphically. Residuals are displayed to help with Mini-Ranger calibration and to show possible problems with signal reception.

When MINAV is in graph mode, a duplicate of what is on the COMPAQ monitor appears on the optional external monitor. This may be useful for the Bridge when steering lines.

The calculation of position in the MINAV system is based on the WGS72 standard.

3. Installation

Connect the Qubit to the Mini-Ranger via the 4 code MRS III connector and the Mini-Ranger cable. Connect the Qubit (left most socket looking from back) to the Compaq with a GPIB cable. Set Mini-Ranger to external control, assuring that HOLD is not pressed in. Channel settings on the Mini-Ranger have no effect on the Minav system. Turn on the Mini-Ranger and verify that the ranges from reference stations are correctly acquired and that the Mini-Ranger is properly calibrated.

```
*****************************************************************
* Important: Equipment must be turned on in the following order:  *
*            Mini-Ranger, Qubit, Corona.                          *
*                                                                 *
* If at any time the system fails turn off all equipment and start *
* over.                                                           *
*****************************************************************
```

4.1 Initialization

Insert the Mini-Ranger Positioning System Disc in the left drive, and the formatted data disc in the right drive. Turn on or reset the computer. When prompted for the date, enter it (eg: 12-31-86 <Return>). Enter the time, precise to one second before you press <Return>.

The program will be automatically loaded and run once time and date have been entered.

4.2 Setting Initial and Reference Station Positions

The navigation routines will begin by requesting the position of the reference stations. These positions may either be entered manually from

the keyboard or read from the previously saved reference stations on disc. If entered manually you will be asked to enter which reference stations are active. If you state that a station is active, then you will have to input the latitude and longitude of the station as accurately as possible.

If the request for entry of position is followed by a number in parenthesis, then this will be the default if <Return> is pressed without entering a new position. This default is the last latitude or longitude that was loaded from the reference position file or entered from the keyboard for this entry line.

There is a great deal of flexibility in the entry of latitude and longitude in this program. They may be entered as decimal degrees (eg: -63.99999999), degrees and decimal minutes (eg: -63 59.9999) or degrees, minutes and decimal seconds (eg: -63 59 59.999). In entering the position degrees, minutes and seconds are seperated by spaces.

```
*****************************************************************
*         Note: Longitude west of Greenwich must be entered     *
*         as a negative value.                                  *
*****************************************************************
```

After entering the position of reference stations, you will be asked if you wish to save the reference station positions to disc. Type 'Y' to save these positions so they may be reloaded the next time the program is run.

Next you are requested to enter the ship's (antenna) position. This is entered in the same way as was the latitude and longitude of the reference stations and need only be entered to the nearest degree.

If any of the positions are not correctly entered, they may be re-entered by answering; no, to the 'Are coordinates correct?' prompt. If you type 'N' for this prompt, you will again be asked if you wish to manually enter or read from disc the reference station coordinates.

4.3 Data Storage on Disc

You will next be asked for the filename for storing the Mini-Ranger and computed position data. It must have no more than 8 characters, starting with a letter and followed by letters and numbers, followed by a period, and no more than three more characters. Since your data disc is in the rightmost drive the filename must be preceeded by a B:. It is a good idea to use the date in your filename. (eg: B:P860211.Dat).

After a filename has been selected, the station latitudes and longitudes are saved to it. Raw and computed position data from QUBIT are then saved to the file at a rate one every ten seconds. Each record of the data file will contain the following information: date, time, ranges in meters to the reference stations, latitude, longitude, course and speed. Course and speed are calculated by using the current position and the position 6 cycles earlier (usually one minute). In addition, if the distance covered in the last 6 cycles is less than a hundredth of a nautical mile, then planer geometry is used to calculate the distance from which speed is derived. Appendix A contains a description of the format used in the data file.

The data file, which is being written to disc is opened and closed once every minute. This is done to protect data from being lost in case the logging system fails (Eg: power failure, jiggled cables).

The <F2> function key closes the current data file and requests the name of a new data file. This key may be pressed any time that the data is being displayed on the screen, except when MINAV is in the process of outputing a line to the screen.

The program will close the data file when the data disc becomes full and will display a message along with an audible tone to indicate that data is no longer being saved. The audible tone will continue to be sounded every ten seconds until the <F2> key is pressed, even though data is still being displayed to the screen. When this key is pressed you may enter the name of the new data file.

When you are prompted for a filename, you may press the <Retrn> key and MINAV will only display data to the screen and not save it to disc.

4.4 Data Display

Data is displayed to the video display once every ten seconds. Each line on the display contains the following information: date, time, latitude, longitude, course, speed and the residual ranges in meters to the reference stations which are in use. Course and speed are calculated by using the current position and the position 6 cycles earlier (usually one minute).

The residual ranges are the difference between the measured range to the reference station and the range in meters between the calculated position and the reference station. The residual will be positive if the Mini-Ranger range is greater than the calculated range. These ranges may be used for Mini-Ranger calibration and to check for bad signal propagation. The range is blank when Mini-Ranger is not receiving a signal from the station, the station was not requested in

the program initialization or the user toggled the station to off.

The display of positions are updated every ten seconds with new display lines replacing old display lines. Every minute the display is scrolled up and a new line is started. This format results in the minute positions being kept on the screen with the latest ten second position on the last line. See Appendix B for a sample of the information as it is displayed on the video screen.

4.5 Data Collection and Loss

As Mini-Ranger data is being collected, sometimes you may notice that the time displayed is either one second before or one second after the ten second interval. This occurs because of the relative clock drift between the Compaq and the QUBIT.

Also when entering text from the initialization mode or from the graphics mode, or entering graphics mode, data often may be lost in the process. This can result in the first record after such an action having a time that is not on a ten second interval.

This can cause problems for plotting the chart where data is not evenly spaced and can contain gaps.

4.6 Special Keys

Three function keys and four numeric keys are detected and processed by the MINAV program. The keys are <F1>, <F2>, <F10> and the <1>, <2>, <3> and <4> numeric keys.

Because the computations used by the program in determining position are quite involved and take more than a second, if keys are pressed on

the keyboard while a data position is being calculated, the keys are often not detected by the program. Therefore it is suggested that keys be pressed only after a line is displayed to the video screen.

4.7 Station Selection

The numeric keys, <1>, <2>, <3> and <4> can be used to toggle the reception of reference stations between off and on. Whether or not a range residual is displayed on the screen will indicate if a station is being received. Note that when the position of reference stations are retrieved from disc, all of the stations with positions entered for them are enabled. It is not possible to receive ranges from a station which was not selected at the beginning of the program either by entering or loading the positions.

4.8 Graphics Mode

Function key <F10> toggles the computer between text and graphics mode. When the program first starts it is in the text mode. Pressing the <F10> key will clear the screen and ask for minimum and maximum latitudes and longitudes, and the number of grid lines along each axes. The minimum and maximum latitudes and longitudes may be entered using free format as described in section 4.2. In subsequent toggles to graph mode from text mode the user has the option of either using the same grid parameters or to enter new grid limits.

Once the plot parameters have been entered, the screen is cleared and a plot of the receiver's position is drawn, with the ten second data points connected via lines. <F10> reverts MINAV back to text mode.

Sometimes it is desirable to have both the X and the Y axis with the same scale in distance. To do this we can use the following formula:

$$\text{Longitude range} = \frac{225}{145 \times \text{Cos (Latitude Range)}}$$

For 45 degrees latitude the range in longitude should be 2.187 that of the range of latitude.

See Appendix C for an example of a plot generated by MINAV.

4.9 Dumping the Screen to the Printer

Often the user will want to get an immediate hard copy of either the text positions or the chart over the last period of time. This is accomplished by pressing the <PrtSc> key while holding down the <Shift> key. While the printer is printing, data is no longer being acquired from Mini-Ranger. Therefore, before doing this, the user should assure that a record of the positions is no longer needed.

Tests show that printouts take the following lengths of time on two of our printers:

	Minutes:Seconds
HP ThinkJet	3:26
HP 82905A	4:10

On the HP ThinkJet Printer, the graphics printout is considerably smaller for the latitude than it is on the Epson +compatible HP 82905A printer. Therefore the ratio of horizontal to vertical distance will be changed.

4.10 Ending MINAV

To end the program, press <F1>, and respond "Y" when asked if you wish to end the program.

5. Future Extensions

Future software development could extend the usefulness of the system. The QUBIT 2784 already has interface cards for ARGO and Loran-C (Internav LC408) navigation systems. It would be a relatively easy modification to the program to make it adaptable to any one of these systems.

It would often be good to know how good your position fix is. Although the residuals can give you some idea of this, low residuals do not always mean a good fix. Often in navigation systems, DOPs are used. Dops, statistically, give you the probability of error in various directions based on the angle of intersection between the incoming signals from the reference stations. Routines to compute DOPs exist here at BIO (eg: HRFIX from BIONAV) and could be implemented as part of this system.

The display could be enhanced by incorporating 'way points' or a desired track to steer, for the guidance of the helmsman. Multiple screen pages would be useful so either track or text could be kept up-to-date in the background. Multiple screens could also be used to plot different sections of the survey, or to plot the survey on different scales.

A memory buffer for screen dumps could eliminate the long wait and lost data while waiting for a printout.

Finally, the new FALCON IV version of Mini-Ranger has an output which may be directly readable by the microcomputer, thus eliminating the need for the QUBIT 2784 interface box.

Appendix A: Data File Format

The data file is written in text mode (ASCII) and consists of a header line and a series of individual one line records. The header line contains the positions of the reference stations. Each of the non-header records contains a time stamp, ranges, position, and course and speed. If position and velocity cannot be calculated the last 4 fields are not used.

