PROGrams SAtNAV AND lINK Descriptions AND USER’S GUIDES

SEE HEAN QUEK

June 1983
PREFACE

In order to make our extensive series of technical reports more readily available, we have scanned the old master copies and produced electronic versions in Portable Document Format. The quality of the images varies depending on the quality of the originals. The images have not been converted to searchable text.
PROGRAMS SATNAV AND LINK
DESCRIPTONS AND USER'S GUIDES

by

SEE HEAN QUEK

Department of Surveying Engineering
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada
E3B 5A3

Technical Report 98

June 1983
PREFACE

This technical report contains the second and third contributions in a series of reports detailing the development of a system for microcomputer control of a CMA 722B satellite Doppler positioning receiver.
ACKNOWLEDGEMENT

This work would not have been possible without the support of Dr. D.E. Wells and Dr. R.B. Langley. Improvements to the CMA 722B/Apple II+ operating system were funded by an operating grant held by Dr. Wells from the Natural Sciences and Engineering Research Council of Canada.
PROGRAM SATNAV

DESCRIPTION AND USER'S GUIDE
ABSTRACT

This supplement describes the changes and improvements to the Digital Data Recorder program (RECEIVER) developed originally by Mark S. Lord, and described in Technical Report 88 of the Department of Surveying Engineering, University of New Brunswick.

These changes have been implemented in a new version of the RECEIVER program called SATNAV.

SATNAV (version 3.0) optimizes the storage of the Doppler data on the diskette and has a realtime majority voting capability for the broadcast satellite message. The following is a list of the new features available under SATNAV (version 3.0).

(1) Realtime majority voting of satellite message.
(2) Realtime accumulation of 30-second Doppler counts.
(3) Validation of majority-voted satellite message.
(4) Verification of 30-second Doppler count sufficiency for pass computation.
(5) Optimization of data storage, i.e., choice of saving only the majority-voted file instead of the much larger raw data file.
(6) Addition of pass diagnostic messages and improved screen display.
(7) Realtime verification of satellite tracked.
(8) Manual rejection of satellite pass.

A user's guide to SATNAV is in Appendix I.
# PROGRAM SATNAV

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. SATNAV</td>
<td></td>
</tr>
<tr>
<td>2.1 Realtime Majority Voting</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Accumulation of 30-Second Dopplers</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Validation of Satellite Message</td>
<td>3</td>
</tr>
<tr>
<td>2.4 30-Second Doppler Sufficiency Test</td>
<td>4</td>
</tr>
<tr>
<td>2.5 Data Storage</td>
<td>5</td>
</tr>
<tr>
<td>2.6 Screen Display</td>
<td>6</td>
</tr>
<tr>
<td>2.7 Realtime Identification of Satellite</td>
<td>6</td>
</tr>
<tr>
<td>2.8 Pass Rejection Capability</td>
<td>7</td>
</tr>
<tr>
<td>2.9 Description of the Majority-Voted File</td>
<td>7</td>
</tr>
<tr>
<td>3. Conclusions and Recommendations</td>
<td>11</td>
</tr>
<tr>
<td>References</td>
<td>12</td>
</tr>
<tr>
<td>Appendix I - User's Guide</td>
<td>13</td>
</tr>
<tr>
<td>Appendix II - Components of the CMA 722B/Apple II+ System</td>
<td>29</td>
</tr>
<tr>
<td>Appendix III - SATNAV program listing</td>
<td>30</td>
</tr>
</tbody>
</table>

# PROGRAM LINK

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>78</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>79</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>80</td>
</tr>
<tr>
<td>2. LINK</td>
<td></td>
</tr>
<tr>
<td>2.1 1200 Baud Communication</td>
<td>81</td>
</tr>
<tr>
<td>2.2 Additional Features</td>
<td>81</td>
</tr>
<tr>
<td>3. Conclusions and Recommendations</td>
<td>83</td>
</tr>
<tr>
<td>References</td>
<td>84</td>
</tr>
<tr>
<td>Appendix I - User's Guide</td>
<td>85</td>
</tr>
<tr>
<td>Appendix II - Components of the Apple II+ System</td>
<td>94</td>
</tr>
<tr>
<td>Appendix III - LINK Program listing</td>
<td>95</td>
</tr>
</tbody>
</table>

vi
1. INTRODUCTION

In 1982, an economical digital recording system was devised as an alternative to the punched paper tape for the CMA 722B Transit satellite receiver. The system software and hardware specifications are detailed in Lord [1982].

Several modifications have been made to both hardware and software components of the system to improve its utility as an intelligent recording device. This supplementary report highlights the alterations made to the program, its implementation, and contains a user's guide to the software (SATNAV). With the exception of the changes mentioned in this report, the rest of the operating environment of the system is as described in Lord [1982] (pp. 28-44).
2. SATNAV

SATNAV is the latest version of the RECEIVER program developed originally by Lord [1982]. The program has been extensively revised and now contains twice as much code which is partitioned into three source text files (Appendix III). A description of the features available under SATNAV (version 3.0) follows.

2.1 Realtime Majority Voting

The satellite broadcast message consists of a set of fixed parameters and a set of variable parameters [Stansell, 1978]. The program majority votes the incoming satellite message in realtime using two 9-character word arrays. The first two paragraphs of the satellite broadcast message are stored in these two separate arrays. Majority voting is done on a digit by digit basis. When the next digit in the third paragraph is received, it is compared with the corresponding digits in the two previously received paragraphs. If the number from the first and second paragraphs agree, the digit from the third data set and subsequent data paragraphs are ignored. If a disagreement exists, a three-way comparison is made and the odd one dropped. If they all disagree, the digit from the latest paragraph is dropped and the process is repeated until two are in agreement.

There are 28 lines in the majority-voted arrays. The variable and fixed parameters each occupy one half of the array. Hence there exists space for the variable parameters from the period \((t_k - 4)\) minutes to \((t_k + 22)\) minutes, with \(t_k\) being the lock-on time of the satellite pass. As for the fixed parameters, only the first 14 of the received parameters are kept. This can be extended to 15 to allow for the detection of the
satellite message injection during a satellite pass.

The present version of SATNAV accepts only numeric data for majority voting. Hence the injection flag (three rows of equal signs) is ignored. This may result in inconsistent majority-voted data on injection passes. It would be possible to further modify SATNAV to detect and correctly handle injection passes, but this has not yet been done.

2.2 Accumulation of 30-Second Dopplers

Many Doppler processing programs are based on 30-second Doppler data rather than on the short 4.6-second counts that the CMA 722B provides. SATNAV extracts and stores, in realtime, the long (approximately) 30-second Doppler counts from the array of 4.6-second accumulated Doppler counts. Both 150 MHz and 400 MHz, 30-second Doppler counts are extracted and kept in a 9-character word array of 32 rows.

2.3 Validation of Satellite Message

In an effort to weed out bad passes due to errors in the received satellite message, SATNAV has the capability of checking and testing satellite ephemerides. Prior to writing the pass on the diskette, SATNAV does the following:

(a) If any of the digits in the 9-character words are undefined, all the digits are set to zero.

(b) If any of the fixed parameters have been zeroed, a '9' appears in the line error code column in the majority-voted file (see Section 2.9).

(c) If any of the fixed parameters do not contain an '8' or a '9' as the first digit, an error flag '9' appears in the line error
code column in the majority-voted file.

(d) If all the fixed parameters pass the above data format checks, the quality of the message is assessed by decoding and testing for the following:

1) time of satellite perigee \(0 \leq t_p < 1440\);
2) rate of change of mean anomaly \(3 \leq \Omega \leq 4\);
3) argument of perigee at time of perigee \(0 \leq \omega \leq 360\).

A '1' appearing in the line error code indicates a warning and usually appears when the parameter is zeroed and is not detrimental to the pass computations. A '9', on the other hand, indicates a fatal error in the received broadcast message and will appear in the majority-voted file if the no-reject option for a pass with a bad message is chosen. Regardless of the option, SATNAV assesses the quality of the majority-voted message and displays the final verdict in the message area on the screen (see Appendix I, figure I-2).

2.4 30-Second Doppler Sufficiency Test

The advantage of tallying the number of 30-second Doppler counts accumulated in a pass is the ability to assess the quality of the resulting position determinations based on the number of Doppler observations available. Due to interference, single frequency Doppler counts may result, leading to seemingly good passes. To remove single frequency measurements, a criterion is imposed of a maximum difference of 1500 Doppler counts between the 150 MHz and 400 MHz Dopplers. Care has been taken in implementing this constraint because, if erroneously applied, it will result in loss of valuable data. A straight difference in the recorded long Dopplers can result in losses of whole 2-minute paragraphs of
data. Hence SATNAV uses the Doppler counts reset every (approximately) 30 seconds, rather than the Dopplers accumulated up to two minutes to implement this restriction.

All actual two-frequency 30-second Doppler counts with differences greater than 1500 counts are zeroed. The remaining counts are tallied and if they do not exceed the selected minimum (program option; default 10), the pass may be rejected. In either case, if the number of counts falls below the preset minimum, a warning to that effect appears in the message area of the screen.

2.5 Data Storage

The raw and majority-voted data are buffered until the end of the satellite pass and then dumped onto the diskette. SATNAV has the user selected capability of either dumping the raw and majority-voted data or only the majority-voted data. Raw and majority-voted data are kept in two separate files. The composition of the majority-voted files is given in Section 2.9. Each majority-voted code file occupies 2 data blocks on a diskette that has a maximum capacity of 270 blocks. Therefore it is possible to store about 135 passes per diskette.

If raw data also is to be stored on the diskette, the number of passes drops dramatically to about 20. The actual number varies with the number of paragraphs per pass that was tracked. To enable a longer period of unattended operation, the boot diskette may be removed, and in its place a blank PASCAL formatted diskette can be inserted. This diskette will be used once the primary data diskette is filled. Based on collecting only the majority-voted data and on an average of 30 good passes per day, the system may be left unattended for about 9 days before a change of data
diskettes is required.

2.6 Screen Display

The original screen display has been modified to accept 9-digit Doppler counts. This was made possible by squeezing the program nesting column.

Currently only 7-digit data are accessed from the CMA 722B. Two zero digits are appended to the 7 digits to make the Dopplers conform to the 9-digit format. In order to use 9-digit data, the interface to the CMA 722B must be changed from the present parallel interface (connected to the CMA 722B computer interface board) to a serial interface (connected to the CMA 722B serial interface board). Within the Apple, the serial interface should be accessed by an input interrupt driven buffer. For the purposes of displaying incoming data from the CMA 722B, the digits should be packed into 4-digit words. This should improve the execution speed of the program.

2.7 Realtime Identification of Satellite

SATNAV constantly attempts to identify, in realtime, the satellite number. When it manages to decode a valid satellite number, it displays it on the screen for possible manual rejection of the pass via the ESC unlock command sequence (see Appendix I). This facility was developed to allow for the future use of an alert table to select desired satellites or passes.
2.8 Pass Rejection Capability

The original design of the CMA 722B does not allow software controlled rejection of a satellite pass. To have this facility, we have constructed a feedback board based on the diagrams and description in a report by Ken Hill [1980], and installed it in the empty slot in the CMA 722B. The end of pass command, used to reject a satellite pass, is issued by SATNAV through the game I/O port of the Apple. Wiring diagrams for the CMA 722B are as indicated in Lord [1982] (page 26). A separate cable leading from the 48-pin female edge connector to the game I/O port has been constructed. Table 2.1 shows the pin connections for this cable.

<table>
<thead>
<tr>
<th>Game I/O Connector on Apple Motherboard</th>
<th>HP Edge Connector on CMA 722B Interface Cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 5 to Pin 22</td>
<td></td>
</tr>
<tr>
<td>Pin 12</td>
<td>Pin 45</td>
</tr>
<tr>
<td>Pin 13</td>
<td>Pin 46</td>
</tr>
<tr>
<td>Pin 14</td>
<td>Pin 47</td>
</tr>
<tr>
<td>Pin 15</td>
<td>Pin 48</td>
</tr>
</tbody>
</table>

Table 2.1
Pass Rejection Feedback Signal Wiring Connections.

2.9 Description of the Majority-Voted File

The majority-voted file is comprised of the majority-voted matrix, 30-second Dopplers, information codes, and the line error code (see Figure 2-1). It contains a 31 by 4 matrix of numbers, preceded by a line giving the date and time at lock-on. The first two columns of the matrix contain the 30-second accumulated Doppler counts at 400 MHz and 150 MHz, respectively. The first 28 rows of the third column comprise the majority-voted broadcast satellite message. Of these, the first 14 rows are the ephemeral parameters spanning $(t_k - 4)$ minutes to $(t_k + 22)$
minutes; with \( t_k \) being the lock-on time of the pass. The remaining 14 rows hold the fixed parameters. The 29th row is, at present, uncoded and can be used to indicate satellite message injection during the satellite pass, if so desired. The second to last row (30th from the top) contains a user defined code describing station, receiver and/or user. The last row of the third column gives the options used in accumulating the Doppler data. The breakdown of the coding is given in Figure 2-1.

The line error-code vector is the fourth column of the majority-voted matrix. The 'health' of the data in each row of the matrix is identified by a corresponding row in the line error-code column. If a '1' appears in the 4th column, it denotes a zeroed or undefined parameter in the majority-voted message; a '9' indicates bad or missing fixed parameters or insufficient Doppler counts. Under the no-reject option, a bad fixed parameter results in a '9' in the penultimate row of the fourth column. A '9' in the last row indicates insufficient Doppler counts have been recorded.
**FIGURE 2-1**

Majority-voted matrix

<table>
<thead>
<tr>
<th>Date and Time stamp</th>
<th>Variable parameter at lock-on-time</th>
<th>Majority-voted broadcast message (MVBM). Lines 1 to 14 are the variable parameters and lines 15 to 28 are the fixed parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>83/02/11-5 16:51:55</td>
<td>081197000 081198300 090010014 0</td>
<td>= begin =</td>
</tr>
<tr>
<td>081197000 081198300 090010014 0</td>
<td>151452700 151456100 600130134 0</td>
<td>= begin =</td>
</tr>
<tr>
<td>151452700 151456100 600130134 0</td>
<td>234337300 234343300 610250334 0</td>
<td>= end =</td>
</tr>
<tr>
<td>234337300 234343300 610250334 0</td>
<td>307294400 307303000 620350593 0</td>
<td>= end =</td>
</tr>
<tr>
<td>307294400 307303000 620350593 0</td>
<td>085282900 085286600 630410894 0</td>
<td>= end =</td>
</tr>
<tr>
<td>085282900 085286600 630410894 0</td>
<td>159683000 159689000 640451231 0</td>
<td>= end =</td>
</tr>
<tr>
<td>159683000 159689000 640451231 0</td>
<td>248259900 248270200 200451580 0</td>
<td>= end =</td>
</tr>
<tr>
<td>248259900 248270200 200451580 0</td>
<td>326995200 327007500 210372147 0</td>
<td>= end =</td>
</tr>
<tr>
<td>326995200 327007500 210372147 0</td>
<td>092972200 092974100 220342470 0</td>
<td>= end =</td>
</tr>
<tr>
<td>092972200 092974100 220342470 0</td>
<td>174875900 174879600 230282744 0</td>
<td>= end =</td>
</tr>
<tr>
<td>174875900 174879600 230282744 0</td>
<td>273238800 273243700 240202965 0</td>
<td>= end =</td>
</tr>
<tr>
<td>273238800 273243700 240202965 0</td>
<td>361277200 361282300 250103090 0</td>
<td>= end =</td>
</tr>
<tr>
<td>361277200 361282300 250103090 0</td>
<td>104383900 104383300 060013145 0</td>
<td>= end =</td>
</tr>
<tr>
<td>104383900 104383300 060013145 0</td>
<td>196419700 196418100 0000000000 1</td>
<td>= end =</td>
</tr>
<tr>
<td>196419700 196418100 0000000000 1</td>
<td>306655000 306649000 039300580 0</td>
<td>= end =</td>
</tr>
<tr>
<td>306655000 306649000 039300580 0</td>
<td>404747400 404736600 0837537250 0</td>
<td>= end =</td>
</tr>
<tr>
<td>404747400 404736600 0837537250 0</td>
<td>115330600 115326200 818388360 0</td>
<td>= end =</td>
</tr>
<tr>
<td>115330600 115326200 818388360 0</td>
<td>216014400 216005500 800199840 0</td>
<td>= end =</td>
</tr>
<tr>
<td>216014400 216005500 800199840 0</td>
<td>335331000 335316400 800047520 0</td>
<td>= end =</td>
</tr>
<tr>
<td>335331000 335316400 800047520 0</td>
<td>440383400 440364800 807449850 0</td>
<td>= end =</td>
</tr>
<tr>
<td>440383400 440364800 807449850 0</td>
<td>8000000000000000 813016500 0</td>
<td>= end =</td>
</tr>
<tr>
<td>8000000000000000 813016500 0</td>
<td>0000000000000000 9000005250 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 9000005250 0</td>
<td>0000000000000000 8000131180 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 8000131180 0</td>
<td>0000000000000000 823901100 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 823901100 0</td>
<td>0000000000000000 8000301400 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 8000301400 0</td>
<td>0000000000000000 817200420 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 817200420 0</td>
<td>0000000000000000 809999140 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 809999140 0</td>
<td>0000000000000000 802066000 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 802066000 0</td>
<td>0000000000000000 8000000000 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 8000000000 0</td>
<td>8000000000000000 111222555 0</td>
<td>= end =</td>
</tr>
<tr>
<td>8000000000000000 111222555 0</td>
<td>0000000000000000 511110030 0</td>
<td>= end =</td>
</tr>
<tr>
<td>0000000000000000 511110030 0</td>
<td>[400 MHz] [150 MHz] [MVBM] [LEC]</td>
<td>= end =</td>
</tr>
</tbody>
</table>

Continue next page
Counts - Accumulated 30-second 400 MHz and scaled 150 MHz nine-digit Doppler counts.

   Coding:  0 - no error detected
            1 - non-fatal error (warning)
            9 - fatal error in matrix (bad MVBM or insufficient counts)

[OPC] - Option Code (511110030)
   Coding sequence digits:
   1 : 5 - minimum number of paragraphs
   2 : 1 - clear Doppler counts when difference exceeds 1500 counts
   3 : 1 - check for sufficiency of counts
   4 : 1 - reject pass if MVBM fails tests
   5,6 : 10 - minimum counts constituting a good pass (2-digits)
   7 : 0 - not used
   8,9 : 3.0 - SATNAV version code (2-digits)

(see section I.2.1a to I.2.1d for further description on options)
3. CONCLUSIONS AND RECOMMENDATIONS

The structure of the SATNAV program under the Apple PASCAL operating system enables easy tailoring of the program to suit user requirements where options are lacking. SATNAV and the Apple II+ microcomputer allow an economical alternative to other methods for controlling and recording data from the CMA 722B Transit satellite receiver.

Further improvements, in descending order of importance, along the lines mentioned below will undoubtedly enhance this system further.

(a) Acquisition of 9-digit data.
(b) Use of alert table to select optimal passes.
(c) Generation of alerts on the Apple.
(d) Computation of satellite fixes on the Apple.
REFERENCES


APPENDIX I

Program : SATNAV

Author : S.H. Quek

Language : PASCAL

Compiler : APPLE PASCAL (1.1)

Type : Interactive

Purpose : Acquisition and Storage of Satellite Doppler Data from the CMA722B Satellite Position/Navigation Receiver

Date : May 1983
The following describes the start-up, or "booting", procedure for the execution of the SATNAV program.

1. Before turning on the power slide the diskette named TRACK into disk drive 1 (as identified by the label on the front of the drive) and a Pascal-formatted blank diskette into the other drive.