Header Record:

Start	Length	Format	Name	Description
2	13	####.########	lat1	station 1 latitude
16	13	####.########	long1	station 1 longitude
30	13	####.########	lat2	station 2 latitude
44	13	####.########	long2	station 2 longitude
58	13	####.########	lat3	station 3 latitude
72	13	####.########	long3	station 3 longitude
86	13	####.########	lat4	station 4 latitude
100	13	####.########	long4	station 4 longitude

Position Records:

Start	Length	Format	Name	Description
1	2	##	day	day of current month
4	2	##	month	month of current year
7	2	##	hour	hour of current day
10	2	##	minute	minute of current hour
13	2	##	second	second of current minute
16	9	######.##	range1	range to station 1 (meters)
26	9	######.##	range2	range to station 2 (meters)
36	9	######.##	range3	range to station 3 (meters)
46	9	######.##	range4	range to station 4 (meters)
56	3	###	latdeg	calculated latitude (degs)
60	6	##.###	latmin	calculated latitude (mins)
67	4	####	longdeg	calculated longitude (degs)
72	6	##.###	longmin	calculated longitude (mins)
79	6	###.##	course	ship's course (degs true)
87	7	####.##	speed	ship's speed (knots true)

ield Names:

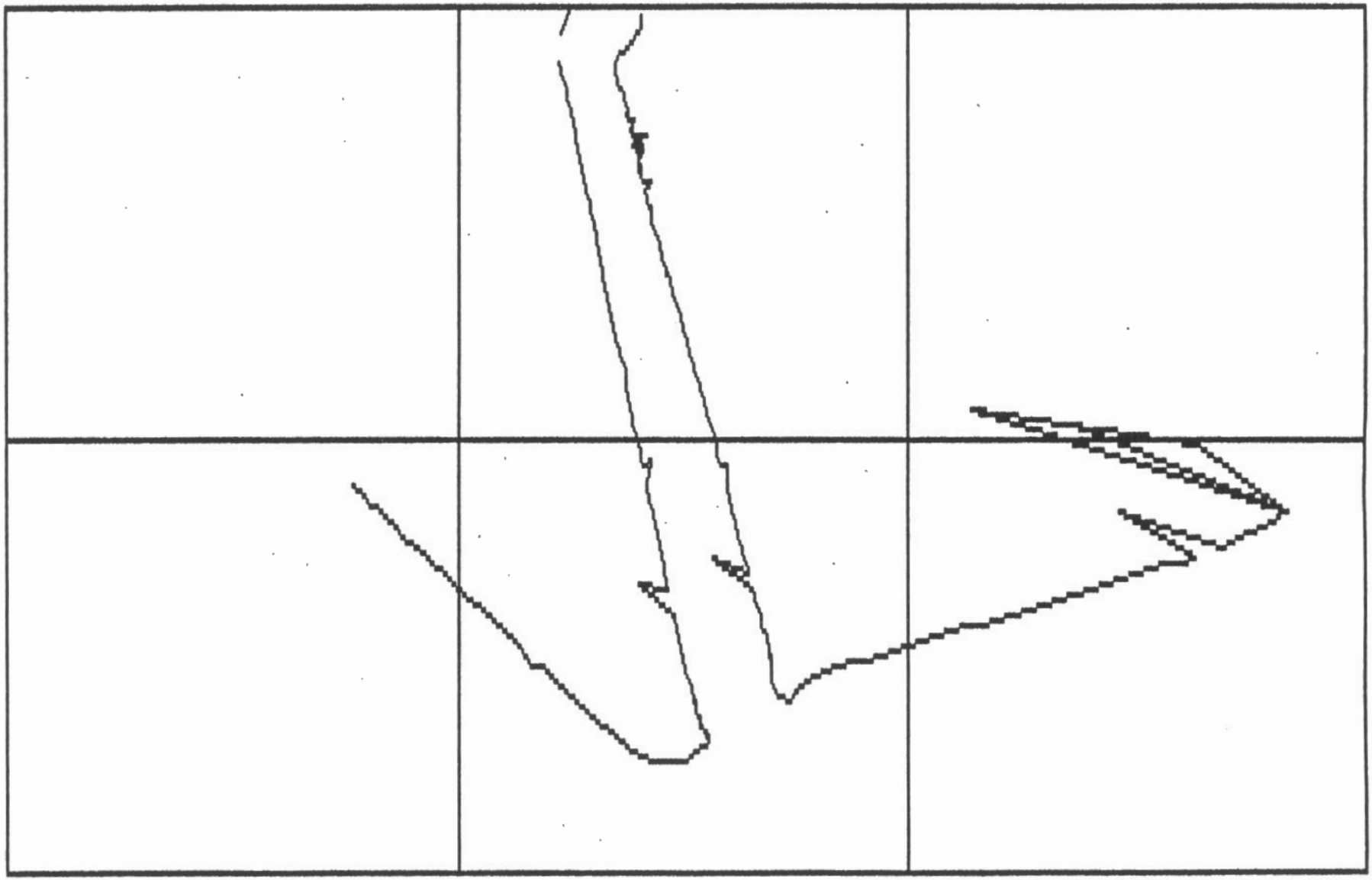
tation 1: Lat	Long		Station 2: Lat	Long		Station 3: Lat	Long		Station 4: Lat	Long
y:mn hr:mn:sc	range1	range2	range3	range4		latitude	longitude	course	speed	

ile Contents:

44.63598639	-63.61467694	44.68096361	-63.67774028		44.72597555	-63.61467694		44.68096361	-63.55161	
2:02 14:52:20	004849.00	004839.00	004844.00	004842.00	44 40.860	-63 36.882	69.63	0.04		
2:02 14:52:22	004848.00	004838.00	004844.00	004844.00	44 40.860	-63 36.883	114.69	0.21		
2:02 14:52:30	004848.00	004839.00	004843.00	004843.00	44 40.860	-63 36.882	186.67	0.16		
2:02 14:52:40	004847.00	004837.00	004845.00	004842.00	44 40.859	-63 36.882	223.84	0.20		
2:02 14:52:50	004960.00	004950.00	004952.00	004948.00	44 40.861	-63 36.880	67.11	0.13		
2:02 14:53:00	004963.00	004948.00	004952.00	004947.00	44 40.862	-63 36.880	33.73	0.15		
2:02 14:53:10	004960.00	004950.00	004959.00	004955.00	44 40.859	-63 36.882	216.27	0.07		
2:02 14:53:20	004957.00	004944.00	004960.00	004953.00	44 40.858	-63 36.884	211.72	0.12		
2:02 14:53:30	004964.00	004945.00	004958.00	004950.00	44 40.860	-63 36.882	303.41	0.03		
2:02 14:53:41	004968.00	004947.00	004957.00	004945.00	44 40.862	-63 36.880	46.84	0.19		
2:02 14:53:51	004850.00	004836.00	004844.00	004841.00	44 40.860	-63 36.882	257.59	0.12		
2:02 14:54:00	004959.00	004947.00	004953.00	004954.00	44 40.860	-63 36.883	245.97	0.15		
2:02 14:54:10	004849.00	004838.00	004844.00	004843.00	44 40.860	-63 36.882	3.78	0.06		
2:02 14:54:20	004960.00	004943.00	004957.00	004957.00	44 40.860	-63 36.886	309.86	0.13		
2:02 14:54:30	004959.00	004945.00	004957.00	004959.00	44 40.859	-63 36.886	252.42	0.16		
2:02 14:54:40	004848.00	004838.00	004845.00	004842.00	44 40.860	-63 36.882	224.24	0.16		
2:02 14:54:50	004848.00	004839.00	004845.00	004842.00	44 40.860	-63 36.882	136.73	0.06		

Graphic Screen Example

Latitude: 44 40 to 44 42 Longitude: -63 39.5 to -63 36.5



25:04 13:59:59 44 40.895' -63 38.733' 314.71 3.20 4 7 10

File Minav.PAS

```
Program MiniRanger_Navigation ( Input, Output );
{$C-} {Disable Cntrl-C}

{ File: Minav.Pas

  Purpose: To provide vessel navigation using a MiniRanger Positioning System,
           connected to the microcomputer via a Qubit intelligent interface.

  Written by: RossAllan McKenna and Everett Coldwell
  Date: March 12, 1986.
  Language: TurboPascal 3.0, with 8087 support.

  Last modified:  February 26/86 - WGS 72 spheroid constants added
                  March 12/86    - File closed every minute          }

Const
   CoordinateFileName = 'MinavCr.Dat';

{   AE = 6378206.4; { Size of Earth - Clarke 1866 Spheroid (NAD 27) }
{   F  = 294.9787;  { Reciprical of flatness of earth - Clarke 1866 (NAD 72) }

   AE = 6378135.; { Size of Earth - WGS 72 Spheroid }
   F  = 298.26;   { Reciprical of flatness of earth - WGS 72 }

   C = 299792458.0;  { Velocity of light in a vacuum (m/s) }

Type
   Line = String [80];
   StnValues = Array [ 1 .. 4 ] of Real;
   OldValues = Array [ 0 .. 6 ] of Real;
   Coords = Record
               Latitude,
               Longitude: Real
               End;                        { Coords }
   StnCoords = Array [1..4] of Coords;
   File_Type = Text;
   FunctionKeyType = ( F1, F2, F10, none );

Var
   Station,
   Qbt: Integer;                           { Global identifier for device. }
   i : Integer;                            { Used in for loops }
   PLat, PLong, Ptime: OldValues;          { Previous latitudes and longitudes. }
   CLat, CLong, Ctime: Real;               { Current latitude and longitude. }
   Weight: StnValues;                      { Station confidence weights. }
   Range : StnValues;                      { Station ranges in meters. }
   RefStn: StnCoords;                      { Reference station coordinates. }
   DataFile: File_Type;                    { File to log data. }
   FileName: String[80];                   { Name of the data file. }
   Outrecord: String[150];                 { Output line to data file }
   OldMin: Integer;                        { Previous value for minutes. }
   OldTenSec : Integer;                    { Previous tens of seconds value. }
   Sp: Real;                               { Ship's speed. }
```

```
   Done: Boolean;                          { Flag to halt program. }
   WriteError: Boolean;                    { Flag indicating disc likely full }
   Errors: Integer;                        { Errors in position fix. }
   Mode: (graphics,text);
   StationOK : Array [1..4] of Boolean;
   FileOk, NoFile : Boolean;

   Lastx, Lasty          : Integer;  { Declarations needed for plotting }
   EastLong, WestLong    : Real;
   NorthLat, SouthLat    : Real;
   NumXlines, NumYlines  : Real;
   XScale, YScale        : Real;
   LimitsSet             : Boolean;
   EastLongStr, WestLongStr    : Line;
   NorthLatStr, SouthLatStr    : Line;

{=================================================================================

{$I Qbt.Prc }
                                           { Initializes the qubit interface, an
                                             includes the TurboPascal handling
                                             routines for the GPIB interface. }

{$I ReadyFile.Prc }
                                           { Prepares the text file to log the
                                             data. }

{$I Init.Prc }
                                           { Obtains the reference station
                                             coordinates, and the ship's initial
                                             location. }

{$I Chart.Prc }                            { Routines to draw map of position }

{$I Sphrd.Prc }
                                           { Determines the geodetic distance (
                                             meters ) between two positions. }

{$I NauticalMile.Prc }
                                           { Returns the distance between two
                                             points on the earth, using planer
                                             geometry }

{$I ReadRn.Prc }
                                           { Obtains the range data ( in meters
                                             from the MiniRanger. }

{$I RangeFix.Prc }
                                           { Produces a navigational position fi
                                             given the most recent observations
                                             distances from the ship to the lora
                                             stations. }

{$I Position.Prc }
                                           { Coordinates the acquisition of a
```

```
                                                  position fix. }

{$I Course.Prc }
                                                { Calculates the ship's course between
                                                  the last two fixes. }

{$I Speed.Prc }
                                                { Determines the ship's speed between
                                                  the last two points. }

{$I CheckMin.Fn }
                                                { Check if it is the start of a new
                                                  minute. }

{$I ReadFunctionKeys.PRC }
                                                { Determine which function key has bee
                                                  pressed. }

{$I Exit.Prc }
                                                { Routine to terminate program. }

{$I OutData.Prc }
                                                { Outputs the position data. }


{$I Headings.Prc }
                                                { Writes the headings for the data. }

{=================================================================================

Begin                                           { Minav }
  ReadyQbt;
{  WriteLn (Lst,#27,'A',#8);  { Initialize HP 82905A printer for graphics }

  Window (1,1,80,25);
  Initialize ( RefStn, Weight, CLat, CLong );
  ReadyFile (NoFile);
  FileOk:= Not NoFile;

  For i:= 1 to 6 do begin
    PLat[i]:= 999;
    Plong[i]:= 999;
    Ptime[i]:= 0;
  End;
  mode:= text;
  LimitsSet:= False;
  OldMin:= -1;
  OldTenSec:= -1;
  Station:= 0;
  Done:= False;
  WriteError:= False;

  ClrScr;
  Headings;
  Window ( 1, 4, 80, 24 );
```

```
GotoXY ( 1, 1 );

Repeat

  OutRecord:= '';
  Repeat
    Case ReadFunctionKeys of
      F1: Exit ( Done );

      F2: Begin
            If FileOk Then Begin
              Close (DataFile);
              Writeln;
              WriteLn ('Disc file "',FileName,'" has been closed.');
            End;
            ReadyFile (NoFile);
            FileOk:= Not NoFile;
          End; {If}

     F10: If mode=text then Begin
            Window (1,1,80,25);
            ClrScr;
            InputChartLimits;
            InitializeChart;
            Window (1,24,80,25);
            GotoXY ( 1, 1 );
            mode:= graphics;
          End
          Else Begin
            TextMode;
            Headings;
            Window ( 1, 4, 80, 24 );
            GotoXY ( 1, 1 );
            mode:= text;
          End;

    End; {Case}

  until NextTenSec or Done;

  If Done then
    WriteLn
  Else Begin
    Position ( CLat, CLong, RefStn, Weight, Errors );

    If Errors <> 0 then Begin
      Write ( '   <<< Unable to obtain fix. >>>' );
      PLat [0]:= 999;
      PLong [0]:= 999
    End {if}
    else Begin
      OutData ( CLat, CLong );
      If Mode = Graphics then
        PlotPosition (CLat,CLong);
      PLat [0]:= CLat;
```

```
      PLong [0]:= CLong;
      Ptime [0]:= CTime;
    End; {Else}

    For i:= 0 to 5 do Begin
      PLat [6-i]:= PLat [5-i];
      PLong[6-i]:= PLong [5-i];
      Ptime[6-i]:= Ptime [5-i];
    End; {For}

    {$I- }
    If FileOk then
      WriteLn ( DataFile, OutRecord );
    {$I+ }

    {                                                                    }
    {>>>> Close file and reopen each minute to save data if system crashes}
    {                                                                    }

    If INT(Ctime/60) <> INT(Ptime[2]/60) then Begin
      Close (DataFile);
      Append (DataFile);
    End; {If}

    If (IOresult <> 0) and (FileName <> '') then Begin
      Close (DataFile);
      WriteLn;
      Write ('End of File or Error writing to disc file "',FileName,'".');
      FileOk:= False;
    End; {If}

    If (Not FileOk) and (FileName <> '') then begin
      Sound (440);
      Delay (1000);
      NoSound;
    End;

  End {If done}

until Done;

TextMode;
IBclr ( Qbt );  { Clear the Qubit }
End. { Minav }
```

```
{$V-}          { relax string length restrictions      }

Const
   { GPIB Commands                                      }
   UNL  =  $3f;                          { GPIB unlisten command          }
   UNT  =  $5f;                          { GPIB untalk command            }
   GTL  =  $01;                          { GPIB go to local               }
   SDC  =  $04;                          { GPIB selected dev clear        }
   PPC  =  $05;                          { GPIB ppoll configure           }
   GET  =  $08;                          { GPIB group execute trigger     }
   TCT  =  $09;                          { GPIB take control              }
   LLO  =  $11;                          { GPIB local lock out            }
   DCL  =  $14;                          { GPIB device clear              }
   PPU  =  $15;                          { GPIB ppoll unconfigure         }
   SPE  =  $18;                          { GPIB serial poll enable        }
   SPD  =  $19;                          { GPIB serial poll disable       }
   PPE  =  $60;                          { GPIB ppoll enable              }
   PPD  =  $70;                          { GPIB ppoll disable             }

   { GPIB status bit vector :                           }
   {       global variable ibsta and wait mask          }

   ERR   = $8000;                        { Error detected                 }
   TIMO  = $4000;                        { Timeout                        }
   UEND  = $2000;                        { EOI or EOS detected            }
   SRQI  = $1000;                        { SRQ detected by CIC            }
   RQS   = $800;                         { Device needs service           }
   CMPL  = $100;                         { I/O completed                  }
   LOK   = $80;                          { Local lockout state            }
   REM   = $40;                          { Remote state                   }
   CIC   = $20;                          { Controller-in-Charge           }
   ATN   = $10;                          { Attention asserted             }
   TACS  = $8;                           { Talker active                  }
   LACS  = $4;                           { Listener active                }
   DTAS  = $2;                           { Device trigger state           }
   DCAS  = $1;                           { Device clear state             }

   { Error messages returned in global variable iberr}

   EDVR  = 0;                            { DOS error
   ECIC  = 1;                            { Function requires GPIB-PC to be CIC
   ENOL  = 2;                            { Write Function detected no Listener
   EADR  = 3;                            { Board not addressed correctly
   EARG  = 4;                            { Invalid argument to function call
   ESAC  = 5;                            { Function requires GPIB-PC to be SAC
   EABO  = 6;                            { I/O operation aborted
   ENEB  = 7;                            { Non-existent interface board
   EOIP  = 10;                           { I/O operation started before previo
                  { operation completed                 }
   ECAP  = 11;                           { No capability for operation
   EFSO  = 12;                           {}
   EBUS  = 14;                           { Command error during device call
   ESTB  = 15;                           { Serial Poll status byte lost
```

```
   ESRQ  = 16;                          { SRQ remains asserted

   { EOS mode bits                                        }

   BIN  = $1000;                        { Eight bit compare                }
   XEOS = $800;                         { Send EOI with EOS byte           }
   REOS = $400;                         { Terminate read on EOS            }

   { Timeout values and meanings                          }

   TNONE    = 0;                        { Infinite timeout (disabled)      }
   T10us    = 1;                        { Timeout of 10 us (ideal)         }
   T30us    = 2;                        { Timeout of 30 us (ideal)         }
   T100us   = 3;                        { Timeout of 100 us (ideal)        }
   T300us   = 4;                        { Timeout of 300 us (ideal)        }
   T1ms     = 5;                        { Timeout of 1 ms (ideal)          }
   T3ms     = 6;                        { Timeout of 3 ms (ideal)          }
   T10ms    = 7;                        { Timeout of 10 ms (ideal)         }
   T30ms    = 8;                        { Timeout of 30 ms (ideal)         }
   T100ms   = 9;                        { Timeout of 100 ms (ideal)        }
   T300ms   = 10;                       { Timeout of 300 ms (ideal)        }
   T1s      = 11;                       { Timeout of 1 s (ideal)           }
   T3s      = 12;                       { Timeout of 3 s (ideal)           }
   T10s     = 13;                       { Timeout of 10 s (ideal)          }
   T30s     = 14;                       { Timeout of 30 s (ideal)          }
   T100s    = 15;                       { Timeout of 100 s (ideal)         }
   T300s    = 16;                       { Timeout of 300 s (ideal)         }
   T1000s   = 17;                       { Timeout of 1000 s (maximum)      }

   { Miscellaneous                                        }

   S    =  $08;                         { Parallel Poll sense bit          }
   LF   =  $0A;                         { ASCII line feed character        }

{ NOT OPTIONAL: put in appropriate size for your buffer }

   MAXIBBUF = $100;                     { maximum buffer size for I/O }


Type

   iobuf =  array[1..MAXIBBUF] of char;
   IOstring = String [ 255 ];
         ibstring =  string[50];
   str4 = string[4];

 Var

    ibsta : integer;                    { status word                      }
    iberr : integer;                    { GPIB error code                  }
    ibcnt : integer;                    { number of bytes sent or, in the }
                                        { event of DOS error, the DOS error }
                                                                { code }
    ibbuf : iobuf;                      { I/O buffer for commands/data     }
    bdname : ibstring;                  { board or device name             }
```

```
    bd,dvm : integer;                    { Board descriptor                }
    vcnt: integer;                       { v or byte count                 }

{ OPTIONAL:                                                          }

    flname : ibstring;                   { file name                       }
    mask : integer;                      { wait mask for IBWAIT ftn.       }
    ppr,spr:integer;                     { parallel,serial poll responses }


{ Ibfn is the common entry point into the language interface, tpib.com.   }
{ Its arguments are generalized to meet the needs of each individual GPIP }
{ function, and are decoded as follows:                                   }
{          ibfnasm (name,iberr,ibcnt,buf,vcnt,bd,fcode,iberr,ibcnt)       }
{ where:        name  = string for bdname, flname, bname                  }
{               iberr = GPIB-PC error code                                }
{               ibcnt = GPIB-PC count                                     }
{               buf   = integer array for (var) rd, wrt, cmd buffers      }
{               vcnt  = integer for v, cnt, (var) spr, (var) ppr          }
{               bd    = integer for bd                                    }
{               fcode = integer for function code                         }

function ibfn (name:ibstring;var iberr,ibcnt:integer;var buf:iobuf;
               var vcnt:integer;bd,fncode:integer):integer; external 'tpib.co

{ You MUST include the appropriate declaration, as                }
{ given below, for each procedure or function you call.           }
{ You may omit declarations for functions you do not call.        }

procedure    ibbna (bd:integer;bname:ibstring);
        begin
        ibsta := ibfn(bname,iberr,ibcnt,ibbuf,vcnt,bd,26);
        end;
procedure    ibcac (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,16);
        end;
procedure    ibclr (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,22);
        end;
procedure    ibcmd (bd:integer;ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,32);
        end;
procedure    ibcmda (bd:integer;ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,33);
        end;
procedure    ibdiag (bd:integer;ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,34);
        end;
procedure    ibdma (bd:integer;v:integer);
        begin
```

```
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,11);
        end;
procedure    ibeos (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,12);
        end;
procedure    ibeot (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,14);
        end;
function     ibfind (bdname:ibstring):integer;
        var
        name : ibstring;
        begin
        name := bdname + chr(0);
        ibfind := ibfn(name,iberr,ibcnt,ibbuf,vcnt,bd,27);
        end;
procedure    ibgts (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,15);
        end;
procedure    ibist (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,10);
        end;
procedure    ibloc (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,5);
        end;
procedure    ibonl (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,1);
        end;
procedure    ibpad (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,8);
        end;
procedure    ibpct (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,24);
        end;
procedure    ibppc (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,7);
        end;
procedure    ibrd (bd:integer;var ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,28);
        end;
procedure    ibrda (bd:integer;var ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,29);
        end;
procedure    ibrdf (bd:integer;flname:ibstring);
        var
```

```
        name : ibstring;
        begin
        name := flname + chr(0);
        ibsta := ibfn(name,iberr,ibcnt,ibbuf,vcnt,bd,17);
        end;
procedure    ibrpp (bd:integer;var ppr:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,ppr,bd,19);
        end;
procedure    ibrsc (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,2);
        end;
procedure    ibrsp (bd:integer;var spr:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,spr,bd,25);
        end;
procedure    ibrsv (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,6);
        end;
procedure    ibsad (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,9);
        end;
procedure    ibsic (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,3);
        end;
procedure    ibsre (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,4);
        end;
procedure    ibstop (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,21);
        end;
procedure    ibtmo (bd:integer;v:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,v,bd,13);
        end;
procedure    ibtrg (bd:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,vcnt,bd,23);
        end;
procedure    ibwait (bd:integer;mask:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,mask,bd,0);
        end;
procedure    ibwrt (bd:integer;ibbuf:iobuf;cnt:integer);
        begin
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,30);
        end;
procedure    ibwrta (bd:integer;ibbuf:iobuf;cnt:integer);
        begin
```

```
        ibsta := ibfn(bdname,iberr,ibcnt,ibbuf,cnt,bd,31);
        end;
procedure    ibwrtf (bd:integer;flname:ibstring);
        var
        name : ibstring;
        begin
        name := flname + chr(0);
        ibsta := ibfn(name,iberr,ibcnt,ibbuf,vcnt,bd,18);
        end;

{=========================================================================}


Procedure Load ( OutString: IOstring; Var OutBuffer: IObuf );

{ This function outputs the information contained in OutString in a character
  array buffer format, that can be sent over the GPIB line. }

Var

   Index: Integer;
   TBuffer: IObuf;

Begin
For Index:= 1 to Length ( OutString ) do
   OutBuffer [ Index ]:= Char ( Copy ( OutString, Index, 1 ) )
End;                                        { Load }


Function StatusErr: Boolean;

{ This function returns a boolean value indicating the status of the GPIB, and
  the devices connected to it. }

Begin
StatusErr:= Boolean ( IBsta and ERR )
End;                                        { StatusErr }


{=========================================================================

Procedure ReadyQbt;

{ Sets up the initialization parameters for the Qubit. }


Const

   DeviceName = 'QUBIT';
   NumParam = 9;

Type

   Parameter = String [ 30 ];
   ParaList = Array [ 1 .. NumParam ] of Parameter;
```

```
Const

   ParaTable: ParaList = (
   'C011',                          { Set minimum display brightness }
   'C00400001',                     { Set line number }
   'C00500001',                     { Set event number }
   { No initialization is required for the LC 408 ( module 8 ). }
   'C01809104010203 04',
   { Set module 9, ON, Miniranger ( 4 code ), stations 1,2,3,4. }
   { No initialization is required for the Argo DM-54 ( module 10 ). }
   'C01200',                        { Set trigger length to 50 microseconds }
   'C01500',                        { Set fixing mode on time (FMT) }
   'C0066000.0',                    { Set inter-event time to 60 minutes }
   'C007',                          { Set start of line }
   'C000' );                        { System initialization completed }

(*****************************************************************************

   Function Time: IOstring;

   { Loads the Qubit Set Time command, and the DOS time into a string. }

   type
     regpack = record
                 ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
               end;

   var
     recpack:          regpack;                {assign record}
     ah,al,ch,cl,dh:   byte;
     hour,min,sec:     string[2];

   begin
     ah := $2c;                                {initialize correct registers}
     with recpack do
     begin
       ax := ah shl 8 + al;
     end;
     intr($21,recpack);                        {call interrupt}
     with recpack do
     begin
       str(cx shr 8,hour);                     {convert to string}
       str(cx mod 256,min);                          { " }
       str(dx shr 8,sec);                            { " }
     end;
   While Length ( Hour ) < 2 do
         Hour:= '0' + Hour;
   While Length ( Min ) < 2 do
         Min:= '0' + Min;
   While Length ( Sec ) < 2 do
         Sec:= '0' + Sec;
   Time := 'C002' + Hour + Min + Sec
   End;
```

```
Function Date: IOstring;

{ Loads the Qubit Set Date command, and the DOS date into a string. }

Type

  RegPack = Record
              AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags: Integer;
            End;

Var

  RecPack:        RegPack;            { Record for MsDos call }
  Month,Day:      String[2];
  Year:           String[4];
  DX,CX:          Integer;

Begin
With Recpack do
   AX := $2A shl 8;
MsDos(recpack);                       { Call function }
With RecPack do
   Begin
   Str(CX,Year);                      { Convert to string }
   Str(DX mod 256,Day);                       { " }
   Str(DX shr 8,Month)                        { " }
   End;
Year:= Copy ( Year, 3, 2 );
If Length ( Month ) < 2
   then
      Month:= '0' + Month;
If Length ( Day ) < 2
   then
      Day:= '0' + Day;
Date := 'C023' + Year + ':' + Month + ':' + Day
End;                                  { Date }


Procedure ErrorMsg;

{ This procedure displays an error message, and pauses to allow the operato
  to remedy the difficulty. }

Var

   Response: Char;

Begin
ClrScr;
Write ( 'Error: Unable to detect device.' );
GotoXY ( 1, 3 );
WriteLn ( 'For this program to work, the following must be true:' );
GotoXY ( 1, 5 );
WriteLn ( '1) The system must be booted with the diskette containing this '
```

```
          'program.' );
WriteLn ( '2) The QUBIT interface device must be on, and attached to the'
          ' computer via a' );
WriteLn ( '    GPIB cable.' );
GotoXY ( 1, 9 );
WriteLn ( 'Press the space bar when you feel you have the system configur
          ' correctly.' );
Repeat
   Read ( Kbd, Response );
   until Response = ' ';
End;                                    { ErrorMsg }


Procedure CheckReady;

{ This procedure attempts to open the device, and returns an error messag
  it is unable to do so. }

Var

   Ready: Boolean;

Begin
Ready:= False;
While not Ready do
   Begin
   Qbt:= IBfind ( DeviceName );
   If ( Qbt < 0 )
      then
         ErrorMsg
      else
         Ready:= True
   End;                                 { While not Ready }
End;                                    { CheckReady }


Procedure QBTinit;

{ This procedure clears the QUBIT, and sets the device to remote control.

Var

   OutBuffer: IObuf;

Begin
Repeat
   IBclr ( Qbt );
   If StatusErr
      then
         ErrorMsg;
   Load ( 'C016', OutBuffer );
   IBwrt ( Qbt, OutBuffer, 4 );
   If StatusErr
      then
         ErrorMsg;
```

```
   Load ( 'C013', OutBuffer );
   IBwrt ( Qbt, OutBuffer, 4 );
   If StatusErr
      then
         ErrorMsg;
   until not StatusErr
End;                                    { QBTinit }


Procedure QBTset;

{ This procedure sets the parameters that control the operation of the
  QUBIT. }

Var

   Index: Integer;
   OutBuffer: IObuf;

Begin
Load ( 'C0221', OutBuffer );
IBwrt ( Qbt, OutBuffer, 5 );        { Set calendar day mode }
Load ( Date, OutBuffer );
Ibwrt ( Qbt, OutBuffer, Length (Date) );
Load ( Time, OutBuffer );
IBwrt ( Qbt, OutBuffer, 10 );

For Index:= 1 to NumParam do
   Begin
   Load ( ParaTable [ Index ], OutBuffer );
   IBwrt ( Qbt, OutBuffer, Length ( ParaTable [ Index ] ) );
   If StatusErr
      then
         ErrorMsg
   End                                  { For Index }
End;                                    { QBTset }


Begin                                   { ReadyQbt }
CheckReady;
QBTinit;
QBTset
End;                                    { ReadyQbt }
```