2. Power on the Apple II+.

3. The screen should display the current time, i.e.
   year/month/day hour/min/second. This should be in Universal Time (UT). A request to type 'I' sometimes appears.

4. If system fails to boot, try swapping the diskettes in the disk drives. If problem persist, contact author.

5. If booting is successful the system is now in the APPLE PASCAL Command Mode. To enter the Execution Mode, type 'X'. The system then asks for the name of the file (program) you wish to execute.

6. Type 'GO' to execute the front program. Disk drive 1 will whirr a bit and a menu of available programs will be displayed.

7. Type '1' to access the SATNAV program (Figure I-1 will appear on the screen).

* (Note: Type 'X' means hitting the X key on the keyboard without the quotes)
FIGURE I-1

Welcoming message to SATNAV
Alternatively, one may directly access the SATNAV program from the PASCAL Command Mode by typing 'SATNAV' instead of 'GO' in response to the file-name prompt.

I.2 SATNAV

The following are the extended descriptions of the various options available with the 3.0 version of the SATNAV program.

I.2.1 Programmed Defaults

To accept all the programmed defaults (given below) respond with 'N'o to the request to change defaults. To enter new values or view option defaults, type 'Y'es. The following are the list of options available:

a) Minimum Paragraphs

The user has the option of setting a minimum number of 2-minute paragraphs of Doppler data that have to be accumulated before the pass is to be saved on the diskette. This reduces the amount of marginal data stored on the diskette and frees diskette space for more useful passes.

(Default := 5)

b) Validation of Broadcast Ephemerides

SATNAV checks the broadcast majority-voted message for a valid time of perigee, rate of change of mean anomaly and the argument of the perigee. The user has the option of rejecting a pass if the message fails these tests. If the no-reject option is selected, the
failure of the majority-voted message is denoted by
'9' in the second last element of the Line Error Code
(LEC) column vector associated with each
majority-voted matrix (see Figure I-3).
(Default := Yes)

c) Zero Dopplers

The accumulated 30-second counts displayed in the
majority-voted matrix (see Figure I-3) are derived
from the accumulated 4.6 second counts in the 2-minute
paragraphs. Due to weak signals or complete signal
loss one of the frequency channels (usually the 400
MHz channel) may unlock during a pass. If relock
occurs within the 2-minute paragraph the accumulated
Doppler counts between the two channels will be offset
by a fixed amount. SATNAV uses the actual 30-second
counts (not the displayed accumulated 30-second
counts) to reject Doppler measurements if the
difference between the 400 MHz and 150 MHz Doppler
counts exceeds 1500 counts. This option can be used to
clear one-frequency Doppler measurements.
(Default := Yes)

d) Minimum Number of Counts

The number of accumulated 30-second Doppler counts is
totalled prior to writing the pass on the diskette. If
an insufficient number of counts has been collected,
SATNAV uses this option to reject the pass. If a
no-reject option is selected and the number of counts
falls below the preset limit, a '9' appears as the
FIGURE I-3

Majority-voted matrix

```
83/02/11-5 16:51:55 <- [Date and Time stamp]
081197000 081198300 090010014 0 = begin =
151452700 151456100 600130134 0
234337300 234343300 610250334 0 <- [Variable parameter at lock-on-time]
307294400 307303000 620350593 0
085282900 085286600 630410894 0
159683000 159689000 640451231 0
248259900 248270200 200451580 0
326995200 327007500 240202965 0 [ Majority voted broadcast message (MVBM) ]. Lines 1 to 14 are the variable parameters and lines 15 to 28 are the fixed parameters.]
0929772200 092974100 220342470 0
174875900 174879600 240202965 0
273238800 273243700 240202965 0
361277200 361282300 250103090 0
104383900 104383300 060013145 0
196419900 196418100 000000000 1
306655000 306649000 039300580 0
404747400 404736600 837537250 0
115330600 115326200 818388360 0
216014400 216005500 800047520 0
335331000 335316400 800301400 0
440383400 440364800 817200420 0
000000000 000000000 809999140 0
000000000 000000000 900000000 0
000000000 000000000 813016500 0
000000000 000000000 900000525 0
000000000 000000000 800131180 0
000000000 000000000 823901100 0
000000000 000000000 800301400 0
000000000 000000000 817200420 0
000000000 000000000 809999140 0
000000000 000000000 802060000 0 = end =
000000000 000000000 000000000 0
000000000 000000000 81122555 0 <- [User/Stn/Rec ID code]
000000000 000000000 511110030 0 <- [OPC]
[400 MHz] [150 MHz] [MVBM] [LEC]
```

Continue next page
**Note:**

Counts - Accumulated 30-second 400 MHz and scaled 150 MHz nine-digit Doppler counts.

    Coding:  0 - no error detected
            1 - non-fatal error (warning)
            9 - fatal error in matrix (bad MVBM or insufficient counts)

[OPC] - Option Code (511110030)
    Coding sequence digits;  1 : 5  - minimum number of paragraphs
                          2 : 1  - clear doppler counts when difference exceeds 1500 counts
                          3 : 1  - check for sufficiency of counts
                          4 : 1  - reject pass if MVBM fails tests
                          5,6 : 10 - minimum counts constituting a good pass (2-digits)
                          7 : 0  - not used
                          8,9 : 3.0 - SATNAV version code (2-digits)

(see section I.2.1a to I.2.1d for further description on options)
last element of the LEC vector.
(Default := 10 counts)

e) Data type

SATNAV allows the user to select the type of data to be stored on the diskettes. Only majority-voted data (MJV) results in more passes per diskette and is usually preferred.
(Default := MJV data only)

I.2.2 USER/STN/REC

SATNAV allows the input of a user-defined nine-digit code which shows up in the majority-voted matrix (see Figure I-3). Preferably, the code is used to identify the user, observing station and the serial number of the receiver. No restriction is placed on the numerical code and the user, if he so desires, is free to use his own coding system.

Having fulfilled the last request for information by the program, you should see a display similar to Figure I-2. SATNAV now waits for the receiver to acquire signals from a satellite. The Message Area displays the assigned next pass number, the increment between pass numbers and the type of file to be used. In the mean time, the timestamp (displayed in the Message Area on the screen) is updated every two minutes. When the CMA 722B acquires a message lock on a
Screen setup during the execution of SATNAV

STATUS: SETUP  ! SATELLITE PASS  ! INPUT
PARA/LINE= 00/00  ! MONITOR PROGRAM  ! WORDS

[Message Area]

PROG! DOPPLER COUNT ! SATELLITE!
NEST! 400-MHZ: 150-MHZ! MESSAGE!
A satellite signal, data is transmitted to the Apple II+ via the Parallel Interface Adapter. The satellite number of any pass tracked is displayed at the earliest possible moment in the Message Area. The user can then reject the pass if he so wishes using the ESC Unlock command. Data displayed on the monitor during a satellite pass is stored in memory until the end of the pass. Upon completion of the pass, the Doppler data is checked and if it passes all tests selected through the options, the data is transferred to the data diskette. SATNAV then waits patiently for the next pass.

During execution of the program the keyboard will respond to only 4 user commands; all of which have to be preceded by the escape <ESC> key. The 4 user commands are as follows:

- **<Q> Quit** - Sets a flag to terminate SATNAV upon completion of the current pass. The pass will be saved before the program exits.
- **<S> Stay** - Negates the effect of a previously issued 'Quit' command.
- **<U> Unlock** - Causes SATNAV to discontinue acquiring data from the receiver and saves the current pass in the usual manner.
- **<K> Kill** - Causes immediate termination of SATNAV. User is returned to the PASCAL Command Mode.
I.3 General Comments

I.3.1 Pass Files

Two files per satellite pass are created on the diskette. The first contains the majority-voted matrix (file-name prefixed by MJV). An example of the contents of the majority-voted matrix is as shown in Figure I-3. The second contains the 2-minute paragraphs of the recorded Doppler data (file-name prefixed by PASS). Figure I-4 illustrates the first part of the contents of a PASS file.

Files can be saved either as ".TEXT" or ".CODE" files. The distinction between the two types of files lies in the use of the files. ".TEXT" files, unlike ".CODE" files, can be edited by the PASCAL System Editor. To enable the editor to read the ".TEXT" files, each file is created with a four block header. Consequently, ".TEXT" files consume available diskette space at a much faster rate than ".CODE" files.

The option of the type of files to be created, along with the starting number suffixed to each file and its increment between passes are stored on the TRACK diskette in a file named ’TRACK:RCV.PARAM.TEXT’. Currently, changes can only be made using the System Editor.

I.3.2 Message Area

This is the 3-line block area on the screen reserved for messages produced during the normal execution of the program (see Figure I-2). The results of the various
Two-minute Paragraphs

[Date and Time stamp at beginning of 2-min paragraph]
83/02/11-5 16:53:55  <- Local lock-on-time (UT)
999999900 999999900 180150034  = begin =
011554200 011554200 090010014  .
023123000 023123400 600130134  .
034706500 034707400 610250334  [Variable parameters]
046305200 046306100 620350593  .
057919500 057920200 630410894  .
069550000 069551000 640451231  .
081197000 081198300 200451580  = end =
092861000 092863000 039300580  = begin =
104542500 104544600 837537250  .
116241900 116244500 818388360  .
127959700 127962500 800199840  .
139696500 139699700 800047520  .
151452700 151456100 807449850  .
163228900 163232500 813016500  [Fixed parameters]
175025600 175029700 900005250  .
186843500 186848100 800131180  .
198683000 198688000 823901100  .
210544700 210550100 800301400  .
222429300 222434800 817200420  .
234337300 234343300 809999140  .
246269400 246275800 802060000  = end =
258226200 258232700 000000000  .
270208500 270215400 000000000  .
282216600 282224200 000000000  .
83/02/11-5 16:53:54  <- Local lock-on-time of next
307294400 307303000 090010014  2-minute paragraph (UT)
012092700 012093100 600130134
024214400 024215300 610250334
......   etc.        ....
[400 MHz 150 MHz] [MVBM]
(7-digit Dopplers in 9-digit format)
checks on the majority-voted matrix are also displayed.

I.3.3 Diskette Maintenance

The floppy diskettes require the same precautions as cassettes or phonograph records. Amongst the many don’ts are the following:

a) Do not write on the diskettes.
b) Do not leave diskettes lying around unprotected.
c) Do not let the door of the disk drive snap shut. This may pinch the diskette. Close door gently without forcing it.
d) Do not bend or crimp diskettes.
e) Do not set anything on top the diskettes.
f) Do not contaminate the diskettes or drives with dust, coffee, chemicals, soda pop, etc.
g) Do not use diskettes as towels, bookmarks or Frisbees.
h) Do not store diskettes in places where they are liable to become hot.

Lastly

i) NEVER NEVER NEVER power off, hit CTRL RESET or remove diskettes when the disk drives are in use (as indicated by the red light).

Once a diskette is damaged, physically or otherwise, chances are that all data on the diskette is lost forever.
I.4 **System Messages**

SATNAV uses the message area on the screen to display information about the system and the satellite passes. The following are the messages that can appear and their explanations.

**UNABLE TO OPEN #4: RCV.SCREEN.TEXT**

File RCV.SCREEN.TEXT is not present on the boot diskette.

**UNABLE TO OPEN #4: RCV.PARAM.TEXT**

File RCV.PARAM.TEXT containing the initial parameters relating to the satellite pass file names is not present on the boot diskette.

**MEMAVAL AT SET UP = XXXX BYTES**

Run-time random access memory available for further program development and data storage.

**PARAMS - PPPP(MMM)XXX.CCC(YY)**

- **PPPP** - prefix attached to pass files containing raw data.
- **MMM** - prefix attached to majority-voted files.
- **XXX** - number assigned to next pass to be saved.
- **CCC** - type of file - either .TEXT or .CODE.
- **YY** - increment between pass numbers.
USER <ESC> COMMANDS: Q, U, S, K

Acceptable user commands, preceded by the escape key

Q - Quit: Terminate SATNAV after current pass acquisition.
S - Stay: Negate effect of previously issued quit command.
U - Unlock: Request receiver to unlock from the current pass.
K - Kill: Immediate termination of SATNAV.

TRACKING SATELLITE NO. XX

Data is currently being acquired from satellite XX.

LAST PASS DELETED - BAD MJV

The last recorded pass was rejected because it did not satisfy all the requirements of a majority-voted message.

MJV MESSAGE FAILS CHECKS

Similar to the message above except the pass is not rejected but saved on the diskette.

PASS DELETED [X] - BAD COUNTS

The last recorded pass was rejected because it only had X number of 30-second Dopplers.

WARNING - BAD DOPPLERS

Similar to the above except that the no-reject option for passes with insufficient 30-second Doppler counts has been selected.
NEW FILE = # D:PPPP XXX.CCC
A new file name has been concatenated from the parameters contained
in RCV.PARAM.TEXT file and SATNAV will attempt to write pass data on
diskette.

UPDATING RCV.PARAM.TEXT
Incrementing the next pass number to be used in the file
RCV.PARAM.TEXT.

NO SPACE FOR OUTPUT FILE <ESC> = KILL; <RETURN> = RETRY
Space cannot be found on any of the diskettes to write the current
pass. Insert a new diskette and press RETURN to save pass data or
press ESC to delete current pass data.

LAST PASS = # D:PPPP XXX.CCC
Last recorded pass on drive # D in the current tracking session.
APPENDIX II

The following is the list of hardware components required by the SATNAV program.

1. Apple II+ microcomputer (with game I/O ports interfaced as in Table 2.1).
2. Video monitor.
3. Disk controller, and two disk drives.
4. 16K language board.
5. Model 7424 calendar/clock module.
7. Canadian Marconi portable NNSS antenna and antenna cable.
9. Special cable linking CMA 722B data output port and Apple II+ parallel interface adapter and game I/O ports.
Files

SNAPP2: SATLITE.TEXT - interface routine for CMA 722B.

SNAPP2: SATBCK31.TEXT - include file*

SNAPP2: SATBCK32.TEXT - include file

SNAPP2: SATBCK33.TEXT - include file

SNAPP2: SATNAV3.TEXT - main program

*Include files are TEXT files which are inserted into the main program during compilation. SATNAV spans more than the maximum size allowable for a single TEXT file.
; MACRO TO POP 16-BIT RETURN ADDRESS:

.MACRO POP
PLA
STA %1
PLA
STA %1+1
.ENDM

; MACRO TO PUSH 16-BIT RETURN ADDRESS:

.MACRO PUSH
LDA %1+1
PHA
LDA %1
PHA
.ENDM

; MEMORY MAP FOR 6821 PERIPHERAL INTERFACE ADAPTER:

PIASLOT .EQU 7 ;APPLE SLOT NUMBER OF PARALLEL INTERFACE CARD
PIABASE .EQU <PIASLOT*10>+0C080
PIADRA .EQU PIABASE+0 ;SIDE "A" DATA DIRECTION REGISTER
PIAPRA .EQU PIABASE+0 ;SIDE "A" PERIPHERAL INTERFACE REGISTER
PIASRA .EQU PIABASE+1 ;SIDE "A" STATUS REGISTER
PIACRA .EQU PIABASE+1 ;SIDE "A" COMMAND REGISTER
PIADRB .EQU PIABASE+2 ;SIDE "B" DATA DIRECTION REGISTER
PIAPRB .EQU PIABASE+2 ;SIDE "B" PERIPHERAL INTERFACE REGISTER
PIASRB .EQU PIABASE+3 ;SIDE "B" STATUS REGISTER
PIACRB .EQU PIABASE+3 ;SIDE "B" COMMAND REGISTER

; SPECIAL SYSTEM MONITOR LOCATIONS:

IRQVECTR .EQU OFFFE ;BASE ADDRESS OF IRQ/BRK INTERRUPT VECTOR
LANGCARD .EQU 0C080 ;BASE ADDRESS FOR SLOT#0 = LANGUAGE-CARD

; PASCAL-SUPPLIED ZERO-PAGE TEMPORARY WORK AREAS:

RTADDR .EQU 00 ;SAVE AREA FOR PASCAL RETURN ADDRESS
STRING .EQU 02 ;USED TO HOLD INDIRECT ADDRESS FOR READPIA

; ROUTINE TO INITIALIZE PIA AND BUFFER QUEUE:

.PROC INITPIA ;ROUTINE TO INITIALIZE PIA HANDLING
.DEF OLDIRQ
.REF QFWDPTR,QBKWPTR,QBYTE1,QBYTE2,IRQHANDL
START  SEI   ;DISABLE INTERRUPTS UNTIL DONE
        POP  RTADDR ;POP RETURN ADDRESS FROM STACK
        LDA  #00   ;CLEAR ACCUMULATOR
        STA  PIACRA ;REQUEST ACCESS TO DDRA
        STA  PIADRA ;SET ALL BITS FOR INPUT
        STA  PIACRB ;REQUEST ACCESS TO DDRB
        STA  PIADRB ;SET ALL BITS FOR INPUT
        LDA  #05   ;LOAD IN COMMAND BITS
        STA  PIACRA ;SET UP COMMAND REGISTER A
        LDA  #04   ;LOAD IN COMMAND BITS
        STA  PIACRB ;SET UP COMMAND REGISTER B
        LDA  #00   ;LOAD INITIAL VALUE FOR BCKWD POINTER
        STA  QBKWPTR ;SAVE BACKWARD POINTER
        LDA  #01   ;SET FWD POINTER TO ONE > THAN QBKWPTR
        STA  QFWDPTR ;SAVE FORWARD POINTER
        LDA  LANGCARD+OB ;REMOVE WRITE LANG-CARD WRITE-PROTECT
        LDA  LANGCARD+OB ;THIS INSTRUCTION HAS TO BE DONE TWICE
        LDA  IRQVECTR ;GET LSB OF CURRENT IRQ VECTOR
        STA  OLDIRQ ;SAVE FOR INTERRUPT HANDLER
        LDA  IRQVECTR+1 ;GET MSB OF CURRENT IRQ VECTOR
        STA  OLDIRQ+1 ;SAVE FOR INTERRUPT HANDLER
        LDA  IRQADR ;GET MSB OF IRQ ROUTINE ADDRESS
        STA  IRQVECTR ;STORE IN MSB OF IRQ VECTOR
        LDA  IRQADR+1 ;GET LSB OF IRQ ROUTINE ADDRESS
        STA  IRQVECTR+1 ;STORE IN LSB OF IRQ VECTOR
        LDA  LANGCARD+B ;WRITE PROTECT THE LANGUAGE-CARD AGAIN
        CLI   ;ENABLE INTERRUPTS AGAIN
        PUSH  RTADDR ;PUSH RETURN ADDRESS BACK ONTO STACK
        RTS   ;RETURN TO CALLING PROGRAM
OLDIRQ .WORD 0000 ;SAVE AREA FOR ORIGINAL MONITOR IRQ VECTOR
IRQADR .WORD IRQHANDL ;ADDR OF INTRPT ROUTINE, LO-BYTE FIRST

; PROCEDURE TO DISABLE PIA INTERRUPTS AND RESTORE IRQ/BRK VECTOR

.PROC RESETIRQ ;CLEANUP ROUTINE FOR END-OF-PROCESSING
.REF  OLDIRQ

START  SEI   ;DISABLE INTERRUPTS
        LDA  #00   ;LOAD CMD WORD FOR PIA = NO INTRPTS ALLOWED
        STA  PIACRA ;STORE IN A-SIDE COMMAND REGISTER
STA PIACRB ;STORE IN B-SIDE COMMAND REGISTER
LDA LANGCARD+OB ;REMOVE WRITE LANG-CARD WRITE-PROTECT
LDA LANGCARD+OB ;THIS INSTRUCTION HAS TO BE DONE TWICE
LDA OLDIRQ ;GET LSB OF ORIGINAL IRQ ADDRESS
STA IRQVECTR ;STORE IN IRQ VECTOR
LDA OLDIRQ+1 ;GET MSB OF ORIGINAL IRQ ADDRESS
STA IRQVECTR+1 ;STORE IN IRQ VECTOR
LDA LANGCARD+S ;WRITE PROTECT THE LANGUAGE-CARD AGAIN
RTS ;RETURN TO CALLING PROGRAM

; PROCEDURE TO RETURN THE NEXT "WORD" FROM THE QUEUE:

.PROC GETWORD,1 ;PROC TO REPLACE CHAR[4] PARAM WITH 4 DIGITS
.DEF IRQHANDL,QBYTE1,QBYTE2,QBKWPTR,QFWDPTR
.REF OLDIRQ

EMPTYCHR .EQ 20 ;EMPTY QUEUE INDICATOR CHARACTER = SPACE

START POP RTADDR ;SAVE PASCAL RETURN ADDRESS
POP STRING ;SAVE ADDRESS OF STRING PARAMETER
LDY #00 ;USE Y AS STR INDEX - SET TO "LENGTH" BYTE
LDX QBKWPTR ;GET BACKWARD POINTER FOR BUFFER QUEUE
INX ;POINT TO NEXT WORD IN BUFFER
CPX QFWDPTR ;CHECK FOR EMPTY QUEUE
BNE GETBYTE1 ;BRANCH IF NOT EMPTY

UNDFLOW LDA #00 ;SET LENGTH OF STRING TO ZERO
STA @STRING,Y ;STORE A SPACE CHARACTER
BEQ EXITGET ;ALWAYS BRANCH (TO EXIT)

GETBYTE1 LDA #04 ;SET LENGTH OF STRING TO 4 BYTES
STA @STRING,Y ;SAVE IN "LENGTH" BYTE
LDA QBYTE1,X ;GET FIRST HALF OF INPUT WORD FROM BUFFER
LSR A ;SHIFT UPPER NIBBLE TO LEFT SIDE OF ACC
LSR A
LSR A
ORA #30 ;CONVERT TO ASCII
CMP #3A ;CHECK FOR NON-NUMERIC DIGIT
BMI ST1 ;BRANCH IF DIGIT IN RANGE 0->9
CLC ;CLEAR CARRY FOR ADD
ADC #07 ;CONVERT DIGIT TO HEX CHAR A->F

ST1 INY ;POINT AT FIRST BYTE OF STRING
STA @STRING,Y ;SAVE AS FIRST CHARACTER IN STRING
LDA QBYTE1,X ;GET ORIGINAL VALUE AGAIN
AND #OF ;ISOLATE LOWER NIBBLE
ORA #30 ;CONVERT TO ASCII
INY Point at second byte of string
STA @STRING,Y ;Save as second character in string

GETBYTE2 LDA QBYTE2,X ;Get 2nd half of input word from buffer
LSR A ;Shift upper nibble to left side of ACC
LSR A
LSR A
ORA #30 ;Convert to ASCII
INY ;Point at third byte of string
STA @STRING,Y ;Save as third character in string
LDA QBYTE2,X ;Get original value again
STX QBKWPTR ;Save new queue pointer
AND #0F ;Isolate lower nibble
ORA #30 ;Convert to ASCII
INY ;Point at fourth byte of string
STA @STRING,Y ;Save as fourth character in string

EXITGET PUSH RTADDR ;Push Pascal return address on stack
RTS ;Return to calling program

QBYTE1 .BLOCK 256 ;Queue area for first 8 bits (15-8)
QBYTE2 .BLOCK 256 ;Queue area for second 8 bits (7-0)
QFWDPTR .BYTE 00
; Pointer to next empty location in queue
QBKWPTR .BYTE 00
; Pointer to item before next input value in queue

; Interrupt-driven routine to buffer data from the PIA.
; To minimize the time required to service interrupts,
; this routine is not coded for re-entrancy. As a result,
; interrupts are left disabled while this routine executes,
; and are re-enabled by the RTI instruction.

OVFLCHAR .EQU 11 ;"Unused" seq code - overflow indicator

IRQHANDL STA SAVEACC ;Save accumulator
PLA ;Get status reg from stack
PHA ;Restore onto stack
AND #10 ;Test "B" bit
BNE NOTBRK ;Skip next section if true interrupt

NOTPIA LDA SAVEACC ;Restore accumulator contents
JMP @OLDIRQ ;Branch to monitor's IRQ/BRK routine

NOTBRK LDA PIASRA ;Was IRQ caused by PIA?
BPL NOTPIA ;If not, branch to monitor's IRQ/BRK rou
TXA ;Save index-x on stack
PHA
LDX QFWDPTR ;SET UP QUEUE POINTER IN INDEX-X
CPX QBKWPTR ;CHECK FOR FULL QUEUE
BNE SAVEDATA ;BRANCH IF QUEUE IS OK

OVERRUN LDA #OVFLCHAR ;LOAD QUEUE OVERFLOW CHARACTER
DEX ;POINT AT PREVIOUS QUEUE ELEMENTS
STA QBYTE1,X ;SAVE IN PLACE OF LAST 16-BITS IN QUEUE
STA QBYTE2,X
BNE EXITIRQ ;ALWAYS BRANCH

SAVEDATA LDA PIAPRB ;GET BITS 15-8 OF INPUT FROM PIA-B
EOR #OFF ;INVERT ALL BITS
STA QBYTE1,X ;SAVE THEM AS QBYTE1
LDA PIAPRA ;GET BITS 7-0 OF INPUT FROM PIA-A
EOR #OFF ;INVERT ALL BITS
STA QBYTE2,X ;SAVE THEM AS QBYTE2
INX ;ADVANCE QUEUE POINTER TO NEXT POSITION
STX QFWDPTR ;SAVE NEW FORWARD POINTER FOR QUEUE

EXITIRQ PLA ;RESTORE INDEX-X FROM STACK
TAX
LDA SAVEACC ;RESTORE ACCUMULATOR
RTI ;RETURN TO INTERRUPTED ROUTINE

SAVEACC .BYTE 00 ;ACCUMULATOR SAVE AREA FOR INTRPT ROUTINE

.END
********************************************************************
*                                                          *
*   FILE :  SNAPP2:SATBCK31.TEXT                            *
*                                                          *
********************************************************************
PROCEDURE CHAR9TOINT(VAR NUMBER:MESSVALUE;
VAR NUMBCHR:MESSCHAR);

(* Version: 19 July 1982 *)
(* Author: See Hean Quek *)
(* Description: *)
(* Converts nine char *)
(* variable number to long integers *)

VAR
I : INTEGER;
NO : INTEGER[9];

BEGIN
NUMBER := 0;
FOR I := 9 DOWNTO 1 DO
BEGIN
NO := ORD(NUMBCHR[I]) - 48;
IF NOT (NUMBCHR[I] IN ['0'..'9']) THEN
BEGIN
WRITELN(I,'th number illegal - CHAR9TOINT');
EXIT(PROGRAM)
END
ELSE
CASE I OF
9 : NUMBER := NUMBER + NO;
8 : NUMBER := NUMBER + NO*10;
7 : NUMBER := NUMBER + NO*100;
6 : NUMBER := NUMBER + NO*1000;
5 : NUMBER := NUMBER + NO*10000;
4 : NUMBER := NUMBER + NO*100000;
3 : NUMBER := NUMBER + NO*1000000;
2 : NUMBER := NUMBER + NO*10000000;
1 : NUMBER := NUMBER + NO*100000000
END (*CASE*)
END; (*IF AND LOOP*)
END; (* CHAR9TOINT*)

PROCEDURE MAJORITY(VAR WORD1,WORD2,WORD3:CHAR);

(* Author: Quek *)
(* Date: June 22 1982 *)
(* Description: *)
(* Comparison between three character *)
(* variables and assigns them *)
(* according to their values. *)

(* *******************************************************)
BEGIN (* MAJORITY *)
(* NON NUMERIC CASE *)
IF NOT(WORD3 IN ['0..'9']) THEN
EXIT(MAJORITY);
IF WORD1=' ' THEN
BEGIN
  WORD1 := WORD3;
  EXIT(MAJORITY)
END
ELSE IF WORD2 = ' ' THEN
BEGIN
  WORD2 := WORD3;
  EXIT(MAJORITY)
END
ELSE IF WORD1=WORD2 THEN
EXIT(MAJORITY)
ELSE IF WORD1=WORD3 THEN
BEGIN
  WORD2 := WORD3;
  EXIT(MAJORITY)
END
ELSE IF WORD2=WORD3 THEN
BEGIN
  WORD1 := WORD3;
  EXIT(MAJORITY)
END
ELSE;
END; (* MAJORITY *)

PROCEDURE MJVLINE(LNCT,PARAG:INTEGER;XLINE:DATALINE);

(*---------------------------------------------------------------
* AUTHOR : QUEK
* DATE : AUGUST 28 1982
* DESCRIPTION
* MAJORITY VOTING BY LINE
* INPUT - LINE,PARAGRAPH NUMBER AND
* MESSAGE LINE
*---------------------------------------------------------------)
VAR
ILINE : INTEGER;
WORD : CHAR;

PROCEDURE LOCSAT(ILINE:INTEGER);

(*---------------------------------------------------------------
*SUBPROCEDURE TO FIND SATELLITE* 
*NUMBER AT THE FIRST AVAILABLE *
*OPPERTUNITY. *
*---------------------------------------------------------------)
VAR STRNUM : STRING;
BEGIN
IF ILINE = 25 THEN
BEGIN
  ISAT := (ORD(MJVPASS[ILINE,6])-48)*10
             + ORD(MJVPASS[ILINE,5])-48;
  IF ISAT IN [13,14,19,48,20] THEN
      BEGIN
        SATLOC := TRUE;
        STR(ISAT,STRNUM);
        SHOWMSG(0,CONCAT('TRACKING SATELLITE NO. ',STRNUM))
      END;
END;  (* LOCSAT *)

BEGIN (* MJVLINE *)

(* TRANSFER OF MESSAGE TO ARRAY *)
FOR K := 21 TO 29 DO
BEGIN
  L := K-20;
  UNPACK9[L] := XLINE[K]
END;

IF LNCT < 8 THEN

(* SECTION FOR VARIABLE PARAMETERS *)
BEGIN
  ILINE := PARAS + LNCT - 2;  (* FIND POSITION IN ARRAY *)
  IF (ILINE <= ENDEPHEMERAL) AND (ILINE > 0) THEN
      BEGIN
        PACK9 := TEMPORARY[ILINE];
        FOR K := 1 TO 9 DO
            BEGIN
              WORD := PACK9[K];
              MAJORITY(MJVPASS[ILINE,K],
                      WORD,
                      UNPACK9[K]);
              PACK9[K] := WORD;
            END;
        TEMPORARY[ILINE] := PACK9;
      END
ELSE

(* SECTION ON FIXED PARAMETERS *)
BEGIN
  ILINE := LNCT - 9 + ENDEPHEMERAL + 1;
  IF ILINE <= MAXPRMETERS THEN
      BEGIN
PACK9 := TEMPORARY[ILINE];
FOR K := 1 TO 9 DO
BEGIN
WORD := PACK9[K];
MAJORITY(MJVPASS[ILINE,K],
WORD,
UNPACK9[K]);
PACK9[K] := WORD;
END;
TEMPORARY[ILINE] := PACK9;
END;
IF NOT SATLOCK THEN LOCSAT(ILINE);
END;
END; (* MJVLINE *)

PROCEDURE CLEARMJVFILE;
(*-------------------------------------------------------------
 * AUTHOR : SEE HEAN QUEK
 * DATE : MAY 12 1982
 * DESCRIPTION
 * CLEAR MAJORITY VOTE FILE AND TEMP*
 * ORARY ARRAYS , *
 *-------------------------------------------------------------)
CONST
  EMPTY = ' ';
  ZERO = '0';

VAR
  BLANK : PACKED ARRAY[1..9] OF CHAR;
  BLANK2 : PACKED ARRAY[1..9] OF CHAR;
BEGIN
  FOR I := 1 TO 9 DO
  BEGIN
  BLANK[I] := EMPTY;  /* ASSIGN BLANKS */
  BLANK2[I] := ZERO;
  END;
  FOR I := 1 TO MAXPRMETERS DO
  BEGIN
  FOR J := 1 TO 9 DO
  MJVPASS[I,J] := EMPTY;
  TEMPORARY[I] := BLANK;
  END;
  FOR I := 1 TO MAXMJV DO
  BEGIN
  DOP30FQ150[I] := BLANK2;
  DOP30FQ400[I] := BLANK2;
  MJVCODE[I] := ZERO;
  END;
  END; (* CLEARMJVFILE *)
PROCEDURE CONDPASSFILE;

(*końce*)

VAR
  DOPMESSAGE : PACKED ARRAY[1..32] OF CHAR;
  LINE, COUNT, MJLINE: INTEGER;
BEGIN (* CONDPASSFILE *)

  SHOWPROC('WMJV', SHOW);
  DOPMESSAGE[32] := CHR(13);
  FOR NO := 1 TO 31 DO
    DOPMESSAGE[NO] := ' ';

  FOR LINE := 1 TO MAXMJV DO
    BEGIN
      IF LINE = 1 THEN
        BEGIN
          FOR COUNT := 1 TO 19 DO
            DOPMESSAGE[COUNT] := LOCKONTIME[COUNT];
        END
      ELSE
        BEGIN
          DOPMESSAGE[10] := ' ';
          DOPMESSAGE[20] := ' ';
          DOPMESSAGE[30] := ' ';
          MJLINE := LINE - 1;
          IF MJLINE <= MAXPRMETERS THEN
            BEGIN
              FOR COUNT := 1 TO 9 DO
                PACK9[COUNT] := MJVPASS[MJLINE, COUNT];
            END
          ELSE
BEGIN
    IF LINE = MAXMJV THEN
      PACK9 := OPTIONCODE
    ELSE IF (LINE = MAXMJV-1) THEN
      PACK9 := RCVCODE
    ELSE
      PACK9 := '0000000000';
    END;

(* CASE WHEN PASS PARAGRAPHS ARE LESS THAN MAXIMUM *)
    IF PACK9[1] = '' THEN
      PACK9 := '0000000000';

(* TRANSFER OF DOPPLER COUNTS SCALED BY 100 *)
    PACK9 := DOP30FQ150[MJLINE];
    FOR NO := 1 TO 9 DO
      DOPMESSAGE[NO] := PACK9[NO];
    END;

    PACK9 := DOP30FQ400[MJLINE];
    FOR NO := 11 TO 19 DO
      BEGIN
        COUNT := NO - 10;
        DOPMESSAGE[NO] := PACK9[COUNT];
      END;

(* TRANSFER OF CODED ARRAY *)
    DOPMESSAGE[31] := MJVCODE[MJLINE];

(* MAJORITY VOTED BROADCAST EMPHEMERIS TRANSFER *)
    FOR NO := 21 TO 29 DO
      BEGIN
        COUNT := NO - 20;
        DOPMESSAGE[NO] := PACK9[COUNT]
      END;

    END;

    MJVFILE := DOPMESSAGE;
    PUT(MJVFILE)

END;

SHOWPROC('WMJV',ERASE);
SHOWMSG(1,CONCAT('MJV FILE= ',MJNAME));
SHOWMSG(2,'SUCCESSFULLY WRITTEN ON DISK');

END; (* CONDPASSFILE *)

PROCEDURE LONGDOPPLERS(LNCT,PARAG:INTEGER;XLINE:DATALINE);

(**************************************************************************
* AUTHOR ; SEE HEAN QUEK   *
* DATE ; AUGUST 28 1982   *
* DESCRIPTION ; *
* ROUTINE ACCUMULATES THE *
* 30 SECOND DOPPLERS AND STORES THEM *
* IN TWO ARRAYS : DOP30FQ150 FOR THE *
* 150MHZ DOPPLER COUNTS AND *
* DOP30FQ400 FOR THE 400MHZ COUNTS *
**************************************************************************
VAR 
LINE : INTEGER;
BEGIN (* LONGDOPPLERS *)

  IF NOT (LNCT IN [1,8,14,21]) THEN
    EXIT(LONGDOPPLERS);
  LINE := (PARAG-1)*4 + LNCT DIV 6;
  IF LINE = 0 THEN EXIT(LONGDOPPLERS);

  FOR NO := 1 TO 9 DO
    PACK9[NO] := XLINE[NO];
    DOP30FQ150[LIN]E := PACK9;

  FOR NO := 11 TO 19 DO
    BEGIN
      K := NO - 10;
      PACK9[K] := XLINE[NO];
    END;
    DOP30FQ400[LIN]E := PACK9;

END; (* LONGDOPPLERS *)

PROCEDURE ZERODOPPLERS;
**************************************************************************
* AUTHOR ; SEE HEAN QUEK   *
* DATE ; 5TH SEPTEMBER 1982   *
* DESCRIPTION ; *
* ZERO 30 SECOND COUNTS *
* THAT EXCEED THE REFMAX DIFFERENCE. *
**************************************************************************
VAR 
CDOP400,CDOP150,LDOP400,LDOP150 : INTEGER[9];
DOP400,DOP150,DIF0 : INTEGER[9];
PREVDOP : BOOLEAN;
BEGIN 
SHOWPROC('ZDOP',SHOW);
PREVDOP := FALSE;
FOR NO := 1 TO MAXMJV DO
BEGIN
IF (DOP30FQ150[NO] = '0000000000') OR
    (DOP30FQ400[NO] = '0000000000') THEN
    BEGIN
        DOP30FQ150[NO] := '0000000000';
        DOP30FQ400[NO] := '0000000000';
        LDOP150 := 0;
        LDOP400 := 0;
        PREVDOP := FALSE
    END
ELSE
    BEGIN
        PACK9 := DOP30FQ150[NO];
        FOR L := 1 TO 9 DO
            BEGIN
                UNPACK9[L] := PACK9[L]
            END;
        SHOWPROC('CN01', SHOW);
        CHAR9TOINT(CDOP400, UNPACK9);
        SHOWPROC('CN01', ERASE);

        PACK9 := DOP30FQ400[NO];
        FOR L := 1 TO 9 DO
            BEGIN
                UNPACK9[L] := PACK9[L]
            END;
        SHOWPROC('CN02', SHOW);
        CHAR9TOINT(CDOP150, UNPACK9);
        SHOWPROC('CN02', ERASE);

        IF (NO = 1) THEN
            BEGIN
                DOP400 := CDOP400;
                DOP150 := CDOP150
            END
ELSE
    BEGIN
        DOP400 := CDOP400 - LDOP400;
        DOP150 := CDOP150 - LDOP150
    END;

    (* ENSURE POSITIVE DIFFERENCES *)
    DIFF := (DOP400 - DOP150) DIV 100;
    IF (DIFF < 0) THEN DIFF := -DIFF;

    (* REJECTION SECTION *)
    IF (DIFF > REFMAX) THEN
        BEGIN
            IF NOT (NO IN [1, 5, 9, 13, 17, 21, 25, 29, 33, 37]) THEN

BEGIN
  IF PREVDOP = FALSE THEN
    BEGIN
      L := NO - 1;
      DOP30FQ150[L] := '0000000000';
      DOP30FQ400[L] := '0000000000';
      END
    ELSE
      BEGIN
        DOP30FQ150[NO] := '0000000000';
        DOP30FQ400[NO] := '0000000000';
        CDOP150 := 0;
        CDOP400 := 0
        END;
      PREVDOP := FALSE
    END;
  IF NO IN [4,8,12,16,20,24,28,32,36] THEN
    BEGIN
      L := NO - 1;
      IF (DOP30FQ150[L] = '0000000000')
        AND (DOP30FQ400[L] = '0000000000') THEN
        BEGIN
          DOP30FQ150[NO] := '0000000000';
          DOP30FQ400[NO] := '0000000000';
          END;
      END
    ELSE
      PREVDOP := TRUE;
      END
    (* PREPARE FOR NEXT COUNT *)
    LDOP400 := CDOP400;
    LDOP150 := CDOP150;
    IF (NO IN [4,8,12,16,20,24,28,32,36]) THEN
      BEGIN
        LDOP400 := 0;
        LDOP150 := 0
        END;
    END; (* ELSE SECTION *)
  END; (* FOR SECTION *)