File ReadyFile.Prc

```
Procedure ReadyFile (Var NoFile : Boolean);
Var
   Valid : Boolean;
   i     : Integer;

Begin
  WriteLn;
  NoFile:= False;
  Repeat
    ClrEOL;
    Filename:= '';
    Write ( 'What is the name of the data file? ' );
    ReadLn ( FileName );
    If FileName = '' then
      NoFile:= True
    Else Begin
      {$I-}
      Assign ( DataFile, FileName );
      Rewrite ( DataFile );
      {$I+}
      Valid:= (IOResult = 0);
      For i:= 1 to 4 do
        If Valid then Begin
          {$I-}
          If Weight [i] > 0 then
            Write (DataFile,RefStn [i].latitude:14:8, RefStn [i].longitude:1
          Else
            Write (DataFile, '              ', '              ');
          {$I+}
          Valid:= (IOResult = 0);
      End; {For i}

      If Valid then Begin
        {$I-}
        Writeln (DataFile);
        {$I+}
        Valid:= (IOResult = 0);
      End; {If}

      If Not Valid then
        WriteLn ( 'Invalid filename or error opening file.' );

    End; {If}
  until Valid;
End; { Procedure ReadyFile }
```

```
(*****************************************************************************

Procedure Decode ( Var InL: Line; Var X: Real; Max, Row, Col: Integer );

   { Decode the latitude or longitude data entered as a string in L.  A blank
     or a decimal may seperate degrees, minutes or seconds.  The result is
     returned in X.  If the data is invalid, line Row, from column Col onward
     is deleted and the position is requested again through a recursive call
     to this routine.

     Examples:  45              =  45 degrees
                45.5            =  45 deg, 30 min, 0 sec
                45 30           =  45 deg, 30 min, 0 sec
                45 0.5          =  45 deg, 0 min, 30 sec
                45 0 30         =  45 deg, 0 min, 30 sec

     Written by : Everett Coldwell   }

   Var
     N                  : Real;
     P, I, Len, Code    : Integer;
     Negative, Error    : Boolean;
     St                 : String [40] ;
     L                  : Line;

   Begin
     L:= InL;
     Len:= Length (L);
     P:= 1;
     Error:= False;
     X:= 0;

     { ---- Skip Blanks ---- }
     If (P <= Len) then
       While (P <= Len) and (L [P] = ' ') do
         P:= P + 1;

     { ---- Error if line empty or all blanks ----- }
     If (P = 0) or (P > Len) then
       Error:= True
     Else Begin

       { ---- Process Sign ---- }
       Negative:= False;
       If L [P] = '-' then
         Negative:= True;
       If L [P] in ['+','-'] then
         P:= P + 1;

       { ---- Error if line empty except for blank ---- }
       If P > Len then
         Error:= True
       Else
```

```
      { ---- Process Degrees, Minutes and Seconds ---- }
      For i:= 1 to 3 do
       If not (error or (p > Len)) then Begin

        { ---- Process Value ---- }
        Len:= Len-P+1;
        L:= Copy(L,P,Len);
        P:= Pos(' ',L);
        If P = 0 then
          P:= Len + 1;
        St:= Copy(L,1,P-1);
        Val (St,N,Code);
        If Code>0 then
          Error:= True
        Else Begin

          Case i of
            1: X:= N;
            2: X:= X + N/60;
            3: X:= X + N/3600;
          End; {Case}

          { ---- Skip Blanks ---- }
          If (P <= Len) then
            While (P <= Len) and (L [P] = ' ') do
              P:= P + 1;

          If ((i=3) or (Pos('.',St) > 0)) and (P <= Len) Then
            Error:= True;

        End; {If Code>0}
      End; {For i:= 1 to 3}
    End; {If (P=0) or (P>Len)}

    If Negative then
      X:= -X;

    If ( X > Max ) or ( X < -Max ) or Error then Begin
      GotoXY ( Col, Row );
      ClrEol;
      InL:= '';
      Read ( InL );
      Decode ( InL, X, Max, Row, Col )        { Recursive call }
    End { X >= Max }

End; { Procedure Decode }

{*****************************************************************}

Procedure Initialize ( Var RefStn: StnCoords; Var Weight: StnValues;
                       Var CLat, CLong: Real );

{ Purpose: To obtain the coordinates of the reference stations, and determin
           the ship's initial position.
```

```
  Written by: RossAllan McKenna and Everett Coldwell
  Date: February 18, 1985.
  Language: TurboPascal 3.0 with 8087 support                    }


Var

   Response, Choice : Char;
   Lat, Long : Array [1..4] of Line;
   LatStr, LongStr : Line;
   Error : Boolean;
   Ref : Integer;

{*****************************************************************}

   Procedure GetRef ( Var RefStn: StnCoords; Var Weight: StnValues );

   { Obtain the reference station coordinates from the user. }

   Var

      L: Line;
      Ref : Integer;
      Max, Row, Col: Integer;
      Response: Char;

   Begin
     ClrScr; GotoXY ( 30, 1 );
     Write ( 'MiniRanger Initialization' );
     GotoXY ( 1, 3 );
     Writeln (' Enter latitude and longitude positions by seperating the');
     Writeln (' numbers by blanks and a decimal. eg: 45 30 10 | 45 30.5 | 45.5

     GotoXY ( 1, 6 );
     For Ref:= 1 to 4 do Begin
       With RefStn [Ref] do Begin
         Write ( 'Is station ', Ref, ' active ? (Y/N) ' );
         Response:= ' ';
         While Not (Response in ['Y','N','y','n'] ) do
           If KeyPressed then
             Read (kbd,Response);
         Writeln;
         If ( Pos ( Response, 'Yy' ) <> 0 ) then begin
           StationOK [Ref]:= True;
           Write ( 'Please enter its latitude' );
           If Lat [Ref] <> '' then
             Write (' (',Lat [Ref],')');
           Write (': ');
           Max:= 90; Row:= WhereY; Col:= WhereX;
           Read ( L );
           If L = '' then
             L:= Lat[ref];
           Decode ( L, Latitude, Max, Row, Col );
           Lat [Ref]:= L;
```

```
          GotoXY ( 17, Row + 1 );

          Write ( 'longitude' );
          If Long [Ref] <> '' then
            Write (' (',Long [Ref],')');
          Write (': ');
          Max:= 180; Row:= WhereY; Col:= WhereX;
          L:= '';
          Read ( L );
          If L = '' Then
            L:= Long [Ref];
          Decode ( L , Longitude, Max, Row, Col );
          Long [ref]:= L;
          GotoXY ( 1, Row+1 );
          Weight [Ref] := 1.0
        End
        else Begin
          StationOK [Ref] := False;
          Weight [Ref] := 0.0;
          Latitude:= 0.   ;  Longitude:= 0.;
        End; {else}
      End { With RefStn }
    End; { For Ref:= 1 to  4 }
  End; { Procedure GetRef }

(*****************************************************************************

  Procedure LoadCoordinates ( Var RefStn: StnCoords;
                              Var Weight: StnValues;
                              Var Error: Boolean );

  { This routine loads the coordinates of the reference stations }

  Var
    Valid : Boolean;
    DataFile : Text;
    Ref      : Integer;

  Begin
    GotoXY ( 1, 24 ); ClrEol;
    Write ( 'Loading Coordinates' );

    {$I-}
    Assign ( DataFile, CoordinateFileName );
    Reset ( DataFile );
    {$I+}
    Error:= (IOResult <> 0);

    If Not Error then
      For Ref:= 1 to 4 do
       If Not Error then
        With RefStn [Ref] do Begin
          {$I-}
          ReadLn (DataFile, Latitude, Longitude, Weight [Ref] );
          ReadLn (DataFile, Lat [Ref] );
```

```
        ReadLn (DataFile, Long [Ref] );
        If Weight [Ref] > 0 then
          StationOk [Ref]:= True;
        {$I+}
        Error:= ( IOresult <> 0 );
      End; { For Ref }
    Close ( DataFile );

    If Error then Begin
      GotoXY (1,25); ClrEol;
      Write ('Coordinate data file not valid');
      Delay (2000);
      ClrScr;
      Exit;
    End; { If }

    ClrScr; GotoXY ( 1, 3 );
    For Ref:= 1 to 4 do
      If Weight [Ref] <> 0 then
        With RefStn [Ref] do Begin
          GotoXY (1, WhereY + 2);
          Write ('Reference Station ',Ref,':');
          GotoXY (25, WhereY);
          Write ('Latitude: ',Lat [Ref] );
          GotoXY (24, WhereY+1);
          Write ('Longitude: ',Long [Ref] );
        End  { With RefStn }
  End; { Procedure LoadCoordinates }

(*****************************************************************************

  Procedure SaveCoordinates ( Var RefStn: StnCoords;
                              Var Weight: StnValues;
                              Var Error: Boolean );

  { This routine saves the coordinates of the reference stations }

  Var
    Valid : Boolean;
    DataFile : Text;
    Ref      : Integer;

  Begin
    GotoXY ( 1,24 ); ClrEol;
    Write ( 'Saving Coordinates' );

    Assign ( DataFile, CoordinateFilename );
    {$I-}
    Rewrite (DataFile);
    {$I+}
    Error:= (IOResult <> 0);

    If Not Error then Begin
      For Ref:= 1 to 4 do
       If Not Error then
```

```
        With RefStn [Ref] do Begin
          {$I-}
          WriteLn (DataFile, Latitude, Longitude, Weight [Ref] );
          WriteLn (DataFile, Lat [Ref] );
          WriteLn (DataFile, Long [Ref] );
          {$I+}
          Error:= ( IOresult <> 0 );
      End; { For Ref }
      Close ( DataFile );
    End; { If Not Error }

    If Error then Begin
      GotoXY (1,25); ClrEol;
      Write ('Error writing to Coordinate data file');
      Delay (2000);
      ClrScr;
    End; { If }
  End; { Procedure SaveCoordinates }

(*****************************************************************************

  Procedure GetPos ( Var CLat, CLong: Real );

  { Obtain the current position coordinates from the user. }

  Var

     L : Line;
     Max, Row, Col: Integer;

  Begin
    GotoXY ( 1, 20 );
    Write ( 'Enter your current latitude' );
    If LatStr <> '' then
      Write (' (',LatStr,')');
    Write (': ');
    Max:= 90; Row:= 20; Col:= WhereX;
    Read ( L );
    If L = '' then
      L:= LatStr;
    Decode ( L, CLat, Max, Row, Col );
    LatStr:= L;

    GotoXY ( 20, 21 );
    Write ( 'longitude' );
    If LongStr <> '' then
      Write (' (',LongStr,')');
    Write (': ');
    Max:= 180; Row:= 21; Col:= WhereX;
    Read ( L );
    If L = '' then
      L:= LongStr;
    Decode ( L, CLong, Max, Row, Col );
    LongStr:= L;
  End;                                  { GetPos }
```

```
(*****************************************************************************

Begin                                          { Initialize }
  LatStr:= ''; LongStr:= '';
  For Ref:= 1 to 4 do Begin
    Lat [Ref]:= ''; Long [Ref]:= '';
  End; {For}

  Repeat
    ClrScr;
    GotoXY (23,11);
    Write ('L - Load coordinates from disc');
    GotoXY (23,12);
    Write ('E - Enter coordinates with keyboard');
    GotoXY (23,14);
    Write ('Please type L or E: ');
    Choice:= ' ';
    While Not (Choice in ['L','E','l','e'] ) do
      If KeyPressed then
        Read (kbd,Choice);
    ClrScr;

    Error:= False;

    If Choice in ['L','l'] then
      LoadCoordinates ( RefStn, Weight, Error )
    Else
      GetRef ( RefStn, Weight );

    If Not Error then Begin
      GetPos ( CLat, CLong );

      GotoXY (1,24); ClrEol;
      Write ( 'Are the coordinates correct? (Y/N)' );
      Response:= ' ';
      While Not (Response in ['Y','N','y','n'] ) do
        If KeyPressed then
          Read (kbd,Response);
    End; { If not error }
  until Response in ['Y','y'] ;

  If Choice in ['E','e'] then Begin
    GotoXy (1,WhereY); ClrEol;
    Write ('Save coordinates to disc? (Y/N)');
    Response:= ' ';
    While Not (Response in ['Y','N','y','n'] ) do
      If KeyPressed then
        Read (kbd,Response);
    If Response in ['Y','y'] then
      SaveCoordinates ( RefStn, Weight, Error );
  End; { If Choice = 'E' }
End;                                           { Initialize }
```