SHOWPROC('ZDOP',ERASE);

END;

PROCEDURE CHECKDOPPLERS;
  (***)
  * AUTHOR : SEE HEAN QUEK *
  * VERSION : 10 AUGUST 1982 *
* DESCRIPTION : *
* ROUTINES CHECKS IF *
* THE NUMBER OF RECORDED COUNTS *
* EXCEED THE PRESELECTED MINIMUM. *
**************************************************************************

VAR
NOCOUNT : STRING[2];

BEGIN (* CHECKDOPPLERS *)
  NO := 0;
  SHOWPROC('CDOP',SHOW);
  FOR I := 1 TO MAXMJV DO
    IF (DDP30FQ150[I] = '0000000000') AND (DDP30FQ400[I] = '0000000000') THEN
      NO := NO + 1;
  NO := MAXMJV - NO;
  STR(NO,NOCOUNT);
  IF (NO <= MINDOP30) THEN BEGIN
    IF FIALDOPRJ = 'Y' THEN BEGIN
      SHOWMSG(0,
        CONCAT('PASS DELETED[',NOCOUNT,'] - BAD COUNTS'));
      SHOWPROC('CHECKMJV',ERASE);
      EXIT(WRITEPASS)
    END
    ELSE BEGIN
      SHOWMSG(2,'WARNING - BAD DOPPLERS');
      I := MAXMJV-1;
      MJVCODE[I] := '9'
    END;
  END;
  SHOWPROC('CDOP',ERASE);
END; (* CHECKDOPPLERS *)

PROCEDURE CHECKMJVFILE;
**************************************************************************
* AUTHOR : SEE HEAN QUEK *
* DATE : 19 JULY 1982 *
* DESCRIPTION *
* PERFORMS A SERIES OF CHECKS*
* TO ASCERTAIN THE QUALITY OF THE *
* MAJORITY VOTED MESSAGE. *
**************************************************************************
VAR 
  VALUE : INTEGER[9];
  NUMCHR : MESSCHAR;

BEGIN (* CHECKMIVFILE *)

(* CHECKING FOR BLANKS AND IF DETECTED LINE ZEROED *)
FOR NO := 1 TO MAXPRMETERS DO
  BEGIN
    FOR K := 1 TO 9 DO
      IF MJVPASS[NO,K] = ' ' THEN
        FOR L := 1 TO 9 DO
          MJVPASS[NO,L] := '0';
    END;

(* CHECKING THE FIRST NUMBER OF ALL *)
(* 9 OF THE 9 FIXED PARAMETERS *)
(* IF FAILS TEST ASSIGN CODE 9 *)
FOR K := ENDEPHEMERAL + 2 TO ENDEPHEMERAL + 13 DO
  IF(MJVPASS[K,1] IN ['0'..'7']) THEN
    MJVCODE[K] := '9';

(* CHECKING FOR ZEROED ROWS IN MESSAGE. TESTING VALUES OF FIXED PARAMETERS *)
FOR NO := 1 TO MAXPRMETERS DO
  BEGIN
    FOR K := 1 TO 9 DO
      BEGIN
        NUMCHR[K] := MJVPASS[NO,K];
        PACK9[K] := NUMCHR[K];
      END;

      SHOWPROC('CNUM',SHOW);
      IF PACK9 = '000000000' THEN
        VALUE := 0
      ELSE
        CHAR9TOINT(VALUE,NUMCHR);
        SHOWPROC('CNUM',ERASE);
  END;

(* NO VALUE *)
  IF(VALUE = 0) THEN
    MJVCODE[NO] := '1'
(* TIME OF PERIGEE *)
  ELSE IF (NO = ENDEPHEMERAL + 1) THEN
    BEGIN
      IF (VALUE > 400000000) THEN
        VALUE := VALUE - 400000000;
VALUE := VALUE DIV 1000000;
IF (VALUE < 0) OR (VALUE > 1440) THEN
    MJVCODE[NO] := '9';
END
(* RATE OF CHANGE OF MEAN ANOMALY *)
ELSE IF (NO = ENDEPHEMERAL + 2) THEN
    BEGIN
        VALUE := (VALUE-800000000) DIV 100000;
        IF (VALUE < 3) OR (VALUE > 4) THEN
            MJVCODE[NO] := '9';
    END
(* ARGUMENT OF PERIGEE AT TP *)
ELSE IF (NO = ENDEPHEMERAL + 3) THEN
    BEGIN
        VALUE := (VALUE-800000000) DIV 100000;
        IF (VALUE < 0) OR (VALUE > 360) THEN
            MJVCODE[NO] := '9';
    END
END;

SHOWMSG(0,' ');

(* DOPPLER COUNT ZEROING AND CHECKING*)
IF ZEROUNDOP = 'Y' THEN ZERODOPPLERS;
CHECKDOPPLERS;

(* CHECKING FOR ZEROS IN FIXED *)
(* PARAMETERS: IF DETECTED SWITCHES *)
(* MJV CODE FROM 1 TO 9 *)
FOR NO := ENDEPHEMERAL + 1 TO ENDEPHEMERAL + 14 DO
    IF (MJVCODE[NO] = '1') THEN
        MJVCODE[NO] := '9';
(* MESSAGE TO SCREEN *)
FOR NO := 1 TO MAXPRMETERS DO
    BEGIN
        IF (MJVCODE[NO] = '9') THEN
            BEGIN
                SHOWMSG(1,'MJV MESSAGE FAILS CHECKS');
                I := MAXMJV-2;
                MJVCODE[I] := '9';

                IF (FAILMJVRJ = 'Y') THEN
                    BEGIN
                        SHOWMSG(0,'LAST PASS DELETED - BAD MJV MESSAGE');
                        SHOWPROC('CMJV',ERASE);
                        EXIT(WRITEPASS)
                    END;
            END;
    END;

EXIT(CHECKMJVFILE);
END;
END;

SHOWMSG(1,'MJV MESSAGE CHECKED -- OK');

END; (* CHECKMJVFILE *)
**********************************************************************
*                                                            *
*       FILE :  SNAPP2:SATBCK32.TEXT                            *
*                                                            *
**********************************************************************
PROCEDURE SHOWMODE(MODE:STRING);
(* DISPLAYS STATUS: 'ACTIVE', 'WAIT', 'DISKIO' *)
BEGIN (* SHOWMODE *)
  GOTOXY(XMODE,YMODE);
  WRITE(MODE:7)
END; (* SHOWMODE *)

PROCEDURE SHOWMSG(MSGNUM:INTEGER;MESSAGE:STRING);
(* USED TO DISPLAY MOST MESSAGES IN 3-LINE MESSAGE AREA *)
(* MESSAGES ARE CENTRED IN THE 33-CHAR DISPLAY AREAS. *)
VAR FILLER:INTEGER;
BEGIN (* SHOWMSG *)
  GOTOXY(XMSG,YMSG+MSGNUM);
  FILLER:=(33-LENGTH(MESSAGE)) DIV 2;
  IF FILLER<O THEN FILLER:=O;
  WRITE(MESSAGE:(33-FILLER),':FILLER);
END; (* SHOWMSG *)

PROCEDURE SHOWLINE(VALUE:DATALINE;FTN:FTNTYPE);
(* USED TO DISPLAY FORMATTED SATELLITE DATA LINES ON SCREEN *)
BEGIN (* SHOWLINE *)
  IF FTN=SHOW THEN
    BEGIN
      GOTOXY(XLINE,SCRLINE);
      WRITE(VALUE:29);
      IF SCRLINE=YLINEMAX THEN
        SCRLINE:=YLINEMIN
      ELSE
        BEGIN
          SCRLINE:=SCRLINE+1;
          GOTOXY(XLINE,SCRLINE);
          WRITE(' ':29)
        END;
      CLEARLINES:=FALSE
    END
  ELSE IF (FTN=CLEAR) AND (NOT CLEARLINES) THEN
    BEGIN
      FOR SCRLINE:=YLINEMAX DOWNTO YLINEMIN DO
        BEGIN
          GOTOXY(XLINE,SCRLINE);
          WRITE(' ':29)
        END;
      SCRLINE:=YLINEMIN;
      CLEARLINES:=TRUE
    END
  END; (* SHOWLINE *)
PROCEDURE SHOWWORD(VALUE:DATAWORD);
(* USED TO DISPLAY INCOMING DATA FROM RECEIVER AS-IS *)
BEGIN (* SHOWWORD *)
  GOTOXY(XWORD,SCRWORD);
  WRITE(VALUE:5);
  IF SCRWORD=YWORDMAX THEN
    SCRWORD:=YWORDMIN
  ELSE
    BEGIN
      SCRWORD:=SCRWORD+1;
      GOTOXY(XWORD,SCRWORD);
      WRITE(" •":5)
    END
  END; (* SHOWWORD *)
PROCEDURE SHOWPROC(NAME:STRING;FTN:FTNTYPE);
(* USED TO DISPLAY CURRENTLY EXECUTING PROCEDURES FOR DEBUGGING *)
(* IF THIS ROUTINE IS CALLED WITH FTN=SHOW, THEN THE PROCNAMES *)
(* IS DISPLAYED ON THE SCREEN, UNDERNEATH ALL PREVIOUS NAMES. *)
(* A SUBSEQUENT CALL WITH FTN=ERASE WILL CAUSE ALL PROCNAMES UP *)
(* TO THE NAME SPECIFIED TO BE DELETED FROM THE SCREEN. IN THIS *)
(* WAY, A SUBROUTINE CAN "EXIT" FROM ITS CALLER AND REMOVE BOTH *)
(* NAMES FROM THE SCREEN AT ONCE. *)
VAR FOUND:boolean;
BEGIN (* SHOWPROC *)
  IF FTN=SHOW THEN
    BEGIN
      GOTOXY(XPROC,SCRPROC);
      PROCNAMES[SCRPROC]:=NAME;
      WRITE(NAME:4);
      SCRPROC:=SCRPROC+1
    END
  ELSE IF FTN=ERASE THEN
    BEGIN
      FOUND:=FALSE;
      REPEAT
        GOTOXY(XPROC,SCRPROC);
        WRITE(" •":4);
        FOUND:=PROCNAMES[SCRPROC]=NAME;
        PROCNAME[SCRPROC]:="•";
        SCRPROC:=SCRPROC-1
      UNTIL FOUND;
      SCRPROC:=SCRPROC+1;
    END;
  END; (* SHOWPROC *)
PROCEDURE FORMATSCREEN;
(* ROUTINE TO READ SCREEN FILE AND INITIALIZE SCREEN DISPLAY *)
VAR SCRNFILE:FILE;
  BLOCKCNT:INTEGER;
  BUFFER: PACKED ARRAY[0..511] OF CHAR;
BEGIN (* FORMATSCREEN *)
  PAGE(OUTPUT);
  (**I--**) RESET(SCRNFILE,'#4:RCV.SCREEN.TEXT'); (**I++*)
  IF IORESULT<>0 THEN
    BEGIN
      GOTOXY(0,7);
      WRITELN('UNABLE TO OPEN #4:RCV.SCREEN.TEXT');
      NOTE(35,50);
      EXIT(PROGRAM)
    END;
  BLOCKCNT:=BLOCKREAD(SCRNFILE,BUFFER,1,1);
  WHILE (IORESULT=0) AND (NOT EOF(SCRNFILE)) DO
    BEGIN
      UNITWRITE(1,BUFFER,512,0,2);
      BLOCKCNT:=BLOCKREAD(SCRNFILE,BUFFER,1)
    END;
  UNITWRITE(1,BUFFER,358,0,2);
  CLOSE(SCRNFILE);
  SCRPC :=YPROCMMIN;
  SCRWORD :=YWORDMIN;
  SCRLINE :=YLINEMIN;
  CLEARLINES:=FALSE;
END; (* FORMATSCREEN *)

PROCEDURE READANSWER(VAR Ans:CHAR);
("**********
* VERSION : AUGUST 1982  *
* AUTHOR : SEE HEAN QUEK *
* DESCRIPTION : *
* CHECKS YES AND NO *
* REPLY TO QUESTIONS. *
**********")
BEGIN (* READANSWER *)
  REPEAT
    READLN(ANS);
    IF NOT (ANS IN ['Y','N','D']) THEN
      BEGIN
        WRITELN;
        WRITELN('ILLEGAL REPLY. RE-ENTER ');
        WRITE('==> ')
      END;
    UNTIL (ANS IN ['Y','N','D'])
END; (* READANSWER *)
PROCEDURE READNUMBER(VAR NUMBER: INTEGER);
(* **********************************************************************
 * VERSION : 1 FEB 1983          *
 * AUTHOR : SEE HEAN, QUEK      *
 * DESCRIPTION :               *
 * CHECK NUMERICAL INPUT. ONLY  *
 * CHECKS SIZE DIGITS OF INPUT. *
 ***********************************************************************)

VAR
  TEXT : STRING;
  SIZE, I : INTEGER;
  ERROR : BOOLEAN;

FUNCTION IEXP(I:INTEGER) : INTEGER;
(*SUBPROCEDURE-EXPONENT FUNCTION*)
VAR
  TEN, J : INTEGER;
BEGIN (* IEXP *)
  TEN := 1;
  IF(I=0) THEN
    BEGIN
      IEXP := 1;
      EXIT(IEXP)
    END
  ELSE
    FOR J := 1 TO I DO
      BEGIN
        TEN := TEN*10;
        IF(TEN>10000) THEN
          BEGIN
            WRITELN('IEXP TO LARGE ** FATAL **');
            EXIT(PROGRAM);
          END;
          IEXP := TEN;
        END; (* IEXP *)
      END;
    END; (* IEXP *)
BEGIN (* READNUMBER *)
  REPEAT
    ERROR := FALSE;
    READLN(TEXT);
    SIZE := LENGTH(TEXT);
    IF (SIZE=0) THEN ERROR := TRUE;
    IF NOT ERROR THEN
      FOR I := 1 TO SIZE DO
        IF(NOT (TEXT[I] IN ['0'..'9','D'])) THEN
          BEGIN
            ERROR := TRUE;WRITELN;
            WRITELN('ILLEGAL INPUT. RE-ENTER !');
          END;
    END; (* READNUMBER *)
WRITE('==>')
END;
UNTIL NOT ERROR;
IF(TEXT[1] = 'D') THEN EXIT(READNUMBER);
IF(SIZE>37) THEN
BEGIN
  WRITELN('NUMBER EXCEED 37 DIGITS ** FATAL **');
  EXIT(PROGRAM);
END;
NUMBER := 0;
FOR I := 1 TO SIZE DO
BEGIN
  L := SIZE+1-I;
  NUMBER := NUMBER + (ORD(TEXT[L]) - 48)*IEXP(I-1);
END;
END; (* READNUMBER *)

PROCEDURE WELCOME;
(**************************************************************************
 * VERSION : 10 AUGUST 1982     *
 * AUTHOR : SEE HEAN, QUEK     *
 * DESCRIPTION :               *
 * WELCOMING MESSAGE           *
 * TO THE SATNAV PROGRAM.      *
**************************************************************************)

VAR
  ANS : CHAR;
BEGIN (* WELCOME *)
  PAGE(OUTPUT);
  GOTOXY(0,2);
  WRITELN(' ----------------------------------- ');
  WRITELN('II -=CANTJ=-
***********
* SATNAV *
***********
II -=CANTJ=-
***********
PROGRAM
***********
II -=CANTJ=-
***********
CMA-722!

  MARK LORD < 1981 >
  SEE HEAN QUEK <1983)
  MAY 1983 (3.0)

  AUTHORS

  (ORIGINAL) : MARK LORD (1981)
  (REVISION) : SEE HEAN QUEK (1983)
  VERSION : MAY 1983 (3.0)

  ----------------------------------- ');

  GOTOXY(0,22);
WRITELN('DO YOU WISH TO ALTER PROGRAMMED?');
WRITE ('DEFAULTS ? == > '); READANSWER(ANS);
IF (ANS IN ['N','D']) THEN SAVEOPTION;
PAGE(OUTPUT);
WRITELN('TO KEEP DEFAULT VALUES, TYPE <D>'); OPTION;
END; (* WELCOME *)

PROCEDURE ADDINFO;
(**************************
* VERSION : JANUARY 12TH 1983 *
* AUTHOR : SEE HEAN QUEK *
* DESCRIPTION : READS IN A NINE DIGIT *
* NUMBER IDENTIFYING RECEIVER . *
**************************)
VAR
RCVSTR : STRING;
IONERR : BOOLEAN;
DIGIT : INTEGER;
ANS : CHAR;
BEGIN (* ADDINFO *)
PAGE(OUTPUT);
GOTOXY(0,5);
WRITELN('INPUT 9-DIGIT CODE TO IDENTIFY');
WRITELN('USER/STN/REC E.G. 123456789');
REPEAT
GOTOXY(0,8);WRITELN(' :30');GOTOXY(0,8);
WRITE(' == > ');READLN(RCVSTR);
GOTOXY(0,9);WRITELN(' :30');GOTOXY(0,9);
IONERR := TRUE;
IF (LENGTH(RCVSTR)<>9) THEN
BEGIN
WRITELN('9-DIGITS EXPECTED. RE-ENTER');
IONERR := FALSE;
END;
IF (IONERR) THEN
BEGIN
FOR I := 1 TO 9 DO
IF NOT (RCVSTR[I] IN ['0'..'9']) THEN
IONERR := FALSE;
IF NOT IONERR THEN
WRITELN('NUMERICAL INPUT PLEASE !');
END;
UNTIL IONERR;

FOR I := 9 DOWNTO 1 DO
BEGIN
  DIGIT := ORD(RCVSTR[I]);
  RCVCODE[I] := CHR(DIGIT);
END;

END; (*ADDINFO*)
(* ROUTINES RELATED TO THE READING *)
(* AND PROCESSING OF THE DOPPLER DATA *)

PROCEDURE READPARA; FORWARD;
PROCEDURE READPASS; FORWARD;

PROCEDURE READWORD;
(* PROCEDURE TO GET NEXT 4-DIGIT INPUT WORD FROM RECEIVER *)
(* USING ASSEMBLER INPUT QUEUE HANDLER, "GETWORD". *)
VAR ENDOFPARA,ENDOPASS:BOOLEAN;

PROCEDURE SCANKB;
(* SUB-PROCEDURE TO SCAN KEYBOARD FOR USER INPUT. TO ISSUE *)
(* A COMMAND, USER MUST FIRST HIT <ESC> KEY, AND THEN THE *)
(* APPROPRIATE KEY FOR HIS COMMAND. *)
CONST QUIT = 'Q'; (* EXIT PGM AFTER CURRENT PASS *)
STAY = 'S'; (* CANCELS EFFECT OF ISSUED "QUIT" *)
UNLOCK = 'U'; (* ISSUE UNLOCK-PASS CMD TO REC *)
KILL = 'K'; (* TERMINATE PROGRAM IMMEDIATELY! *)
VAR KBCHR:CHAR;
BEGIN
SHOWPROC('SCKB',SHOW);
READ(KEYBOARD,KBCHR);
IF NOT ESCPRESSED THEN
  ESCPRESSED:= KBCHR=CHR(27)
ELSE
  ESCPRESSED:=FALSE;
  UNITCLEAR(2);
  IF KBCHR IN [QUIT,STAY,UNLOCK,KILL] THEN CASE KBCHR OF
  QUIT:QUITREQUESTED:=TRUE;
  STAY:QUITREQUESTED:=FALSE;
  UNLOCK:BEG
  UNLOCKPASS;
  ENDOFPASS:=TRUE
  END;
  KILL:BEG
  PARACNT:=0;
  QUITREQUESTED:=TRUE;
  ENDOFPASS:=TRUE
  END;
END (* CASE *)
SHOWPROC('SCKB',ERASE);
END; (* SCANKB *)

BEGIN (* READWORD *)
SHOWPROC('RDWD',SHOW);

BEGIN (* READWORD *)
SHOWPROC('RDWD',SHOW);
ENDOFPARA:=FALSE;
ENDOFPASS:=FALSE;

IF KEYPRESS THEN SCANKB;
GETWORD(INPUTWORD);

IF LENGTH(INPUTWORD)=0 THEN BEGIN
  SHOWMODE('WAIT');
  REPEAT
    IF KEYPRESS THEN SCANKB;
    GETWORD(INPUTWORD)
  UNTIL (LENGTH(INPUTWORD)<>0) OR ENDOFPARA OR ENDOFPASS;
  SHOWMODE('ACTIVE')
END;

IF LENGTH(INPUTWORD)=4 THEN CASE INPUTWORD[1] OF
  '0': IF INPUTWORD<>"0000" THEN
    BEGIN
      ENDOFPARA:=TRUE;
      SHOWWORD('T2MIN')
    END
  ELSE BEGIN
    ENDOFPASS:=TRUE;
    SHOWWORD('R2MIN')
  END;

  '1','2','3','4':BEGIN
    ENDOFPARA:=TRUE;
    SHOWWORD(INPUTWORD);
    WRITE(CHR(7))
  END;

  'B':BEGIN
    ENDOFPARA:=TRUE;
    SHOWWORD('S2MIN')
  END;

  'C':BEGIN
    ENDOFPASS:=TRUE;
    SHOWWORD('ENDPS')
  END;

  '5','6','7','9','A','B','D','E','F': SHOWWORD(INPUTWORD);
END; (* CASE *)

IF ENDOFPASS THEN BEGIN
  SHOWPROC('RDPS',ERASE);
  EXIT(READPASS)
END
ELSE IF ENDOFPARA THEN
BEGIN
    SHOWPROC('RDPA', ERASE);
    EXIT(READPARA)
END;

SHOWPROC('RDWD', ERASE);
END; (* READWORD *)

PROCEDURE READLINE;
(* ROUTINE TO FORMAT NEXT LINE OF RECEIVER INPUT (9 WORDS) *)
(* INTO VARIABLE "INPUTLINE". SEQUENCE CODES OF THE INPUT *)
(* WORDS ARE CHECKED FOR PROPER SEQUENCE, AND THE DOPPLER *)
(* COUNTS ARE TESTED TO ENSURE THAT THEY CONTAIN ONLY BCD *)
(* DIGITS. THIS TESTING IS NOT DESIRED FOR THE SATELLITE *)
(* MESSAGE (LAST 3 WORDS). *)
VAR WORDNUM, DIGIT: INTEGER;
    DATAERROR : BOOLEAN;
BEGIN (* READLINE *)
    SHOWPROC('RDLN', SHOW);
    DATAERROR := FALSE;
    DIGIT := 1;
    FOR WORDNUM := 1 TO 9 DO
BEGIN
READWORD;
        DATAERROR := TRUE
ELSE
    CASE WORDNUM OF
      1, 2, 4, 5:
        IF NOT ((INPUTWORD[2] IN ['0'..'9']) AND
                (INPUTWORD[3] IN ['0'..'9']) AND
                (INPUTWORD[4] IN ['0'..'9'])) THEN
            DATAERROR := TRUE
        ELSE
            BEGIN
                INPUTLINE[DIGIT] := INPUTWORD[2];
                INPUTLINE[DIGIT+1] := INPUTWORD[3];
                INPUTLINE[DIGIT+2] := INPUTWORD[4];
                DIGIT := DIGIT + 3
            END;
      3, 6:
        IF NOT (INPUTWORD[4] IN ['0'..'9']) THEN
            DATAERROR := TRUE
        ELSE
            BEGIN
                INPUTLINE[DIGIT] := INPUTWORD[4];
                INPUTLINE[DIGIT+1] := '0';
                INPUTLINE[DIGIT+2] := '0';
            END;
      ELSE:
        IF DATAERROR THEN
            BEGIN
                INPUTLINE[DIGIT] := '0';
                INPUTLINE[DIGIT+1] := '0';
                INPUTLINE[DIGIT+2] := '0';
            END;
    END;
END;

DIGIT:=DIGIT+4
END;
7,8,9:
BEGIN
  INPUTLINE[DIGIT] :=INPUTWORD[2];
  INPUTLINE[DIGIT+1]:=INPUTWORD[3];
  INPUTLINE[DIGIT+2]:=INPUTWORD[4];
  DIGIT:=DIGIT+3
END;
END; (* CASE *)
IF DATAERROR THEN
BEGIN
  WRITE(CHR(7));
  SHOWPROC('RDPA',ERASE);
  EXIT(READPARA)
END;
SHOWPROC('RDLN',ERASE);
END; (* READLINE *)

PROCEDURE READPARA;
(* ROUTINE TO SET TIMESTAMP FOR NEXT PARAGRAPH OF INPUT *)
(* AND THEN TO CALL READLINE ENOUGH TIMES TO OBTAIN A *)
(* COMPLETE PARAGRAPH. IF ANY ERRORS OCCUR IN READLINE,*)
(* OR IF READWORD ENCOUNTERS 2-MINUTE MARKS, THEN THIS *)
(* ROUTINE WILL NEVER COMPLETE AND THUS THE PARACNT *)
(* POINTER WILL NOT BE ADVANCED, THUS CAUSING THE INPUT *)
(* PARAGRAPH TO BE IGNORED. NOTE THAT 2-MINUTE MARKS *)
(* BETWEEN PARAGRAPHS WILL CAUSE THE TIMESTAMP TO BE *)
(* UPDATED, BUT WILL HAVE NO ILL EFFECTS OTHERWISE. *)
VAR PARANUM,LINECNT:INTEGER;
  CURRENTTIME:TIMESTAMP;
  DISPLAYSTRING:STRING;
BEGIN (* READPARA *)
  SHOWPROC('RDPA',SHOW);
  PARANUM:=PARACNT+1;
  GOTOXY(XPNUM,YPNUM); WRITE(PARANUM:2);
  READTIME(CURRENTTIME);
  CURRENTTIME[SIZEOF(CURRENTTIME)]:=CHR(13);
  IF PARANUM = 1 THEN
    LOCKONTIME := CURRENTTIME;
    DISPLAYSTRING:=" "; (* 19 SPACES *)
    MOVELEFT(CURRENTTIME[1],DISPLAYSTRING[1],19);
    SHOWMSG(2,CONCAT('TIMESTAMP = ',DISPLAYSTRING));
SHOWLINE(INPUTLINE,CLEAR);
WITH PASSPARA[PARANUM] DO
BEGIN
PASSTIME:=CURRENTTIME;
FOR LINECNT:=1 TO MAXLINE DO
BEGIN
GOTOXY(XLNUM, YLNUM); WRITE(LINECNT:2);
READLINE;
PASSLINE[LINcNT]:=INPUTLINE;
SHOWLINE(INPUTLINE, SHOW);
SHOWPROC('VOTE', SHOW);
MJVLINE(LINECNT, PARANUM, INPUTLINE);
SHOWPROC('VOTE', ERASE);
LONGDOPPLERS(LINECNT, PARANUM, INPUTLINE);
END
PARACNT:=PARANUM;
SHOWPROC('RDPA', ERASE);
END; (* READPARA *)

PROCEDURE READPASS;
(* THIS PROCEDURE COLLECTS PASS DATA UNTIL EITHER THE *)
(* END OF PASS IS REACHED (READWORD WILL CAUSE EXIT), *)
(* OR UNTIL IT HAS COLLECTED THE MAXIMUM ALLOWABLE *)
(* NUMBER OF DATA PARAGRAPHS - WHICHEVER OCCURS FIRST.*)
BEGIN (* READPASS *)
SHOWPROC('RDPS', SHOW);
PARACNT:=O;
(* CLEAR MAJORITY VOTING AND TEMPORARY ARRAYS. *)
CLEARMJVFILE;
SATLOCK := FALSE;
REPEAT
READPARA
UNTIL (PARACNT=MAXPARA);
UNLOCKPASS;
SHOWPROC('RDPS', ERASE);
END; (* READPASS *)
FILE: SNAPP2: SATNAV3. TEXT
(*$S+*) (* TURN LEVEL-1 COMPILER SWAPPING ON FOR LARGE PROGRAM *)

***************************************************************
* VERSION : 3.0 *
* ORIGINAL AUTHOR : MARK LORD *
* MODIFICATION BY : SEE HEAN QUEK *
* REVISED MANUAL : MAY 1983 *
* *
* ## AS THE PROGRAM HAS EXCEED THE MAX *
* TEXT FILE SIZE, CERTAIN PROCEDURES *
* ARE NOW KEPT IN A DIFFERENT FILE. *
* CONSULT THE DOCUMENTATION TO CLARIFY *
* ANY MAJOR DETAILS. *
***************************************************************

PROGRAM SATNAV3;
USES APPLESTUFF, PEEKPOKE;

CONST MAXPARA = 8; (* THIS LINE SPECIFIES # OF PARAGRAPHS/PASS *)
MAXLINE = 25; (* THIS LINE SPECIFIES # OF LINES/PARAGRAPH *)
REFMAX = 1500; (* MAXIMUM DIFFERENCE BETWEEN 2 FREQ COUNTS *)

(* NOTE: MAXMJV MUST BE > MAXPRAMETERS + 1 - ESSENTIAL *)
ENDEPHERMAL = 14; (* LENGTH OF VARIABLE PARAMETERS *)
MAXPRMETERS = 28; (* MAXIMUM LENGTH OF ALL PARAMETERS *)
MAXMJV = 32; (* MAXIMUM LENGTH OF COMPACT DOPPLERS *)

(* THE FOLLOWING CONSTANTS ARE USED FOR *)
(* POSITIONING ITEMS ON THE SCREEN, AND *)
(* MOST CAN BE SAFELY ALTERED TO MODIFY *)
(* THE SCREEN FORMAT. *)
XMODE=7; YMODE=0;
XPNUM=11; YPNUM=1;
XLNUM=14; YLNUM=1;
XMSG='1'; YMSG='3';
XLINE=5; YLINEMIN=10; YLINEMAX=23;
XPROC=0; YPROCMIN=10; YPROCMAX=23;
XWORD=35; YWORDMIN=3; YWORDMAX=23;

TYPE FTNTYPE = (SHOW, ERASE, CLEAR);
DATAWORD = STRING[5];
DATALINE = PACKED ARRAY[1..30] OF CHAR;
TIMESTAMP = PACKED ARRAY[1..20] OF CHAR;
PARARECORD = RECORD
    PASSTIME: TIMESTAMP;
    PASSLINE: ARRAY[1..MAXLINE] OF DATALINE
END;
LONGLINE = PACKED ARRAY[1..30] OF CHAR;
MESSAGELINE = PACKED ARRAY[1..9] OF CHAR;
PACKLINE = PACKED ARRAY[1..32] OF CHAR;
DOPLINE    = PACKED ARRAY[1..9] OF CHAR;
MESSCHAR   = ARRAY[1..9] OF CHAR;
MESSVALUE  = INTEGER[9];

VAR PROCNAMES:ARRAY[YPROCMIN..YPROCMAX] OF STRING[8];
PASSPARA  :ARRAY[1..MAXPARA] OF PARARECORD;
SEQCODES  :PACKED ARRAY[1..9] OF CHAR;
INPUTWORD :DATAWORD;
INPUTLINE :DATALINE;
PARACOMPLETED :BOOLEAN;
MEMUNUSED :STRING[5];
SCRPROC,SCRLINE,SCRWORD,PARACNT :INTEGER;
CLEARLINES,QUITREQUESTED,ESCPRESSED :BOOLEAN;
PARAMFILE :TEXT;
PASSFILE :FILE OF PARARECORD;

PFNUMBER,PFINCREMENT :INTEGER;
PFDEVICE :STRING[7];
PFROOTNAME,PFEXT,MJROOTNAME :STRING[14];
PFNNAME,MJNAME :STRING[26];

MJVPASS : ARRAY[1..MAXPARAMETERS] OF MESSCHAR;
TEMPORARY : ARRAY[1..MAXPRMETERS] OF MESSAGELINE;
DOP30FQ150,DOP30FQ400: ARRAY[1..MAXMJV] OF DOPLINE;

LOCKONTIME: PACKED ARRAY[1..20] OF CHAR;
MJVFILE : FILE OF PACKLINE;
MJVCODE : PACKED ARRAY[1..MAXMJV] OF CHAR;
OPTIONCODE,RCVCODE: MESSAGELINE;

MINPARA,MINDOP30 : INTEGER;
FAILMJVRJ,FAILDOPRJ : CHAR;
ZEROUNDOP : CHAR;
MJVONLY,SATLOCK : BOOLEAN;
ISAT : INTEGER;

(* GLOBAL VARIABLES *)
NO,I,J,K,L : INTEGER;
PACK9 : MESSAGELINE;
UNPACK9 : MESSCHAR;

PROCEDURE WRITEPASS; FORWARD;
PROCEDURE OPTION; FORWARD;
PROCEDURE SAVEOPTION; FORWARD;

(*$I #5:SATBCK32.TEXT *)

PROCEDURE SAVEOPTION;

(********************************************
* VERSION : SEPTEMBER 5TH 1982  *
* AUTHOR : SEE HEAN, QUEK   *
* DESCRIPTION :  *
  * WRITES OPTION SELECTED *
  * ON THE LAST LINE OF MJV PASS FILE. *
  ******************************************
VAR
  OPT : CHAR;

BEGIN
  (* MINPARA *)
  OPT := CHR(MINPARA + 48);
  OPTIONCODE[1] := OPT;
  (* ZERO DOPPLERS *)
  IF ZEROUNDOP = 'Y' THEN
    OPTIONCODE[2] := '1'
  ELSE
    OPTIONCODE[2] := '0';
  (* REJECT PASS ON DOPPLERS *)
  IF FAILDOPRJ = 'Y' THEN
    OPTIONCODE[3] := '1'
  ELSE
    OPTIONCODE[3] := '0';
  (* REJECT PASS ON MJV *)
  IF FAILMJVRJ = 'Y' THEN
    OPTIONCODE[4] := '1'
  ELSE
    OPTIONCODE[4] := '0';
  (* MINDOP30 *)
  OPT := CHR((MINDOP30 DIV 10) + 48);
  OPT := CHR((MINDOP30 - ((MINDOP30 DIV 10)*10)) + 48);
  (* NOT USED *)
  OPTIONCODE[7] := '0';
  (* VERSION CODE-1ST DECIMAL*)
  OPTIONCODE[8] := '3';
  (* VERSION CODE-2ND DECIMAL*)
  OPTIONCODE[9] := '0';

  EXIT(WELCOME)
END;

PROCEDURE OPTION;

 ******************************************
* VERSION : SEPTEMBER 5TH 1982  *
* AUTHOR : SEE HEAN, QUEK   *
* DESCRIPTION :  *
  * ALLOWS A CHANGE OF *
  * PROGRAMMED DEFAULTS.  *
VAR
  ANS, TEXT : CHAR;
  MINDOP : INTEGER;

PROCEDURE OPTCONT;
BEGIN
  WRITELN;
  WRITELN('DO YOU WANT A MINIMUM NUMBER OF 30-SEC');
  WRITELN('COUNTS FOR EACH PASS TO BE ENFORCED?');
  WRITE('DEFAULT = ', FAILDOPRJ, ' CHANGE TO?');
  READANSWER(ANS);
  IF (ANS = 'N') THEN FAILDOPRJ := 'N';
  IF (ANS = 'Y') THEN
    BEGIN
      WRITELN;
      WRITELN('INPUT MINIMUM 30-SEC TWO FREQ COUNTS?');
      WRITE('DEFAULT = ', MINDOP30, ' CHANGE TO?');
      READNUMBER(MINDOP);
      IF (MINDOP >= MAXMJV) THEN
        WRITELN('ERROR - VALUE = OR > ',
        MAXMJV, '. DEFAULT USED')
      ELSE
        MINDOP30 := MINDOP;
    END;
  END;

  WRITELN;
  WRITELN('SELECT DATA TO BE SAVED ON DISK.');
  WRITELN(' 1. MJV DATA ONLY');
  WRITELN(' 2. MJV AND RAW DATA');
  WRITELN;
  WRITELN('NOTE: IF INPUT <> 1 OR 2, DEFAULT USED');
  IF MJVONLY THEN
    NO := 1
  ELSE
    NO := 2;
  WRITE('DEFAULT = ', NO, ' CHANGE TO ?');
  READNUMBER(NO);
  IF NO = 1 THEN
    MJVONLY := TRUE
  ELSE
    MJVONLY := FALSE;

  WRITELN;
  WRITE('DONE');
  FOR I := 1 TO 1000 DO
    MINDOP := MINDOP + 1;
  (* KEEP OPTIONS *)
  SAVEOPTION;
END;
BEGIN (* OPTION *)

WRITELN;
WRITELN('MINIMUM NUMBER OF TWO MINUTE PARAGRAPHS');
WRITELN('TO BE OBTAINED BEFORE WRITING TO');
WRITELN('DISK DRIVES ? ');
WRITE('<DEFAULT = ',MINPARA,'> NEW VALUE ?');
READNUMBER(MINPARA);

WRITELN;
WRITELN('IN THE EVENT THAT THE MAJORITY VOTED');
WRITELN('MESSAGE FAILS THE BUILT-IN TESTS,');
WRITELN('DO YOU STILL WISH TO SAVE THE ');
WRITELN('PASS ON THE DISKETTE ?');
IF (FAILMJVRJ='Y') THEN
  TEXT:= 'N'
ELSE
  TEXT:= 'Y';
WRITE('<DEFAULT = ',TEXT,'> CHANGE TO ?');
READANSWER(ANS);
IF(ANS = 'Y') THEN FAILMJVRJ := 'N'
ELSE FAILMJVRJ := 'Y';

WRITELN;WRITELN;
WRITELN('IF THE DIFFERENCE BETWEEN THE DOPPLER');
WRITELN('COUNTS AT 400MHZ AND 150MHZ EXCEEDS ',REFMAX);
WRITELN('COUNTS, DO YOU WANT THEM TO BE ZEREOED ?');
WRITE('<DEFAULT = ',ZEROUNDOP,'> CHANGE TO ?');
READANSWER(ANS);
IF(ANS = 'N') THEN ZEROUNDOP := 'N';

OPTCONT;
END; (* OPTION *)

(**$I #S:SATBCK31.TEXT *)

PROCEDURE INITPIA; EXTERNAL;
PROCEDURE GETWORD (VAR STRING4); EXTERNAL;
PROCEDURE RESETIRQ; EXTERNAL;
PROCEDURE READTIME(VAR PKCHAR19); EXTERNAL;

PROCEDURE UNLOCKPASS;
(* ROUTINE TO SET ANNUNCIATOR OUTPUTS AND STROBE TO COMMAND *)
(* RECEIVER TO UNLOCK FROM THE CURRENT PASS. THESE OUTPUTS *)
(* ARE PART OF THE APPLE GAME I/O CONTROLLERS. *)
VAR STROBE:INTEGER;
BEGIN (* UNLOCKPASS *)
  TTLOUT(0,TRUE);
  TTLOUT(1,FALSE);
TTLOUT(2,FALSE);  
TTLOUT(3,TRUE);  
STROBE:=PEEK(-16320);  
END; (* UNLOCKPASS *)