File Chart.Prc

```
{ Procedure Library:  Chart.Prc
  Author:             Everett Coldwell
  Date:               Feb 7,1986

    These procedures enable the calling program to draw a chart of position
  on the COMPAQ screen.

  Needed Declations:

      Type
        Line = String [80];

      Var
        Lastx, Lasty          : Integer;
        CLat, CLong           : Real;
        EastLong, WestLong    : Real;
        NorthLat, SouthLat    : Real;
        NumXlines, NumYlines  : Real;
        XScale, YScale        : Real;
        EastLongStr, WestLongStr    : Line;
        NorthLatStr, SouthLatStr    : Line;
        LimitsSet             : Boolean;

   Procedure Decode ( Var InL: Line; Var X: Real; Max, Row, Col: Integer );}

{*****************************************************************************}

Procedure InputChartLimits;

Var
  i, Max     : Integer;
  Row, Col   : Integer;
  X          : Real;
  L          : Line;
  key        : Char;

Begin
     ClrScr; GotoXY ( 20, 1 );
     Write ( 'MiniRanger Chart Initialization' );

     If LimitsSet Then Begin
       GotoXY ( 1, 3 );
       Write (' Reenter Plot Limits? (Y/N)  ');
       key:= ' ';
       While not (key in ['Y','y','N','n']) do
         If Keypressed then
           Read (kbd,key);
       If key in ['Y','y'] then Begin
         LimitsSet:= False;
         GotoXY (1,WhereY);
         ClrEol;
       End; {If}
     End; {If}
```

```
If Not LimitsSet then Begin
  GotoXY ( 1, 3 );
  Writeln (' Enter latitude and longitude positions by seperating the',
           ' numbers by');
  Writeln (' either a blank or a decimal.  eg: 45 30 10 | 45 30.5 | 45.5'

  GotoXY ( 1, 6 );
  i:= 1;
  While i<7 do Begin
    Case i of
      1: Begin
           GotoXY ( 30, 10);
           Write ('South Latitude');
           Max:= 90;
         End;
      2: Begin
           GotoXY ( 30, 6);
           Write ('North Latitude');
           Max:= 90;
         End;
      3: Begin
           GotoXY ( 10, 8);
           Write ('West Longitude');
           Max:= 180;
         End;
      4: Begin
           GotoXY ( 50, 8);
           Write ('East Longitude');
           Max:= 180;
         End;
      5: Begin
           GotoXY ( 10, 12);
           Write ('Number of X axis Lines');
           Max:= 30;
         End;
      6: Begin
           GotoXY ( 10, 13);
           Write ('Number of Y axis Lines');
           Max:= 15;
         End;

    End; {Case}

    Write (':  ');  ClrEol;
    Read ( L );
    Row:= WhereY; Col:= WhereX;
    Decode ( L, X, Max, Row, Col );

    Case i of
      1: Begin
           SouthLatStr:= L;
           SouthLat:= x;
         End;
      2: Begin
```

```
                NorthLatStr:= L;
                NorthLat:= x;
                If SouthLat >= NorthLat then Begin
                  i:=0;
                  GotoXY (20, 15);
                  Write ('North must be more than South Latitude');
                  Delay (3000);
                  GotoXY (20, 15);
                  ClrEol;
                End; {If}
              End; {If}
            3: Begin
                WestLongStr:= L;
                WestLong:= x;
              End;
            4: Begin
                EastLongStr:= L;
                EastLong:= x;
                If EastLong <= WestLong then Begin
                  i:=2;
                  GotoXY (20, 15);
                  Write ('East must be more than West Longitude');
                  GotoXY (20, 16);
                  Write ('(Longitudes are negative in this part of the world
                  Delay (4000);
                  GotoXY (20, 15);
                  ClrEol;
                  GotoXY (20, 16);
                  ClrEol;
                End; {If}
              End; {If}
            5: NumXLines:= x;
            6: NumYLines:= x;
          End; {Case}

        i:= i + 1;

      End; { While }

      LimitsSet:= True;
    End; {If Not LimitsSet}

    LimitsSet:= True;

End; { Procedure InputChartLimits }

{*****************************************************************************

Procedure InitializeChart;
Var
    x, y          : Real;
    xstep, ystep  : Real;

Begin
  { ***** Compute Scale factors ***** }
```

```
  xscale:= 639 / (EastLong - WestLong);
  yscale:= 170 / (NorthLat - SouthLat);
  xstep:= 639 / NumXlines;
  ystep:= 170 / NumYlines;

  { ***** Initialize Plot ***** }

  ClrScr;
  HiRes;
  GraphWindow (0,10, 639,180);

  { ***** Draw lines of Latitude ***** }

  x:= 639;
  While x >= 0 do Begin
    Draw (Round(x),0, Round(x),170, white);
    x:= x - xstep;
  End; {While}
  Draw (0,0, 0,170, white);

  { ***** Draw lines of Longitude ***** }

  y:= 0;
  While y <= 170 do Begin
    Draw (0,Round(y), 639,Round(y), white);
    y:= y + ystep;
  End; {While}

  { ***** Initialize Plot Variables ***** }

  LastX:= -1;
  LastY:= -1;

  GOTOXY (1,1);
  WRITE ('Latitude: ',SouthLatStr,' to ',NorthLatStr,
         '     Longitude: ',WestLongStr,' to ',EastLongStr);

End; { Procedure InitializeChart }

{*****************************************************************************

Procedure PlotPosition (CLat, CLong : Real);
Var
    x, y : Integer;

Begin
  If (CLat >= SouthLat) and (CLat <= NorthLat) and
     (CLong >= WestLong) and (CLong <= EastLong) then Begin

    x:= round ( xscale * (CLong - WestLong) );
    y:= 170 - round ( yscale * (CLat - SouthLat) );

    If (LastX >= 0) and (LastY >= 0) then
      Draw ( LastX, LastY, x, y, white )
```

```
    Else
      Plot ( x, y, white );
    Lastx:= X;
    Lasty:= Y;
  End
  Else Begin
    Lastx:= -1;
    Lasty:= -1;
  End;

End; { Procedure PlotPosition }

{*****************************************************************************

{ Example Calling Routine:
Begin
  InputChartLimits;
  InitializeChart;
  Clat:= (NorthLat - SouthLat) * 0.31 + SouthLat;
  CLong:= (EastLong - WestLong) * 0.45 + WestLong;
  PlotPosition (CLat,CLong);
  Clat:= (NorthLat - SouthLat) * 0.39 + SouthLat;
  CLong:= (EastLong - WestLong) * 0.46 + WestLong;
  PlotPosition (CLat,CLong);
  Clat:= (NorthLat - SouthLat) * 0.91 + SouthLat;
  CLong:= (EastLong - WestLong) * 0.95 + WestLong;
  PlotPosition (CLat,CLong);
End.  }
```