(*$I #5:SATBCK33.TEXT *)

PROCEDURE OPENPASSFILE;
(* THIS PROCEDURE ATTEMPTS TO OPEN A NEW PASS FILE FOR *)
(* SAVING CURRENT PASS DATA IN. FILE SIZE IS *)
(* COMPUTED, AND RCV.PARAM IS USED TO MAKE A NEW FILE *)
(* NAME UP. ATTEMPTS ARE THEN MADE TO PRE-EXTEND THIS *)
(* FILE TO ITS FULL SIZE ON AN OUTPUT DISK, GIVING LAST*)
(* PREFERENCE TO THE (USUALLY) BOOT DISKETTE IN DRIVE *)
(* #4. IF ALL ATTEMPTS FAIL, THE USER IS PROMPTED BY A*)
(* HIGH-PITCHED BEEP-BEEP NOISE TO SPECIFY A FURTHER *)
(* COURSE OF ACTION FOR THE PROGRAM: EITHER TERMINATE, *)
(* OR TRY AGAIN TO FIND SPACE (IE. IF THE USER FIRST *)
(* INSERTS A NEW DISKETTE). *)
(* MODIFIED - S.H. QUEK *)
(* OPENS BOTH PASS AND MAJORITY VOTED DISK FILES. *)
VAR  
  PFBLOCKCNT,PREFERENCE,DUMMY:INTEGER;  
  IERMJ,IERPF,MJBLOCKCNT :INTEGER;  
  REPLY :CHAR;  
  PFSIZE,PFDIGITS,MJSIZE :STRING[5];  
  PFPARTIALNAME,MJPARTIALNAME :STRING[19];

PROCEDURE CHECKSPACE;
BEGIN  
  PFNAME:=CONCAT(PFDEVICE,PFPARTIALNAME);  
  MJNAME:=CONCAT(PFDEVICE,MJPARTIALNAME);  
  IF MJVONLY THEN
      SHOWMSG(0,CONCAT('NEW FILE=',MJNAME))
  ELSE
      SHOWMSG(0,CONCAT('NEW FILE=',PFNAME));
  (*$I--*)
  IERP := 0;
  IF NOT MJVONLY THEN
      BEGIN
          REWRITE(PASSFILE,PFNAME);
          IERP := IORESULT;
      END;
  END;
  REWRITE(MJVFILE,MJNAME); (*$I++)
  IERMJ := IORESULT;
  IF (IERPF=0) AND (IERM=0) THEN
      BEGIN
          SHOWPROC('OFIL',ERASE);
EXIT(OPENPASSFILE)
END;

(* CASE WHEN ONLY ONE FILE IS SUCCESSFULLY OPENED *)
IF NOT MJVONLY THEN
  IF IERPF=0 THEN
    CLOSE(PASSFILE);
  IF IERMJ=0 THEN
    CLOSE(MJVFILE);
END; (*CHECKSPACE*)

BEGIN (* OPENPASSFILE *)
SHOWPROC('OFIL',SHOW);
STR(PNUMBER,PFDIGITS);

(* THE FOLLOWING LINES DETERMINE THE REQUIRED FILE SIZE *)
(* IN BLOCKS OF THE OUTPUT PASS FILE. "TEXT" FILES ARE*)
(* A SPECIAL CASE BECAUSE THEY REQUIRE A 2-BLOCK HEADER *)
(* RECORD (WRITTEN BY OPERATING SYSTEM) AND THEY MUST *)
(* BE WRITTEN IN EVEN INCREMENTS OF 2-BLOCKS. *)

IF PFEXT='.TEXT' THEN
  BEGIN
    PFBLOCKCNT:=2*(1+(PARACNT*SIZEOF(PARARECORD) DIV 1024))+2;
    MJBLOCKCNT:=2 + 1+(MAXMJV*SIZEOF(PACKLINE) DIV 512)
  END
ELSE
  BEGIN
    PFBLOCKCNT:=1+(PARACNT*SIZEOF(PARARECORD) DIV 512);
    MJBLOCKCNT:=1+(MAXMJV*SIZEOF(PACKLINE) DIV 512)
  END;

STR(PFBLOCKCNT,PFSIZE);
PFPARTIALNAME:=CONCAT(PFROOTNAME,PFDIGITS,PFEXT,['',PFSIZE,''])

STR(MJBLOCKCNT,MJSIZE);
MJPARTIALNAME:=CONCAT(MJROOTNAME,PFDIGITS,PFEXT,['',MJSIZE,''])

(* WE CAN USE THE SAME DISK AS LAST TIME ONLY IF IT WAS *)
(* NOT THE BOOT DRIVE (#4:). OTHERWISE, WE HAVE TO GO *)
(* SEARCHING FOR SPACE ELSEWHERE FIRST. *)

IF PDEVICE<>'#4:' THEN
  CHECKSPACE;

(* THE FOLLOWING LOGIC SEARCHES FOR AN OUTPUT DISK, IN *)
(* THE ORDER OF PRIORITY SPECIFIED WITHIN THE CASE BELOW*)

REPEAT
  SHOWMSG(2, '[SEARCHING FOR NEW OUTPUT DISK]')
  FOR PREFERENCE:=1 TO 6 DO
    BEGIN

CASE PREFERENCE OF
(* THESE ARE PASCAL DISK DRIVE UNITS *)
1: PFDEVICE:="#5:'"; (* FIRST CHOICE *)
2: PFDEVICE:="#11:'"; (* SECOND CHOICE *)
3: PFDEVICE:="#12:'"; (* THIRD CHOICE *)
4: PFDEVICE:="#9:'"; (* FOURTH CHOICE *)
5: PFDEVICE:="#10:'"; (* FIFTH CHOICE *)
6: PFDEVICE:="#4:'" (* LAST RESORT ONLY! *)
END; (* CASE *)
END;
CHECKSPACE;
SHOWMSG(1,"NO SPACE FOR OUTPUT FILE");
SHOWMSG(2,"<ESC>=KILL; <RETURN>=RETRY");
UNITCLEAR(2);
WHILE NOT KEYPRESS DO
BEGIN
NOTE(45,25); (* BEEP AND *)
FOR DUMMY:=1 TO 2000 DO (* DELAY! *)
END;
READ(KEYBOARD,REPLY);
SHOWMSG(1,"' ");
UNTIL REPLY=CHR(27); (* ESCAPE CHARACTER *)
QUITREQUESTED:=TRUE;
SHOWPROC("WPAS",ERASE);
EXIT(WRITEPASS)
END; (* OPENPASSFILE *)

PROCEDURE CLOSEPASSFILE;
(* THIS ROUTINE CLOSES THE CURRENT PASSFILE AND UPDATES *)
(* RCV.PARAM.TEXT TO REFLECT THE NEXT PASS NUMBER TO BE *)
(* USED IN CREATING PASS FILES. *)
(* MODIFIED - QUEK; CLOSE ALL FILES *)
BEGIN (* CLOSEPASSFILE *)
SHOWPROC("CFIL",SHOW);
IF NOT MJVONLY THEN
BEGIN
CLOSE(PASSFILE,LOCK);
SHOWMSG(1,"PASS FILE SUCCESSFULLY WRITTEN");
END;
CLOSE(MJVFILE,LOCK);
SHOWMSG(2,"[UPDATING RCV.PARAM.TEXT]");
PFNNUMBER:=PFNNUMBER+PFINCREMENT;
(*$I- *) (* PURGE EXISTING FILE *)
RESET(PARAMFILE,"#4:RCV.PARAM.TEXT");
IF IORESULT = 0 THEN
CLOSE(PARAMFILE,PURGE);
(*$I++*)
REWRITE(PARAMFILE,"#4:RCV.PARAM.TEXT[4]");
WRITELN(PARAMFILE,PFROOTNAME);
PROCEDURE WRITEPASS;
(* THIS ROUTINE HANDLES THE (VERY) FAST TRANSFER OF A GROUP *)
(* OF DATA PARAGRAPHS (IE. THE CURRENT PASS) TO A PASS FILE *)
(* ON DISKETTE. THE TWO PROCEDURES ABOVE AID IN THIS QUEST.*)
VAR PARANUM:INTEGER;
BEGIN (* WRITEPASS *)
    SHOWPROC('CMJV',SHOW);
    CHECKMJVFILE; (* CHECK ON CONTENTS OF MJV ARRAY *)
    SHOWPROC('CMJV',ERASE);
    SHOWPROC('WPAS',SHOW);
    RESETIRQ; (* DISABLE INTERRUPTS WHILE USING DISKETTE DRIVES *)
    SHOWMODE('DISKIO');
    OPENPASSFILE;
    CONDPASSFILE;
    IF NOT MJVONLY THEN
        BEGIN
            FOR PARANUM:=1 TO PARACNT DO
                BEGIN
                    PASSFILE^:=PASSPARA[PARANUM];
                    PUT(PASSFILE)
                END;
        END;
    CLOSEPASSFILE;
    INITPIA; (* ENABLE INTERRUPTS AGAIN *)
    SHOWMODE('ACTIVE');
    SHOWPROC('WPAS',ERASE);
END; (* WRITEPASS *)

PROCEDURE SETPARAMETERS;
(* THIS ROUTINE ATTEMPTS TO READ THE PASS FILE NAMING *)
(* PARAMETERS FROM #4:RCV.PARAM.TEXT. IF THE FILE *)
(* CANNOT BE OPENED, AN ERROR MESSAGE IS DISPLAYED *)
(* AND THE PROGRAM Terminates. *)
(* THE PARAMETERS EXPECTED ARE: (ON SEPARATE LINES) *)
(* 1. ROOTSUFFIX - FOR PASSES *)
(* 2. ROOTSUFFIX - FOR VOTED PASS FILE *)
(* 3. NEXT PASSNUMBER & PASSNUMBERINCREMENT *)
(* 4. EXTENSION *)
(* MODIFIED - SH QUEK *)
(* ADD LINE 2 TO READ MAJORITY VOTED FILE NAME PREFIX *)
VAR
STRNUM,STRINC,STRST : STRING;
BEGIN
SHOWPROC('SETP',SHOW);
SHOWMODE('DISKIO');
PFDEVICE:='#5:';
(*$I-*) RESET(ParamFile,'#4:RCV.PARAM.TEXT'); (*$I+*)
IF IORESULT<>0 THEN
BEGIN
SHOWMSG(1,'UNABLE TO OPEN #4:RCV.PARAM.TEXT');
NOTE(35,50);
EXIT(PROGRAM)
END;
READLN(ParamFile,PfRootName);
READLN(ParamFile,MjRootName);
READLN(ParamFile,PfNumber,PfIncrement);
READLN(ParamFile,PfExt);
STR(PfNumber,STRNum);
STR(PfIncrement,STRINC);
STRST:= CONCATC(' ',MjRootName,')',STRNum,PfExt,(' ',STRINC,')');
SHOWMSG(0,CONCATC('PARAMS - ',PfRootName,STRST));
CLOSE(ParamFile,NORMAL);
SHOWPROC('SETP',ERASE);
END;

BEGIN (* SATNAV *)
(* DEFAULT OPTIONS FOR THE PROGRAM *)
MINDOP30 := 10; (* MIN. NO OF 30-SEC DOPRS FOR PASS ACCPT *)
MINPARA := 5; (* MINIMUM PARAGRAPHS ACCUMULATED B4 SAVING *)
FAILMJVRJ := 'Y'; (* REJECT PASS IF FAILS MJV MESSAGE TESTS *)
ZEROUNDOP := 'Y'; (* ZERO 30-SECOND DOPPLERS IF DIFF > REFMAX *)
FAILDOPRJ := 'Y'; (* REJECT PASS IF FAILS MIN COUNT *)
MJVONLY := TRUE;(* SAVE ONLY MJV DATA ON THE DISK *)
WELCOME;
ADDINFO;
FORMATSCREEN;
SHOWPROC('SATN',SHOW);
STR((2*MEMAVAIL),MEMUNUSED);
SHOWMSG(1, CONCAT('MEMAVAIL AT SETUP = ', MEMUNUSED, ' BYTES'));
SETPARAMETERS;
INITPIA;
SHOWMODE('ACTIVE');

INPUTLINE[10] := " ";
INPUTLINE[20] := " ";
INPUTLINE[30] := CHR(13);
SEQCODES := "5679ABDEF";
ESCPRESSED := FALSE;
QUITREQUESTED := FALSE;
UNITCLEAR(2);

REPEAT
   SHOWMSG(1, 'USER <ESC> COMMANDS: Q,S,U,K');
   READPASS;
   IF PARACNT >= MINPARA THEN
      WRITEPASS
   UNTIL QUITREQUESTED;

SHOWMODE('QUIT');
RESETIRQ;
SHOWPROC('SATN', ERASE);
PAGE(OUTPUT)

END. (* SATNAV *)
PROGRAM LINK

DESCRIPTION AND USER'S GUIDE
This supplement describes the various modifications to the Digital Data Transfer (TALK) program developed originally by Mark S. Lord, and described in Technical Report 88 of the Department of Surveying Engineering, University of New Brunswick.

These changes have been implemented in a revised version of the TALK program, called LINK.

The program can now communicate with VSPC on the IBM 3032 at 1200 baud and has the capability of saving VSPC files on Apple diskettes.

A user's guide to LINK is provided in Appendix I.
# PROGRAM LINK

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>78</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>79</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>80</td>
</tr>
<tr>
<td>2. LINK</td>
<td>81</td>
</tr>
<tr>
<td>2.1 1200 Baud Communication</td>
<td>81</td>
</tr>
<tr>
<td>2.2 Additional features</td>
<td>82</td>
</tr>
<tr>
<td>3. Conclusions and Recommendations</td>
<td>83</td>
</tr>
<tr>
<td>References</td>
<td>84</td>
</tr>
<tr>
<td>Appendix I - User's Guide</td>
<td>85</td>
</tr>
<tr>
<td>Appendix II - Components of the Apple II+ system</td>
<td>94</td>
</tr>
<tr>
<td>Appendix III - LINK Program Listing</td>
<td>95</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The TALK program was initially developed to enable the Apple II+ to communicate with VSPC. This allowed the transfer of accumulated Doppler pass files to the IBM for further processing. Since the inception of the program, the communication line from VSPC has been upgraded to support a 1200 baud data rate. Accessing the higher baud rate decreases considerably the time needed to transfer a set of passes from the Apple to VSPC. To achieve 1200 baud communication, it was necessary to develop an assembler interface to handle the transmissions between the Apple and the IBM. The basic structure of the program, however, remains identical to that described in Lord [1982].
2. **LINK**

LINK is the latest version of the TALK program developed originally by Lord [1982] to transfer Doppler data to the IBM. To support 1200 baud communication between the Apple and the IBM, several assembler routines were added. The program has been segmented to accommodate more object code; i.e., only part of the program remains in memory at a time.

2.1 **1200 Baud Communication**

The TALK program was designed to use a 300 baud communication link to the IBM. When the hardware was upgraded to support 1200 baud communication, the program, due to the inherent slowness of the PASCAL language in which TALK is written, was unable to keep up, and this resulted in missing characters. In an effort to speed the operation of the program, the transfer and receiving routines have been rewritten in assembler language. Data coming in from the IBM is placed in a 256 character buffer by the assembler routines. The program then empties the buffer at its own pace. Data going out to the IBM is sent whenever the transmit register is free.

To ensure that all incoming characters are placed in the buffer, the interrupt capability of the Apple II+ is employed. Whenever data is received from the IBM, the program is interrupted and the assembler interrupt service routine picks up the input character from the receive register and places it in the input queue. The program later picks up the data from the queue in the order in which they arrived. When the rate at which data are received exceeds the rate the program is emptying the queue, the buffer starts to fill up. There may be instances when the buffer gets
full. In this unlikely event a '?' appears, denoting missing characters due to the buffer overflowing.

2.2 Additional Features

2.2.1 Pass Transfer Routine

The pass transfer routine now has the capability of transferring both the majority-voted data file and the raw pass file to VSPC. It can be requested to transfer only the majority-voted file.

2.2.2 Copy Routine

The LINK program now supports two-way transfers of text files between VSPC and the Apple. The copy routine, which enables a VSPC file to be saved on the Apple diskette, was extracted from Slipp [1982].
3. CONCLUSIONS AND RECOMMENDATIONS

The LINK program, although primarily developed for the IBM 3032, can be modified to operate with any computer. Changes to the communication protocol can easily be done with the LINK program. Baud rate, parity and data bit changes are achieved by writing appropriate values to the command and control registers of the Super Serial Card.

Versatility of the LINK program will allow data to be transmitted from and to the field via a telephone link.
REFERENCES


APPENDIX I

Program : LINK

Author : S.H. Quek

Language : PASCAL

Compiler : APPLE PASCAL (1.1)

Type : Interactive

Purpose : Software Package for Apple II+ to VSPC Communications.

Date : June 1983
**Execution of Program**

The following describes the start-up, or "booting", procedure for the execution of the LINK program.

1. Before turning on the power to the Apple slide the diskette named TRACK into disk drive 1 (as identified by the label on the front of the drive) and a Pascal-formatted diskette into the other drive.

2. Power on the Apple.

3. The screen should display the current time, i.e. year/month/day hour/min/second. A request to type 'I' sometimes appears. Do this if requested.

4. If system fails to boot, try swapping the diskettes in the disk drives. If problem persists, contact author.

5. If booting is successful the system is now in the APPLE PASCAL Command Mode. To enter the Execution Mode, type 'X'. The system then asks for the name of the file (program) you wish to execute.

6. Type 'GO' to execute the front program. Disk drive 1 will whirr a bit and a menu of available programs will be displayed.

7. Type '2' to select the LINK program. Note that the LINK program requires the use of the Super Serial Card in the Apple for communication to VSPC.

*(Note: Type 'X' means hitting the X key on the keyboard without the quotes)*
Link
The LINK program is an extended version of the original TALK program (by Mark Lord - see reference) currently existing in the Department of Surveying Engineering. The original program was modified to
a) operate at 1200 baud,
b) allow for the transfer of a satellite Doppler majority-voted matrix along with the pass file down to VSPC,
and c) copy VSPC files onto Apple diskettes.
The name of the program was changed to identify this version of the software.
After the program identification message (see figure 1), a short menu (see figure 2) appears and a "beep" prompts the user to select an option from the menu. The following are the options supported by this program.

1. Dumb Terminal Mode
This option allows the Apple keyboard and screen to be used as a half-duplex asynchronous ASCII terminal. This allows the user full control over the VSPC logon procedure, since it is up to the user to achieve logon. (Two keystrokes of the 'RETURN' key should elicit a response from VSPC). The scintillating cursor is used to identify this mode of operation. To exit from this mode, type a 'CTRL <C>'.
Figure 1 - Program LINK Identification Message

APPLE II COMMUNICATIONS INTERFACE

USING SSC TO VSPC

BY

SEE HEAN QUEK

(1983)
Figure 2 - Program Menu

---

SUPER SERIAL COMMUNICATIONS PROGRAM

---

COMMAND MODE

OPTIONS ARE:
- D = DUMB TERMINAL MODE
- P = TRANSFER SATELLITE PASS FILES
- T = TRANSFER ANY TEXT FILE
- C = COPY ANY VSPC FILE
- Q = QUIT

---

ENTER COMMAND =>
2. Pass File Transfer Mode

This mode allows the user to transfer pass and majority-voted files, created by the SATNAV program, to VSPC. Before this mode is entered, the user should first sign on to VSPC using the Dumb Terminal Mode.