File Sphrd.Prc

```
Procedure Sphrd ( LatPt1, LongPt1, LatPt2, LongPt2 : REAL;
                  VAR Dist, DDP, DDL               : REAL );

{     Purpose :  Sphrd calculates the distance between two lat/long
                 positions along with two distance derivatives.  It can
                 also calculate the forward and reverse azimuths, although
                 currently commented out.

      Written By :  D. W. Hackett
      Date       :  June, 1985

      Background :  Copied and transfered to turbo from the fortran program
                    SPHRD written by: P. Delorme, using T. Vincenty's
                    algorithim.

VAR     LAT1, LAT2, LONG1, LONG2  : REAL;        { The two positions
        FL, FUZ, DARG, BE         : REAL;
        TU1, TU2, U1, U2          : Real;
        SU1, SU2, CU1, CU2        : REAL;
        DL, XDL, DL1, SDL, CDL    : Real;
        CS, SS, SIG, SA, CA, C2SM : REAL;
        U, A, B, C, DSIG          : Real;
        SDL1, CDL1                : REAL;
        DLON, SL, CL, SP, CP, SD  : REAL;

{=================================================================}

FUNCTION DEPS (DARG : REAL) : REAL;

{ Returns the smallest number such that 1 + Deps > 1. }

BEGIN
Deps:= 2.0e-16
END;

{=================================================================}

FUNCTION TAN (Angle : REAL) : REAL;

{     Purpose:  Computes the Tan of the inputted angle - in radians     }

BEGIN
   TAN := SIN(Angle) / COS(Angle);
END;

{=================================================================}

BEGIN
   FL := 1.0 / F;
   FUZ := 20.00 * DEPS(DARG);
   BE := AE * ( 1.0 - FL );

   {=====     Make the Lat/Long #1 the largest of the two     =====}
```

```
{=====     and convert to radian degrees                    =====}

IF ( LONGPT1 <= LONGPT2 ) THEN
   BEGIN
   LAT1 := ( LATPT1 * PI ) / 180.0;
   LAT2 := ( LATPT2 * PI ) / 180.0;
   LONG1 := ( LONGPT1 * PI ) / 180.0;
   LONG2 := ( LONGPT2 * PI ) / 180.0;
   END
ELSE
   BEGIN
   LAT1 := ( LATPT2 * PI ) / 180.0;
   LAT2 := ( LATPT1 * PI ) / 180.0;
   LONG1 := ( LONGPT2 * PI ) / 180.0;
   LONG2 := ( LONGPT1 * PI ) / 180.0;
   END;

{=====     Calculate the reduced latitude and its trig functions     ====

TU1 := ( 1.0 - FL ) * TAN(LAT1);
TU2 := ( 1.0 - FL ) * TAN(LAT2);
U1 := ARCTAN(TU1);
U2 := ARCTAN(TU2);
SU1 := SIN(U1);
SU2 := SIN(U2);
CU1 := COS(U1);
CU2 := COS(U2);

{=====     1st Approx:  Difference in Longitude on ellipsoid is     =====
{=====     equal to the difference in Longitude on sphere           =====

DL := LONG2 - LONG1;
XDL := DL;
DL1 := DL;

REPEAT
   DL := DL1;
   SDL := SIN(DL);
   CDL := COS(DL);
   CS := (SU1 * SU2) + (CU1 * CU2 * CDL);
   SS := SQRT(1.0 - SQR(CS));
   SIG := ARCTAN(SS / CS);

   (****************************
   *****   new means of calculating SIG
   UMEAN  := ( u1 + u2 ) / 2.0;
   HALPHI := ( U2 - U1 ) / 2.0;
   HALLAM := ( DL / 2.0;
   NORTH  := COS (HALLAM) * SIN(HALPHI);
   EAST   := COS(UMEAN) * SIN(HALLAM);
   SINDH  := SQRT(NORTH * NORTH + EAST * EAST)
   COSDH  := SQRT(1.0 - SINDH * SINDH);
   SIG    := 2.0 * ARCTAN2 (SINDH,COSDH);
   ******************************)
```

```
   IF ( ABS(SS) < FUZ ) THEN
      SS := FUZ;
   SA := ( CU1 * CU2 * SDL ) / SS;
   IF SA > 1-SQRT(FUZ) then begin
     Writeln;
     WriteLn ( LatPt1:15:10, LongPt1:15:10, LatPt2:15:10, LongPt2:15:10 );
     Writeln ('SA=',sa,'  ss=',ss,'  sdl=',sdl);
     Writeln ('cs=',cs,' cu1=',cu1,' cu2=',cu2);
     Writeln ('dl=',dl,' su1=',su1,' su2=',su2);
     Writeln ('fl=',fl,'  u1=',u1,'  u2=',u2);
     Writeln ('tu1=',tu1,'tu2=',tu2);
   End; {If}
   CA := SQRT( 1.0 - SQR(SA));
   C2SM := CS - ( 2.0 * SU1 * SU2 ) / SQR(CA);

   {=====     Calculate the difference in longitude on the auxiliary sphere

   C := ( FL / 16.0 ) * SQR(CA) * ( 4.0 + ( FL * (4.0 - 3.0 * SQR(CA))));
   DL1 := XDL + (1.0-C) * FL*SA*(SIG+C*SS*(C2SM+C*CS*(-1.0+2.0*SQR(C2SM)))
UNTIL ABS(DL1 - DL) <= (10E-10);

U := SQR(CA) * (SQR(AE) - SQR(BE)) / SQR(BE);
A := ( 1.0 + (U / 256.0) * (64.0 + U * (-12.0 + 5.0 * U)));
B := ( U / 512.0 ) * (128.0 + U * (-64.0 + (37.0 * U)));
DSIG := B * SS * ( C2SM + 0.2500 * B * CS * (-1.0 + 2.0 * SQR(C2SM)));
DIST := BE * A * (SIG - DSIG);
SDL1 := SIN(DL1);
CDL1 := COS(DL1);

(*********************************************************
****   CALCULATE AZIMUTHS
X := CU1 * SU2 - SU1 * CU2 * CDL1;
Y := CU2 * SDL1;
IF (X < 0) AND (Y < 0) THEN
   AZ1 := (ARCTAN(Y/X) + (PI * -1)) * (180.0 / PI);
IF ( X = 0) AND (Y < 0) THEN
   AZ1 := ((PI * -1) / 2.0) * (180 / PI);
IF (X = 0) AND (Y > 0) THEN
   AZ1 := (PI / 2.0) * (180 / PI);
IF (X < 0) AND (Y > 0) THEN
   AZ1 := (ARCTAN(Y/X) + PI) * (180.0 / PI);
IF (X > 0) THEN
   AZ1 := ARCTAN(Y/X) * (180.0 / PI);
IF (X = 0) AND (Y = 0) THEN
   ERRORS := 1;
IF LONGPT1 > LONGPT2 THEN
   AZ1 := AZ1 + 180;
X := SU1 * CU2 - CU1 * SU2 * CDL1;
Y := -1.0 * CU1 * SDL1;
IF (X < 0) AND (Y < 0) THEN
   AZ2 := (ARCTAN(Y/X) + (PI * -1)) * (180.0 / PI);
IF ( X = 0) AND (Y < 0) THEN
   AZ2 := ((PI * -1) / 2.0) * (180 / PI);
IF (X = 0) AND (Y > 0) THEN
   AZ2 := (PI / 2.0) * (180 / PI);
```

```
   IF (X < 0) AND (Y > 0) THEN
      AZ2 := (ARCTAN(Y/X) + PI) * (180.0 / PI);
   IF (X > 0) THEN
      AZ2 := ARCTAN(Y/X) * (180.0 / PI);
   IF (X = 0) AND (Y = 0) THEN
      ERRORS := 1;
   IF AZ2 < 0 THEN
      AZ2 := AZ2 + 360;
   IF LONGPT1 >= LONGPT2 THEN
      AZ2 := AZ2 + 180;

*********************************************************)

   {=====     Calculate the derivatives     =====}

   DLON := ( LONGPT1 - LONGPT2 ) * PI / 180.00;
   SL := SIN(DLON);
   CL := COS(DLON);
   SP := SIN( LATPT1 * PI / 180.00);
   CP := COS( LATPT1 * PI / 180.00);
   SS := SIN( LATPT2 * PI / 180.00);
   CS := COS( LATPT2 * PI / 180.00);
   SD := SIN( DIST / AE );
   DDP := (( SP * CS * CL - CP * SS ) / SD ) * AE * PI / 180.00;
   DDL := ( CP * CS * SL / SD ) * AE * PI / 180.00;
END;
```