To transfer pass and majority-voted data to VSPC, the user has to specify which files are to be sent. The program first prompts for a "rootname" of the pass file, which is the name of the diskette and files (residing on it) to be transferred, less the file number suffixed to each name. For example, if the pass file and majority-voted file are called "PASS100.TEXT" and "MJV100.TEXT" respectively on the diskette in drive #5, then the rootname of the pass file will be "#5:PASS:TEXT".

The program next prompts for the majority-voted file prefix. Using the above example, the prefix would be 'MJV'.

The program then requests the starting, ending and increment of the range of suffixed numbers associated with each file(s) to be transferred. The starting number is incremented until it sequence through the specified range of files to be transferred. The program further asks the user whether to transfer only the majority-voted file (i.e. only "#5:MJV100.TEXT") for each pass file or both files (i.e. both "#5:MJV100.TEXT" and 
"#5:PASS100.TEXT"). If a given file is not found or cannot be opened, the reason for it is displayed and the
program proceeds to the next file until all requested file transfers are attempted or until the user interrupts the routine by hitting a 'CTRL <C>' to return to the main menu.

Each pass file in VSPC has the majority-voted data and, depending on the above option chosen, the raw tracking data. Each pass is stored in a separate VSPC file. The name of each VSPC file is constructed from the pass file name on the diskette. If the pass file name is "#5:PASS100.TEXT", the VSPC file name would be "PASS100".

3. Text File Transfer Mode

This mode is similar to that described above for the transfer of pass files, except that it will work for any diskette file with a "_.TEXT" suffix. This mode must be used for transferring files which have been edited using the PASCAL System Editor, including pass files. The program prompts for a file name, including diskette name, and then asks for a VSPC workspace name under which to save the text. Once the user has entered these two items, the program proceeds to transfer all the text contained in the diskette file until all has been sent and saved in VSPC, or until the user hits 'CTRL <C>' to return to the main menu. An example of a diskette file name is: 'DATA:MYPROG.TEXT'

4. Copy VSPC files

This option allows the user to copy text files in the VSPC library onto Apple diskettes. The maximum file size
that can be copied at one time is about 25000 characters.
If the size of the file exceeds 13000 characters, it is stored as two separate .TEXT files on the diskette. If the file size exceeds 25000 characters, a warning to that effect is issued and LINK attempts to save the text file up to the last carriage return before exceeding 25000 characters.

5. Quitting the LINK program

This option allows the user to gracefully exit from the LINK program.

Special Keys

During execution of the program the following keys are programmed for non-standard purposes:

1. Left Arrow - This key generates a VSPC "RUBOUT" sequence of a backspace followed by a linefeed. This effectively "deletes" the last character typed on the current line.

2. Right Arrow - This key generates tab characters, the same as a "TAB" key on most standard terminals. The tabs will show up as a single space on the display screen.

3. CTRL <C> - This code is obtained by typing a 'C' while holding down the 'CTRL' key. It causes an immediate return to the LINK program's main menu and can be used to terminate file transfers prematurely. Note that all I/O will remain in the state prior to exit from the transfer routines, i.e. all files remain in the open state and VSPC in the Input Mode.
Warnings
Terminating the LINK program does not log you off from VSPC. Therefore you have to return to the Dumb Terminal Mode in order to sign off after accessing the other routines.
It is not advisable to go for coffee or tea when the transfer routine starts transmitting data to VSPC. Data loss does occur during transmission and when that happens, the input line (i.e. the last data line shown on the screen) has to be manually typed in (hit 'RETURN' at the end of the input) and the data transfer continues upon receiving the 'RETURN' signal.
Switching off the Apple will log you off from VSPC.

Reference


Components of the Apple II+ System

The following is the list of hardware components required by the LINK program.

1. Apple II+ microcomputer.
2. Video monitor.
3. Disk controller, and two disk drives.
4. 16K language board.
5. Apple Super Serial Card.
Files

FTAPP2: FTACIA.TEXT - assembler code
FTAPP2: FTCOPY.TEXT - include file*
FTAPP2: FTCOM.TEXT - main program

* Include files are files that are inserted into the main program at the time of compilation. They are usually used when the text files are too large for the PASCAL editor.
*******************************
*       *
*  FILE : FTAPP2:FTACIA.TEXT *
*       *
*******************************
.TITLE "COMMUNICATION ROUTINES"
.MACROLIST
.PATCHLIST

; MACRO TO POP 16-BIT RETURN ADDRESS:

.MACRO POP
PLA
STA %1
PLA
STA %1+1
.ENDM

; MACRO TO PUSH 16-BIT RETURN ADDRESS:

.MACRO PUSH
LDA %1+1
PHA
LDA %1
PHA
.ENDM

; MEMORY MAP FOR SYS6551 ACIA

;APPLE SLOT NUMBER OF SUPER SERIAL CARD
SSCSLOT .EQU 2
SSCBASE .EQU <SSCSLOT*10>+0C080
DIPSW1 .EQU SSCBASE+1 ;DIP SWITCH 1 MEMORY
DIPSW2 .EQU SSCBASE+2 ;DIP SWITCH 2 MEMORY
TDREG .EQU SSCBASE+8 ;TRANSMIT REGISTER
RDREG .EQU SSCBASE+8 ;RECEIVE REGISTER
STATUS .EQU SSCBASE+9 ;STATUS REGISTER
SSCMD .EQU SSCBASE+0A;COMMAND REGISTER
SSCNTL .EQU SSCBASE+0B;CONTROL REGISTER

; SPECIAL SYSTEM MONITOR LOCATIONS:
IRQVECTR .EQU OFFFE ;BASE ADDRESS OF IRQ/BRK INTERRUPT VECTOR
LANGCARD .EQU 0C080 ;BASE ADDRESS FOR SLOT#0 = LANGUAGE-CARD

; PASCAL-SUPPLIED ZERO-PAGE TEMPORARY WORK AREAS:
RTADDR .EQU 00 ;SAVE AREA FOR PASCAL RETURN ADDRESS
CHRINT .EQU 02 ;TEMPORARY WORKAREA - WORD
CHRCHR .EQU 04 ;TEMPORARY WORKAREA - WORD

; ROUTINE TO INITIALIZE SSC AND BUFFER QUEUE:
.PROC INITSSC ;ROUTINE TO INITIALIZE SSC HANDLING
.DEF OLDIRQ
.REF QFWDPTR, QBKWPTR, QBYTE1, IRQHANDL

START
SEI ;DISABLE INTERRUPTS UNTIL DONE
POP RTADDR ;POP RETURN ADDRESS FROM STACK
LDA #38 ;LOAD CONTROL REGISTER
STA SSCNTL
LDA #29 ;LOAD COMMAND REGISTER
STA SSCMD

LDA #00 ;SET BACKWARD POINTER
STA QBKWPTR ;SAVE BACKWARD POINTER
LDA #01 ;SET FORWARD POINTER
STA QFWDPTR ;SAVE FORWARD POINTER

LDA LANGCARD+OB ;REMOVE WRITE LANG-CARD WRITE-PROTECT
LDA LANGCARD+OB ;THIS INSTRUCTION HAS TO BE DONE TWICE

LDA IRQVECTR ;GET LSB OF CURRENT IRQ VECTOR
STA OLDIRQ ;SAVE FOR INTERRUPT HANDLER
LDA IRQVECTR+1;GET MSB OF CURRENT IRQ VECTOR
STA OLDIRQ+1 ;SAVE FOR INTERRUPT HANDLER

LDA IRQADR ;GET MSB OF IRQ ROUTINE ADDRESS
STA IRQVECTR ;STORE IN MSB OF IRQ VECTOR
LDA IRQADR+1 ;GET LSB OF IRQ ROUTINE ADDRESS
STA IRQVECTR+1;STORE IN LSB OF IRQ VECTOR

LDA LANGCARD+B;WRITE PROTECT THE LANG-CARD AGAIN

CLI ;ENABLE INTERRUPTS AGAIN

PUSH RTADDR ;PUSH RETURN ADDRESS BACK ONTO STACK
RTS ;RETURN TO CALLING PROGRAM

OLDIRQ .WORD 0000 ;SAVE AREA FOR ORIGINAL MONITOR IRQ VECTOR
IRQADR .WORD IRQHANDL ;ADDRESS OF INTERRUPT ROUTINE, 16-BIT

; PROCEDURE TO DISABLE SSC INTERRUPTS AND
; RESTORE ORIGINAL IRQ/BRK VECTOR

.PROC RESETIRQ ;CLEANUP ROUTINE FOR END-OF-PROCESSING
.REF OLDIRQ

START
SEI ;DISABLE INTERRUPTS

LDA #29 ;LOAD COMMAND WORD
STA SSCMD
LDA LANGCARD+OB ;REMOVE WRITE LANG-CARD WRITE-PROTECT
LDA LANGCARD+OB ;INSTRUCTION HAS TO BE DONE TWICE
LDA OLDIRQ ;GET LSB OF ORIGINAL IRQ ADDRESS
STA IRQVECTR ;STORE IN IRQ VECTOR
LDA OLDIRQ+1 ;GET MSB OF ORIGINAL IRQ ADDRESS
STA IRQVECTR+1 ;STORE IN IRQ VECTOR
LDA LANGCARD+8 ;WRITE PROTECT THE LANGUAGE-CARD AGAIN
RTS ;RETURN TO CALLING PROGRAM

;PROCEDURE TO SEND CHARACTER
.PROC SENDCHAR,1 ;PROCEDURE TO TRANSMIT DATA

POP RTADDR ;SAVE PASCAL RET ADDR
POP CHRINT ;SAVE PARAMETER

NOTCLEAR LDA STATUS ;CHECK STATUS BIT
AND #10
BEQ NOTCLEAR

TRANSMIT LDA CHRINT ;CLEAR FOR TRANSMIT
AND #7F ;STRIP PARITY
STA TDREB ;TRANSMIT
CLI ;ENABLE INTERRUPT

RESTORE PUSH RTADDR ;PUSH RETURN ADDRESS
RTS ;EXIT SUBROUTINE

; PROCEDURE TO RETURN THE NEXT "CHAR" FROM THE QUEUE:

.PROC GETCHAR,2 ;PROCEDURE TO EXTRACT CHAR
.DEF IRQHANDL,QBYTE1,QBKWPTR,QFWDPTR
.REF OLDIRQ

START POP RTADDR ;SAVE PASCAL RETURN ADDRESS
POP CHRINT ;SAVE ADDRESS OF INTEGER PARAMETER
POP CHRCHR ;SAVE ADDRESS OF CHAR PARAMETER
LDY #01
LDA #00 ;CLEAR ACCUMULATOR
STA @CHRINT,Y ;CLEAR HIGH BYTE
STA @CHRCHR,Y ;CLEAR HIGH BYTE
DEY

LDX QBKWPTR ;GET BACKWARD POINTER FOR BUFFER
INX ;POINT TO NEXT WORD IN BUFFER
CPX QFWDPTR ;CHECK FOR EMPTY QUEUE
BNE GETBYTE1 ;BRANCH IF NOT EMPTY
UNDFFLOW
LDA #00
STA @CHRINT,Y
STA @CHRCHR,Y
BEQ EXITGET ;ALWAYS BRANCH (TO EXIT)

GETBYTE1 LDA QBYTE1,X ;GET CHARACTER FROM BUFFER
STA @CHRINT,Y ;SAVE VALUE FROM BUFFER
STA @CHRCHR,Y ;SAVE ASCII FROM BUFFER
STX QBKWPTR

EXITGET PUSH RTADDR ;PUSH PASCAL RETURN ADDRESS ON STACK
RTS ;RETURN TO CALLING PROGRAM

QBYTE1 .BLOCK 256 ;QUEUE AREA FOR 8 BITS OF INPUT DATA
QFWDPTR .BYTE 00 ;POINTER TO NEXT EMPTY LOCATION IN QUEUE
QBKWPTR .BYTE 00 ;POINTER TO ITEM BY NEXT VALUE IN QUEUE

; INTERRUPT-DRIVEN ROUTINE TO BUFFER DATA FROM THE SSC.
; NOT CODED FOR RE-ENTRY.
; INTERRUPTS ARE RE-ENABLED BY THE RTI INSTRUCTION.

OVFLCHAR .EQU 3F ;USE AS OVERFLOW INDICATOR

IRQHANDL STA SAVEACC ;SAVE ACCUMULATOR
PLA ;GET STATUS REG FROM STACK
PHA ;RESTORE ONTO STACK
AND #10 ;TEST "B" BIT
BEQ NOTBRK ;SKIP NEXT SECTION IF TRUE INTERRUPT

NOTSSC LDA SAVEACC ;RESTORE ACCUMULATOR CONTENTS
JMP @OLDIRQ ;BRANCH TO MONITOR'S IRQ/BRK ROUTINE

NOTBRK LDA STATUS ;WAS IRQ CAUSED BY SSC?
BPL NOTSSC ;IF NOT, BRANCH TO MONITOR'S IRQ/BRK
TXA ;SAVE INDEX-X ON STACK
PHA

LDX QFWDPTR ;SET UP QUEUE POINTER IN INDEX-X
CPX QBKWPTR ;CHECK FOR FULL QUEUE
BNE SAVEDATA ;BRANCH IF QUEUE IS OK

OVERRUN LDA #OVFLCHAR ;LOAD QUEUE OVERFLOW CHARACTER
DEX ;POINT AT PREVIOUS QUEUE ELEMENTS
STA QBYTE1,X ;SAVE IN PLACE OF LAST IN QUEUE
BNE EXITIRQ ;ALWAYS BRANCH

SAVEDATA LDA RDREG ;GET 8-BIT DATA
AND  #7F ;STRIP PARITY
STA  QBYTE1,X ;SAVE THEM AS QBYTE1
AND  #75 ;STRIP LINEFEEDS
BEQ  EXITIRQ ;NO SAVE

INX    ;ADVANCE QUEUE POINTER
STX  QFWDPTR ;SAVE NEW FORWARD POINTER FOR QUEUE

EXITIRQ PLA ;RESTORE INDEX-X FROM STACK
TAX
LDA  SAVEACC ;RESTORE ACCUMULATOR
RTI ;RETURN TO INTERRUPTED ROUTINE

SAVEACC .BYTE 00 ;ACCUMULATOR SAVE AREA

.END
FILE: FTAPP2:FTCOPY.TEXT
SEGMENT PROCEDURE TEXTCOPY;

(*----------------------------------------------------------------------------*
 *PROCEDURE TO COPY TEXT FILES IN VSPC *
 *ONTO THE APPLE DISK DRIVES.          *
 *----------------------------------------------------------------------------*)

CONST BUFLIMT = 25000; (* MAX SIZE OF VSPC CHAR BUFFER*)

VAR ONEFILE :BOOLEAN;
BUFFSIZE :INTEGER;
FILE1 :INTEGER;
CHARBUFF :PACKED ARRAY [1..BUFLIMT] OF CHAR;
TEXTFILE :TEXT;
BLKS, FILENAME, VSPCNAME :STRING[20];

PROCEDURE DUMP (NAME : STRING;
START, STOP : INTEGER);

VAR I, IOERR : INTEGER;
SUFFIX : STRING[1];
TEXTNAME : STRING[40];
BEGIN
REPEAT
  WRITE(' RECEIVING FILE : ');
  IF ONEFILE THEN
    SUFFIX := ''
  ELSE
    IF 0=LENGTH(NAME) THEN
      SUFFIX :='1'
    ELSE
      SUFFIX := '2';
  END
  IF 0=LENGTH(NAME) THEN
    BEGIN
      READLN(FILENAME);
      IF 0=LENGTH(FILENAME) THEN
        EXIT(PROCESSCOMMAND);
      I := POS('.',FILENAME);
      IF I<>0 THEN
        FILENAME := COPY(FILENAME,1,I-1)
    END
  ELSE
    BEGIN
      WRITELN(NAME,SUFFIX,'.TEXT');
      FILENAME := NAME
    END;
TEXTNAME := CONCAT(FILENAME,SUFFIX,'.TEXT[',BLKS,']');
(*$I-*)
REWRITE(TEXTFILE, TEXTNAME);
IOERR := IORESULT;
(*$I++*)
WRITELN;
IF IOERR<>0 THEN
BEGIN
  WRITE(chr(7));
  WRITE(' ');
  IF IOERR=8 THEN
    WRITELN('NO ROOM ON DISK')
  ELSE
    WRITELN('I/O ERROR ', IOERR);
  WRITELN
END;
UNTIL IOERR=0;
FOR I := START TO STOP DO
  WRITE (TEXTFILE, CHARBUFF[I]);
WRITELN (TEXTFILE);
CLOSE (TEXTFILE, LOCK)
END; (* DUMP *)

PROPERTY COPYWS1;
(************************************************************************
 * THIS ROUTINE COPIES A BLOCK OF DATA *
 * FROM A USER SPECIFIED VSPC WORKSPACE* 
 * OR THE OUTPUT FROM A FORTRAN PROGRAM* 
 * AND STORES IT ON A DISKETTE. THE *
 * BLOCK SIZE IS DEPENDENT UPON THE *
 * "BUFFLIMIT" CONSTANT DECLARED AT THE*
 * BEGINNING OF THIS PROGRAM. *
***********************************************************************)
VAR MESSAGE : STRING;
IOERR, I : INTEGER;
BEGIN
WRITELN('== PROCEDURE TO COPY IBM FILES ==');
WRITELN;
WRITELN('== SELECT 1 OF THE FOLLOWING OPTIONS ==');
I := MEMAVAIL*2;
WRITELN(' (MEMORY LEFT : ',I:5,' BYTES) '); 
WRITELN;
WRITELN(' 1. - COPY VSPC FILE CONTENTS');
WRITELN(' 2. - RUN A VS FORTRAN PROGRAM');
WRITELN;
WRITE('== ENTER 1 OR 2 : '); 
REPEAT
  READ(KEYBOARD, KBCHR)
UNTIL KBCHR IN ['1', '2'];
WRITELN(KBCHR);
WRITELN;
IF KBCHR = '1' THEN
BEGIN
  WRITELN('== ENTER THE VSPC FILE NAME ==');
  WRITE('== FILE NAME == ');
  READLN(VSPCNAME);
  IF 0=LENGTH(VSPCNAME) THEN
    EXIT (PROCESSCOMMAND);
  PAGE(OUTPUT);
  XMITVSPC('TAPE');
  XMITVSPC('');
  XMITVSPC(CONCAT('LOAD ',VSPCNAME));
  MESSAGE:='LIST NOLINE '
END
ELSE
BEGIN
  WRITELN('== ENTER THE FORTRAN PROGRAM''S NAME');
  WRITE('== PROGRAM NAME == ');
  READLN(VSPCNAME);
  IF 0=LENGTH(VSPCNAME) THEN
    EXIT (PROCESSCOMMAND);
  PAGE(OUTPUT);
  XMITVSPC('TAPE');
  XMITVSPC('');
  MESSAGE := CONCAT('RUN ',VSPCNAME,' ')
END;
MESSAGE[LENGTH(MESSAGE)] := CHR(13);
I:=0;
REPEAT
  I:=I+1;
  SCANKEYBOARD;
  SCANACIA;
  WRITE(MESSAGE[I]);
  SENDACIA(ORD(MESSAGE[I]))
UNTIL MESSAGE[I] = CHR(13)
END; (* COPYWS1 *)

PROCEDURE COPYWS2;
VAR LASTCR :BOOLEAN;
I,L :INTEGER;
BEGIN
  I:=1;
  SCANACIA;
  WHILE (REPLYVAL <> DC1) AND (I <= BUFFLIMT) DO
    BEGIN
      CHARBUFF[I] := CREPLYVAL;
      IF REPLYVAL <> 0 THEN
        I := I + 1;
      SCANKEYBOARD;
      SCANACIA
    END
  END
END
FILE1 := 13000;
L := 13000;
IF (I > 13000) THEN
  REPEAT
    L := L + 1;
    IF ORD(CHARBUFF[L]) = 13 THEN
      FILE1 := L;
  UNTIL (FILE1 = L) OR (L = I);