File NauticalMile.Prc

```
Function NauticalMiles ( lat1, long1, lat2, long2 : Real ) : Real;
{ This function returns the distance in nautical miles between two positions.
  Since the routine does not use spherical geometry, it is recommended that
  it be used only for short distances.
  The definition of a nautical mile, as used here, is 1 minute of arc.

  Written by: Everett Coldwell, Feb 17, 1986 }

Var
  Dlat, Dlong : Real;
  AvgLat      : Real;
  ToRads      : Real;

Begin
  DLat:= lat1 - lat2;
  DLong:= long1 - long2;
  AvgLat:= (lat1 + lat2) / 2.;
  ToRads:= 3.141592654 / 180.;

  NauticalMiles:= SQRT ( SQR(Dlat*60) + SQR(Dlong*60*COS(AvgLat*ToRads)) );

End;
```

File ReadRn.Prc

```
Procedure ReadRn ( Var Rng: StnValues );

{ Purpose: To read the current time mark and range data for the MiniRanger
           Positioning System from the Qubit interface.

  Written by: RossAllan McKenna
  Date: August 27th, 1985.
  Language: TurboPascal 3.0, with 8087 support.

  Last modified: Dec. 23rd by R. McKenna

                 Feb 14, 1986 by Everett Coldwell }

Type

   Str12 = String[12];
Var

   MinRng: IObuf;
   Error: Boolean;
   Station,
   Code: Integer;
   RngStr: Str12;

{*************************************}
Function Value (c:char) : Integer;
Begin
  Value:= Pos (c, '0123456789') - 1;
End; {Function Value}

{*************************************}


Procedure ReadMR ( Var MinRng: IObuf; Var Error: Boolean );
{ Modified Jan 31, 1986 by Everett Coldwell - Timestamp output format change


Var

   Found: Boolean;
   Rd: IObuf;
   Pos,
   Position,
   SearchPos,
   Startpos: Integer;


Begin

  IBwait ( Qbt, Mask );
  If ( IBsta and Err ) <> 0 then Begin
    WriteLn ( 'Wait error.' );
    Error:= True
```

```
End; { If then }
If ( IBsta and Timo ) <> 0 then Begin
  WriteLn ( 'Timeout.' );
  Error:= True
End; { If then }

IBrd ( Qbt, Rd, MAXIBBUF );          { Clear buffer }
If ( IBsta and Err ) <> 0 then Begin
  WriteLn ( 'Read error.    ' );
  Error:= True
End; { If then }

Found:= False;
While not Found do Begin

  Write (''); {So that ^C will stop program}

  IBwait ( Qbt, Mask );
  If ( IBsta and Err ) <> 0 then Begin
    WriteLn ( 'Wait error.' );
    Error:= True
  End; { If then }
  If ( IBsta and Timo ) <> 0 then Begin
    WriteLn ( 'Timeout.' );
    Error:= True
  End; { If then }

  IBrd ( Qbt, Rd, MAXIBBUF );
  If ( IBsta and Err ) <> 0 then Begin
    WriteLn ( 'Read error.    ' );
    Error:= True
  End; { If then }

  For SearchPos:= 1 to MAXIBBUF do
    If ( Rd [ SearchPos ] = Chr (10) ) and      { Linefeed }
       ( Rd [ SearchPos + 1 ] = '0' ) and       { Module number }
       ( Rd [ SearchPos + 2 ] = '9' )           { '0', '9' = MiniRanger }
       then Begin
         StartPos:= 1;
         While (Rd [StartPos] <> 'R') and (StartPos < SearchPos) do
           StartPos:= StartPos + 1;
         Found:= (StartPos < SearchPos);
         Position:= SearchPos
       End; { If then }
End; { While not Found }


Ctime:= value(Rd[Startpos + 8])*36000. + value(Rd[StartPos + 9])*3600.
      + value(Rd[Startpos + 11])*600. + value(Rd[StartPos + 12])*60.
      + value(Rd[Startpos + 14])*10. + value(Rd[StartPos + 15])
      + value(Rd[Startpos + 17])/10.;

GotoXY (1,WhereY);
Write (' ');
For Pos:= StartPos + 2 to StartPos + 15 do Begin
```

```
    If Pos <> Startpos + 7 then begin
      Write ( Rd[Pos] );                    { Write time stamp }
      OutRecord:= OutRecord + Rd[Pos];
    End {If then}
    Else begin
      Write (' ');
      OutRecord:= OutRecord + ' ';
    End; {If else}
  End; {For Pos}

  While WhereX <> 80 do
    Write (' ');

  GotoXY (16, WhereY);
  For Pos:= Position to Position + 46 do
    MinRng [ Pos - Position ]:= Rd [ Pos ];
End; { Procedure ReadMR }


{-----------------------------------------------------------------------------

Procedure Astring ( MinRng: IObuf; Station: Integer; Var RngStr: Str12;
                    Var Error: Boolean );

{ This procedure extracts the range data for the specified station from the
  input array. }
Var

   ChrPos: Integer;

Begin
RngStr:= '';
For ChrPos:= ( 4 + Station*11 - 10 ) to ( 4 + Station*11 - 2 ) do
   RngStr:= RngStr + Char ( MinRng [ ChrPos ] );
End;                                        { Astring }

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Begin
  ReadMR ( MinRng, Error );
  For Station:= 1 to 4 do Begin
    Astring ( MinRng, Station, RngStr, Error );
    OutRecord:= OutRecord + ' ' + RngStr;
    Val ( RngStr, Rng[Station], Code )
  End { For }
End; { Procedure ReadRn }
```

File RangeFix.Prc

```
Procedure RangeFix ( Var CLat, CLong: Real; RefStn: StnCoords;
                     Weight, Range: StnValues; Var Errors: Integer);

{   RangeFix inputs a Lat/Long along with 4 Range readings and the Range
    Stations Lat/Longs.  Depending on the fix type, it will move the inputed
    lat/long until the fix error is less then .00001.  The new lat/long
    is then outputted.

    Written By : D. W. Hackett
    Date       : July, 1985

    Background : Copied from RNGFIX written by Grant, July 1982 for the
                 BioNav system.

    Modified by: R. McKenna, Aug., Dec., 1985.

CONST  FixError = 1.0E-5;                    { Fix error limit                }

VAR    Finished                : Boolean;    { Flag to terminate loop         }
       Count, I                : Integer;    { Loop control varibles          }
       Lat,Long                : Real;       { Local latitude and longitude   }
       B11, B21, B22           : Real;       { Matrix components              }
       DDP, DDL, DDPDiff       : Real;       { Matrix derivatives             }
       D1, D2, DD              : Real;       { Distance derivatives           }
       Distance                : Real;       { Distance between two lat/longs }
       LatAdjust               : Real;       { Change from Approx to New Lat  }
       LongAdjust              : Real;       { Change from Approx to New Long }
       Residue                 : StnValues;  { Residue distance               }

{=================================================================================

BEGIN

  {===   Initialize   ===}

  Lat:= CLat;
  Long:= CLong;
  Finished        := FALSE;
  Errors          := 0;
  Count := 0;

  {===   Start iteration fix calculation   ===}

  REPEAT
     Count := Count + 1;

     {===   Initialize matrix and derivatives   ===}

     B11 := 0.0;
     B21 := 0.0;
     B22 := 0.0;
     D1  := 0.0;
     D2  := 0.0;
```

```
{===   Fill matrix with calculated distances    ===}
{===   Calculate accumulative derivatives       ===}

FOR I := 1 TO 4 DO
   BEGIN
   SPHRD ( Lat, Long, RefStn[I].Latitude, RefStn[I].Longitude, Distanc
           DDP, DDL);
   B11 := B11 + Weight[I] * DDP * DDP;
   B22 := B22 + Weight[I] * DDL * DDL;
   B21 := B21 + Weight[I] * DDP * DDL;

   {=====     Calculate the difference between the range      =====}
   {=====     readings and the computed distances between     =====}
   {=====     the approx lat/long and the stations            =====}

   Residue[I] := Distance - Range[I];
   D1 := D1 + DDP * Weight[I] * Residue[I];
   D2 := D2 + DDL * Weight[I] * Residue[I];
   END;   { For loop }


{=====     Calculate and apply the Lat/Long adjustment    =====}

DD := B11 * B22 - B21 * B21;
IF ( DD <> 0 ) THEN
   BEGIN
   LatAdjust  := ( B22 * D1 - B21 * D2 ) / DD;
   LongAdjust := ( B11 * D2 - B21 * D1 ) / DD;
   Lat  := Lat  - LatAdjust;
   Long := Long - LongAdjust;
   END   { If Statement }
ELSE
   Errors := 2;

{=====     If no fix after 21 tries - assume none will be found     ==

IF ( Count >= 21 ) THEN
   Errors := 1;

{=====     Check if the fix was good enough to quit     =====}

IF ( ABS(LatAdjust) <= FixError ) OR ( ABS(LongAdjust) <= FixError ) T
   Finished := TRUE;

UNTIL ( Finished ) OR ( Errors <> 0 );

{=====     Set the adjusted Lat/Long to be the new Lat/Long     =====}

IF ( Errors = 0 ) THEN
   BEGIN
   CLat  := Lat;
   CLong := Long;

   (*************************************************************
```

```
*************   Not in use or needed   **********************
*                                                            *
* RaDeg := PI / 180.00;                                      *
* PHI := LAT * RaDeg;                                        *
* ACON := SQR( AE * RADEG );                                 *
* A11 := ( B22 / DD ) * ACON;                                *
* A21 := -( B21 / DD ) * ACON * COS(PHI);                    *
* A22 := ( B11 / DD ) * ACON * SQR(COS(PHI));                *
* DCON := SQR(SQRT( A11 + A22 )) - (A11 * A22 - SQR(A21));   *
* Theta := 0.5 * ArcTan2 (2 * A21  A11 - A22) / RaDeg;       *
* SMJAX := 2.45 * SQRT ( 0.5 * (A11 + A22 + DCON));          *
* SMNAX := 2.45 * SQRT ( 0.5 * (A11 + A22 - DCON));          *
* RNORM := 0.0;                                              *
* IF NUM > 3 THEN                                            *
*    BEGIN                                                   *
*    FOR I := 1 NUM DO                                       *
*       RNORM := RNORM + RES[I] * RES[I] * WEIGHT[I];        *
*    RNORM := RNORM / ( NUM - 2 );                           *
*    END;                                                    *
*************************************************************)
END;   { If Statement }
END;   { End Procedure }
```

File Position.Prc

```
Procedure Position ( Var CLat, CLong: Real; RefStn: StnCoords;
                     Weight: StnValues; Var Errors: Integer );

{ Purpose: To obtain the new position, according to the current observed
           distances ( in meters ) to the stations.

  Written by: RossAllan McKenna
  Date: August 27th, 1985.
  Language: TurboPascal 3.0 with 8087 support.

  Last modified: Feb. 7 by E. Coldwell

Var
   Stat: Integer;

Begin
  ReadRn ( Range );
  RangeFix ( CLat, CLong, RefStn, Weight, Range, Errors )
End;                                        { Position }
```

File Course.Prc

```
Function Course ( Lat1, Long1, Lat2, Long2: Real ): Real;
{ This function returns the bearing between two geographic coordinates. }
{ Note that this function uses planer and not sperical geometry, therefore
  its accuracy decreases with distance between positions }

Var

   DLat, DLong,
   Angle: Real;

   Function Sgn ( X: Real ): Real;
   Begin
     IF X>=0 then
       Sgn:= 1
     Else
       Sgn:= -1;
   End; { Sgn }

Begin
If (abs(lat1) > 90) or (abs(long1)>180) then Begin
  course:= 0;
  Exit;
End;
DLat:= Lat2 - Lat1;
If ((Long2>170) and (Long1<-170)) or ((Long2<-170) and (Long1>170))
   then
      DLong:= Sgn(Long1) * ((180-Abs(Long2)) + (180-Abs(Long1)))
   else
      DLong:= Long2 - Long1;
If DLat = 0
   then
      If DLong = 0
         then
            Angle:= 0
         else
            Angle:= Sgn(DLong) * Pi/2
   else
      If DLong <> 0
         then
            Angle:= ArcTan(DLong/DLat)
         else
            Angle:= 0;

Angle:= 180 * Angle / Pi;
If DLat < 0
   then
      Angle:= Angle + 180;
If Angle < 0
   then
      Angle:= Angle + 360;
Course:= Angle
End;                                        { Course }
```

File Speed.Prc

```
Function Speed ( Lat1, Long1, Lat2, Long2, Interval: Real ): Real;

{ This function determines the ship's speed ( in knots ) between two points.
{ Modified by E Coldwell, Feb, 86 - Only use Elliptical Geometry if distance
  is more than 50 meters as computed from planer geometry }

Var

   Dist,                                  { Distance ( in m ) between points. }
   Dum1, Dum2,                            { Dummy variables }
   Vel: Real;                             { Ship's velocity ( in m/s ) }

Begin
  If (abs(lat1) > 90) or (abs(long1)>180) then Begin
    speed:= 0;
    Exit;
  End;

  Dist:= NauticalMiles ( Lat1, Long1, Lat2, Long2 );

  If Dist > 1/100 then Begin
    Sphrd ( Lat1, Long1, Lat2, Long2, Dist, Dum1, Dum2 );
    { Dist in m, Interval in s }
    Vel:= Dist / Interval;
    Speed:= Vel * 100 / 50.8; { m/s --> kt. }
  End
  Else Begin
    Vel:= Dist / Interval;   { Dist in Nautical Miles, Int in sec, Vel in NM
    Speed:= Vel * 3600;  {NM/s --> Knots }
  End;
End; { Procedure Speed }
```

File CheckMin.Fn

```
Function NextMin: Boolean;

{ This function checks whether or not the DOS clock minute has changed. }

Type RegPack = Record
                  ax,bx,cx,dx,bp,si,di,ds,es,flags: integer
               End;

Var

   RecPack:          RegPack;              { Assign record }
   ah,al,ch,cl,dh: Byte;

Begin
ah:= $2C;
With RecPack do
   Begin
   ax:= ah shl 8 + al;
   End;
Intr ( $21, RecPack );
With RecPack do
   Begin
   NextMin:= ( OldMin <> ( cx ) );
   OldMin:= cx
   End
End; { Function NextMin }


{*****************************************************************}
Function NextTenSec: Boolean;
{ This function checks whether or not the DOS clock ten seconds has changed.

Type RegPack = Record
                  ax,bx,cx,dx,bp,si,di,ds,es,flags: integer
               End;

Var

   RecPack:          RegPack;              { Assign record }
   ah,al,ch,cl,dh: Byte;
   NewTenSec       : Integer;

Begin
  ah:= $2C;
  With RecPack do
    ax:= ah shl 8 + al;
  Intr ( $21, RecPack );
  With RecPack do
    NewTenSec:= (dx shr 8) div 10;
  NextTenSec:= (OldTenSec <> NewTenSec);
  OldTenSec:= NewTenSec;
End; { Function NextTenSec }
```

File ReadFunctionKeys.PRC

```
Function ReadFunctionKeys : FunctionKeyType;

{ This function determines which function key has been pressed. }
{ Last Modified: Feb 12, 1986 by E Coldwell }

Var
   Ch: Char;
   Num: Integer;

Begin
  ReadFunctionKeys:= none;
  If KeyPressed then Begin
    Read ( Kbd, Ch );

    If Ch in ['1','2','3','4'] then Begin
      num:= ord(ch) - ord('0');
      If StationOk [num ] then
        Weight [num]:= 1 - Weight [num]
      Else
        Weight [num]:= 0;
    End; {If}

    If ( Ch = #27 ) and KeyPressed then Begin
      Read (Kbd, ch);
      Case Ch of
        #59 : ReadFunctionKeys:= F1;
        #60 : ReadFunctionKeys:= F2;
        #68 : ReadFunctionKeys:= F10;
      End; {Case}
    End { If ( Ch = #27 ) }

  End { If KeyPressed }
End; { Function ReadFunctionKeys }
```

File Exit.Prc

```
Procedure Exit ( Var Done: Boolean );
Var
  key : char;

Begin
  Writeln;
  Write ( 'Are you sure you wish to terminate this program ? [Y/N]   ' );
  While WhereX < 80 do
    Write (' ');
  GotoXY (57, WhereY);
  key:= ' ';
  Repeat until keypressed;
  Read (kbd,key);
  If key in ['Y','y'] then
    Done:= True
   else Begin
    GotoXY ( 1, WhereY);
    ClrEol;
    GotoXY (80, WhereY - 1);
    Done:= False
  End { else }
End; { Procedure Exit }
```

File OutData.Prc

```
Procedure OutData ( Var CLat, CLong: Real );
{ Modified Jan 31, 1986 by Everett Coldwell
          - Datafile format changed
           Feb 3, 1986 by Everett Coldwell
          - Write to screen only once per minute }

Var
  Cr, Dist, DDP, DDL : Real;
  I : Integer;
  Seconds: Real;

Function StrFN ( X : Real; n, m: Integer ) : Line;
Var
  S : Line;
Begin
  Str ( X:n:m , S );
  StrFN:= S;
End; { Fucntion StrFN }


Begin

  Cr:= Course ( PLat[6], Plong[6], Clat, CLong );

  Seconds:= Ctime-Ptime [6];
  If Seconds < 0 then
    Seconds:= Seconds + 86400.;
  Sp:= Abs ( Speed ( PLat [6], PLong [6], CLat, CLong, Seconds));

  OutRecord:= OutRecord + StrFN ( Int(CLat),4,0 )
                        + StrFN ( Abs(60*Frac(CLat)),7,3 )
                        + StrFN ( Int(CLong),5,0 )
                        + StrFN ( Abs(60*Frac(CLong)),7,3 )
                        + StrFN ( Cr,8,2 )
                        + StrFN ( Sp,8,2 );

  Write ( Int(CLat):4:0, #248, Abs(60*Frac(CLat)):6:3, #39 );
  Write ( Int(CLong):5:0, #248, Abs(60*Frac(CLong)):6:3, #39 );
  Write ( '  ', Cr:7:2 );
  Write ( '  ', Sp:7:2 );
  Write ( ' ' );
  For I:= 1 to 4 do
    If ( Range [i] > 0 ) and (Weight [i] > 0) and StationOK [i] then Begin
      With RefStn [i] do
        Sphrd ( CLat, CLong, latitude, longitude, Dist, DDP, DDL );
      Write (' ', Round ( Range [i] - Dist ):4)
     End { If }
    Else
      Write ('     ');

  If NextMin Then
    Writeln;
  While WhereX < 80 do
```

```
    Write (' ');
End; { OutData }
```

File Headings.Prc

```
Procedure Headings;
{ Modified Jan 31, 1986 by E. Coldwell - deleted headings to data file }

Begin
  GotoXY ( 22, 1 );
  WriteLn ( 'MiniRanger Positioning System' );
  GotoXY ( 1, 1 );
  WriteLn ( '     Time                    Ship''s              Course    Speed',
            '     Range Residuals' );
  Write   ( '(DY:MT HH:MM:SS)  Latitude    Longitude    (degs)   (Knots)',
            '       (meters)' );
End; { Headings }
```