IF I > BUFLIMIT THEN
BEGIN
  WRITELN('** BUFFER FULL **');
  WRITE(CHR(7));
  LASTCR := FALSE;
  I := BUFLIMIT;
  REPEAT
    I := I - 1;
    IF (ORD(CHARBUFF[I]) = 13) THEN
      LASTCR := TRUE;
  UNTIL LASTCR;
END;
BUFSIZE := I - 1;
PAGE(OUTPUT)
END; (* COPYWS2 *)

PROCEDURE COPYWS3;
VAR I1, I2 : INTEGER;
BEGIN
  IF BUFSIZE <= 16000 THEN
    BEGIN
      I1 := 2 * ROUND(1.5 + BUFSIZE / 1024);
      ONEFILE := TRUE
    END
  ELSE
    BEGIN
      I1 := 2 * ROUND(1.5 + FILE1 / 1024);
      I2 := 2 * ROUND(1.5 + (BUFSIZE - FILE1 - 1) / 1024);
      ONEFILE := FALSE
    END;
GOTOXY (10, 1); WRITELN('== COPY IBM FILES ==');
GOTOXY (0, 4); WRITELN('1. IBM --> MEMORY');
GOTOXY (31, 4); WRITELN('ASCII');
GOTOXY (5, 6); WRITELN('SOURCE FILE : ', VSPCNAME);
GOTOXY (5, 10); WRITELN('BUFFSIZE:5, CHARACTERS WITHIN BUFFER');
GOTOXY (0, 13); WRITELN('2. MEMORY --> DISK');
IF ONEFILE THEN
BEGIN

GOTOXY (5,15); WRITELN('ESTIMATED BLOCK REQUIREMENT',I1:4);
WRITELN; STR(I1,BLKS);
DUMP('',1,BUFFSIZE)
END
ELSE
BEGIN
GOTOXY (30,13); WRITELN('2 FILES');
GOTOXY (4,15); WRITELN('-FILE #1 : BLOCK REQUIREMENT',I1:5);
WRITELN; STR (I1,BLKS);
DUMP('',1,FILE1);
WRITELN;
WRITELN('-FILE #2 : BLOCK REQUIREMENT',I2:5);
WRITELN; STR (I2,BLKS);
DUMP (FILENAME,FILE1+1,BUFFSIZE)
END
END; (* COPYWS3 *)

BEGIN (*TEXTCOPY*)

COPYWS1;
COPYWS2;
COPYWS3

END; (*TEXTCOPY*)
(*SWAPPING MODE ON*)

PROGRAM:FTCOM (OLD NAME: TALK)
* WRITTEN: 19-APR-82 BY MARK S LORD
* MODIFIED BY
* 21-JAN-82 BY SEE HEAN QUEK
* REMODIFIED
* 8-APR-83 BY SEE HEAN QUEK

THIS PROGRAM ALLOWS COMMUNICATIONS BETWEEN THE APPLE COMPUTER AND AN OUTSIDE SOURCE, VIA THE SUPER SERIAL INTERFACE CARD IN APPLE SLOT #2.

INTERFACE CARD - SSC

FOUR DIFFERENT MODES OF OPERATION CAN BE USED AS SELECTED FROM THE PROGRAM'S MAIN MENU:

D)UMB TERMINAL MODE:
* KEYS: <CTRL^-C> RETURNS USER TO MAIN PROGRAM MENU.
* <RIGHTARROW> ACTS AS A VSPC CHARACTER <DEL> KEY.
* <LEFTARROW> SENDS A TAB CHARACTER TO VSPC.

T)RANSFER TEXT MODE:
* <ESC> AND <CTRL^-C> KEYS MAY BE USED TO PREMATURELY TERMINATE THE TRANSFER.

C)OPY VSPC FILE
* RECEIVES AND STORES FILES ON DISKETTE. USING IBM PROTOCOL.

P)ASSFILE TRANSFERING:
* THE STANDARD CHARACTER SEQUENCE:
* #5:PASS340.TEXT
* #5:PASS350.TEXT
* #5:PASS370.TEXT
* THEN THE APPROPRIATE ROOTNAME WOULD BE: #5:PASS.TEXT
* THIS IS FOLLOWED BY A REQUEST FOR THE 2ND ROOTSUFFIX WHICH FOLLOWING THE ABOVE EXAMPLE:
* #5:MJV340.TEXT
* #5:MJV350.TEXT
* #5:MJV370.TEXT
* WHERE THESE ARE THE MAJORITY *
* VOTED DATA FILES FOR THE ABOVE *
* PASSES, THEN THE APPROPRIATE *
* 2ND ROOTSUFFIX WOULD BE: MJV *

********************************************************************

PROGRAM FTCOM;
USES APPLESTUFF, PEEKPOKE;
CONST ESCAPE = 27; (* ASCII CODE FOR <ESC> CHAR. *)
LINEFEED = 10; (* ASCII CODE FOR <LF> CHAR. *)
LEFTARROW = 8; (* CODE FOR SPECIAL APPLE KEY *)
RIGHTARROW = 21; (* CODE FOR SPECIAL APPLE KEY *)
CTRLC = 3; (* ASCII CODE FOR CONTROL^C *)
DC1 = 17; (* ASCII CODE FOR DC1 CHAR. *)

TYPE LONGSTRING = STRING[255];

VAR KBCHR: CHAR;
KBVAL, REPLYVAL: INTEGER;
CREPLYVAL: CHAR;
QUITREQUESTED: BOOLEAN;
I, J: INTEGER;

PROCEDURE INITSSC; EXTERNAL;
PROCEDURE RESETIRQ; EXTERNAL;
PROCEDURE GETCHAR(VAR CHRCHR: CHAR;
VAR CHRINT: INTEGER); EXTERNAL;
PROCEDURE SENDCHAR(OUTVALUE: INTEGER); EXTERNAL;

(* FORWARD BLOCK *)
PROCEDURE SENDACIA(OUTVALUE: INTEGER); FORWARD;
PROCEDURE SCANACIA; FORWARD;
PROCEDURE SCANKEYBOARD; FORWARD;
PROCEDURE PROCESSCOMMAND; FORWARD;
PROCEDURE XMITVSPC(MESSAGE: LONGSTRING); FORWARD;

SEGMENT PROCEDURE DUMBTERMINAL;
********************************************************************
* THIS ROUTINE ALLOWS DIRECT USER *
* COMMUNICATIONS WITH A REMOTE DEVICE *
* BY CAUSING THE APPLE TO BEHAVE AS A *
* NON-INTELLIGENT ASYNC ASCII TERMINAL*
********************************************************************

BEGIN (* DUMBTERMINAL *)
    WRITELN('== DUMB TERMINAL MODE ==');
    WRITELN;
    WRITELN('== HIT <CTRL^C> TO QUIT ==');
    WRITELN(CHR(7));
    KBVAL := 0;
REPEAT
   SCANACIA;
   SCANKEYBOARD
   UNTIL KBVAL=CTRLC;
END; (* DUMBTERMINAL *)

(*$I #5:FTCOPY.TEXT *)

SEGMENT PROCEDURE PASSTRANSFER;
(*******************************************************************************
 * THIS ROUTINE SERVES AS THE DRIVER *
 * FOR THE SENDPASS ROUTINE. IT PROMPTS *
 * THE USER FOR THE PASS FILE RANGES *
 * AND THEN LOOPS, CALLING SENDPASS TO *
 * TRANSFER INDIVIDUAL PASS FILES. OPEN* *
 * ERRORS ARE LOGGED ON THE SCREEN FOR *
 * THE USER TO OBSERVE AS THE PROGRAM *
 * CONTINUES WITH THE NEXT FILE IN SEQ.*
*******************************************************************************)

TYPE DATALINE = PACKED ARRAY[1..26] OF CHAR;
TIMESTAMP = PACKED ARRAY[1..20] OF CHAR;
MESSLINE = PACKED ARRAY[1..32] OF CHAR;
PARARECORD = RECORD
   PASSTIME:TIMESTAMP;
   PASSLINE:ARRAY[1..25] OF DATALINE
END;

VAR ROOTNAME,PASSNAME,NUMSTRING,VSPCNAME,OUTSTRING:STRING;
MJVNAME,MJVSUFFIX:STRING;
PASSPARA :PARARECORD;
MJVPASS :MESSLINE;
MJVONLY : BOOLEAN; (* PASS TO IBM ONLY MJV FILES *)

PASSFILE :FILE OF PARARECORD;
MJVFILE :FILE OF MESSLINE;
ANSWER :CHAR;

DOTPOS,IOERR,PASSNUM,LASTNUM,INCREMENT:INTEGER;
IOERR2,SEMICOLON:INTEGER;

PROCEDURE SHORTRANSFER;
BEGIN
   WRITELN;WRITELN('TRANSFER ONLY MAJORITY VOTED DATA ?');
   REPEAT
      WRITE('==> ');
      READLN(ANSWER);
      IF NOT (ANSWER IN ['Y','N']) THEN
         WRITELN('Y OR N. RE-ENTER ');
      UNTIL ANSWER IN ['Y','N'];
      IF ANSWER='Y' THEN
MJVONLY := TRUE
ELSE
MJVONLY := FALSE;
END;

PROCEDURE SENDPASS;
(* **************************************************************
 * THIS ROUTINE HANDLES THE ACTUAL *
 * TRANSFER OF A PRE-OPENED PASS FILE *
 * TO VSPC. A VSPC WORKSPACE IS NAMED *
 * AND SAVED FOR THE PASS, THE NAME *
 * USED BEING THE SAME AS THAT OF THE *
 * PASS FILE, LESS DEVICE NAME AND *
 * EXTENSION OF COURSE. *
 * **************************************************************
 VAR DOTPOS, LINENUM: INTEGER;
BEGIN (* SENDPASS *)
 XMITVSPC('CLEAR');
 VSPCNAME:= PASSNAME;
 DELETE(VSPCNAME, 1, POS(';', VSPCNAME));
 DOTPOS:= POS(';', VSPCNAME);
 DELETE(VSPCNAME, DOTPOS, (1+ LENGTH(VSPCNAME) - DOTPOS));
 XMITVSPC(CONCAT('NAME ', VSPCNAME));
 XMITVSPC('INPUT 1 1 ');
REPEAT
 BEGIN
 MJVPASS:= MJVFILE^;
 OUTSTRING := '';
 MOVELEFT(MJVPASS, OUTSTRING[1], 31);
 XMITVSPC(OUTSTRING);
 GET(MJVFILE)
 END
 UNTIL EOF(MJVFILE);
(* AVOIDED IF MJV TRANSFERED ONLY *)
IF NOT MJVONLY THEN
BEGIN
REPEAT
 WITH PASSPARA DO
 BEGIN
 PASSPARA:= PASSFILE^;
 OUTSTRING := '';
 MOVELEFT(PASSTIME[1], OUTSTRING[1], 19);
 XMITVSPC(OUTSTRING);
 OUTSTRING := '';
 FOR LINENUM:= 1 TO 25 DO
 BEGIN
 MOVELEFT(PASSLINE[LINENUM], OUTSTRING[1], 25);
 XMITVSPC(OUTSTRING)
 END;
 END
*)
XMITVSPC('TAPE');
XMITVSPC('');
XMITVSPC('ENTER DATA');
XMITVSPC('');

REPEAT
  SCANACIA;
  SCANKEYBOARD;
  PASSNAME:=ROOTNAME;
  MJVNAME := ROOTNAME;
  STR(PASSNUM,NUMSTRING);
  SEMICOLON :=POS(':' ,MJVNAME);
  DELETE(MJVNAME, (SEMICOLON+1) , (DOTPOS-SEMICOLON-1));
  INSERT(CONCAT(MJVSUFFIX,NUMSTRING),MJVNAME,POS('.',MJVNAME));
  INSERT(NUMSTRING,PASSNAME,DOTPOS);
  (**I**) RESET(PASSFILE,PASSNAME);
  IOERR:=IORESULT;
  RESET(MJVFILE,MJVNAME); (**I**)}
  IOERR2:=IRESULT;

  WRITELN;
  WRITELN(PASSNAME,' AND/OR ',MJVNAME);
  WRITELN;
  IF (IOERR=0) AND (IOERR2=0) THEN
    BEGIN
      WRITELN('== NOW BEING SENT ==');
      SENDPASS;
      CLOSE(PASSFILE);
      CLOSE(MJVFILE)
    END;
  ELSE
    IF (IOERR=10) OR (IOERR2=10) THEN
      WRITELN('== NOT FOUND ==')
    ELSE
      BEGIN
        WRITELN(CHR(7),PASSNAME,' OPEN ERR#',IOERR,' ==')
        WRITELN(CHR(7),MJVNAME,' OPEN ERR#',IOERR2,' ==')
      END;
    PASSNUM:=PASSNUM+INCREMENT
    UNTIL PASSNUM>LASTNUM;
END; (* PASSTRANSFER *)

SEGMENT PROCEDURE TEXTTRANSFER;
(**************************************************************************
 * THIS ROUTINE HANDLES TRANSFERRING OF *
 * NORMAL TEXT FILES TO VSPC. THE USER *
 * IS PROMPTED FOR A FILE SPECIFICATION*
GET(PASSFILE)
END
UNTIL EOF(PASSFILE);
END;
XMITVSPC(''');
XMITVSPC(CONCAT(''SAVE'',VSPCNAME));
CLOSE(PASSFILE);
CLOSE(MJVFILE);
END;(*SENDPASS*)

BEGIN (*PASSTRANSFER*)
WRITELN('==PASSFILETRANSFERPROCEDURE==''');
WRITELN;
WRITELN('==ENTERROOT-NAME(DEV:SUFFIX.EXT)==''');(*$I-*);
REPEAT
    WRITE('==ENTER==>');
    READLN(ROOTNAME);
    IF LENGTH(ROOTNAME)=0 THEN
        EXIT(PASSTRANSFER);
    DOTPOS:=POS('',ROOTNAME)
    UNTIL NOT(DOTPOS IN [0,1,LENGTH(ROOTNAME)]);
    RESET(PASSFILE,ROOTNAME);
    IOERR:=IRESULT;
    CLOSE(PASSFILE)
UNTIL (IOERR<>7);(*WAITFORVALIDFILESPEC*)
WRITELN;
WRITELN('==ENTERMAJORITYVOTEDFILE''SPREFIX==''');
WRITE('==ENTER==>');
READLN(MJVSSUFFIX);
WRITELN('==ENTERPASSNUMBERRANGE==''');
PASSTNUM:=0;
REPEAT
    WRITE('==ENTERFIRSTNUMBER==>');
    READLN(PASSNUM)
UNTIL (IRESULT=0) AND (PASSTNUM>0);
LASTNUM:=0;
REPEAT
    WRITE('==ENTERLASTNUMBER==>');
    READLN(LASTNUM)
UNTIL (IRESULT=0) AND (LASTNUM>0);
INCREMENT:=0;
REPEAT
    WRITE('==ENTERINCREMENT=====>');
    READLN(INCREMENT)
UNTIL (IRESULT=0) AND (INCREMENT>0);
SHORTTRANSFER;(*$I++*)
* IN WHICH THE "TEXT" IS OPTIONAL, *
* AND THEN PROCEEDS TO TRANSFER THE *
* FILE TO A USER-SPECIFIED VSPC WS. *
*****************************************************************************

VAR TEXTNAME, VSPCNAME: STRING;
  TEXTLINE: LONGSTRING;
  IOERR: INTEGER;
  TEXTFILE: TEXT;
BEGIN (* TEXTTRANSFER *)
  WRITELN('== PROCEDURE TO TRANSFER TEXT FILES ==');
  WRITELN;
  WRITELN('== ENTER NAME OF FILE ==');
  REPEAT
    WRITE('== FILE NAME ==> ');
    READLN(TEXTNAME);
    IF LENGTH(TEXTNAME)=0 THEN
      EXIT(TEXTTRANSFER);
    (*$I-*)
    RESET(TEXTFILE, TEXTNAME);
    IOERR:=IORESULT;
    IF IOERR=10 THEN
      BEGIN
        INSERT('.TEXT', TEXTNAME, (1+LENGTH(TEXTNAME)));
        RESET(TEXTFILE, TEXTNAME);
        IOERR:=IORESULT
      END;
    (*$I+*)
    IF IOERR<>0 THEN
      BEGIN
        IF IOERR=10 THEN
          WRITELN('FILE NOT FOUND - RE-ENTER')
        ELSE
          WRITELN('OPEN ERROR #', IOERR, ' - RE-ENTER')
      END
    UNTIL IOERR=0;
  WRITELN;
  WRITELN('== ENTER NAME FOR VSPC WORKSPACE ==');
  WRITE('== WORKSPACE NAME ==> ');
  READLN(VSPCNAME);
  IF 0=LENGTH(VSPCNAME) THEN
    BEGIN
      CLOSE(TEXTFILE);
      EXIT(TEXTTRANSFER)
    END;
  XMITVSPC('TAPE');
  XMITVSPC('');
  XMITVSPC('CLEAR');
  XMITVSPC(CONCAT('NAME ', VSPCNAME));
  XMITVSPC('INPUT 1 1');
  IF NOT EOF(TEXTFILE) THEN
REPEAT
  READLN(TEXTFILE,TEXTLINE);
  IF LENGTH(TEXTLINE)=0 THEN
    TEXTLINE:='';
  XMITVSPC(TEXTLINE)
UNTIL EOF(TEXTFILE);
XMITVSPC('');
XMITVSPC(CONCAT('SAVE ',VSPCNAME));
CLOSE(TEXTFILE);
END; (* TEXTTRANSFER *)

PROCEDURE SCANACIA;
  (***************************************
   * THIS ROUTINE SCANS THE INTERNAL *
   * BUFFER FOR INCOMING DATA. IF PRESENT*
   * IT IS DISPLAYED ON THE APPLE MONITOR*
   * AND THE ASCII NUMERIC VALUE IS *
   * PLACED IN "REPLYVAL". *
   ***************************************)
BEGIN (* SCANACIA *)
  GETCHAR(CREPLYVAL,REPLYVAL);
  WRITE(CREPLYVAL);
END; (* SCANACIA *)

PROCEDURE SENDACIA; (*OUTVALUE:INTEGER*)
  (***************************************
   * THIS ROUTINE WILL TRANSMIT A BYTE *
   * OUT THROUGH THE ACIA. IT WAITS *
   * UNTIL THE "READY" FLAG OF THE ACIA *
   * IS SET, AND THEN TRANSFERS THE DATA *
   * BYTE SPECIFIED BY ITS ASCII NUMERIC *
   * VALUE IN "OUTVALUE". *
   ***************************************)
BEGIN (* SENDACIA *)
  SENDCHAR(OUTVALUE)
END; (* SENDACIA *)

PROCEDURE SCANKEYBOARD;
  (***************************************
   * THIS ROUTINE CHECKS TO SEE IF ANY *
   * MORE KEYBOARD INPUT HAS BEEN ENTERED*
   * BY THE USER. IF SO, IT IS PROCESSED*
   * AS DESCRIBED AT THE TOP OF THIS *
   * PROGRAM IN THE D)DUMB TERMINAL CMD. *
   ***************************************)
BEGIN (* SCANKEYBOARD *)
  IF KEYPRESS THEN
    BEGIN
      READ( KEYBOARD, KBCHR );
    END; (* KEYPRESS *)
END; (* SCANKEYBOARD *)
IF EOLN(KEYBOARD) THEN
  KBCHR := CHR(13);
  KBVAL := ORD(KBCHR);
IF KBVAL IN [ESCAPE, CTRL, LEFTARROW, RIGHTARROW] THEN
  CASE KBVAL OF
    ESCAPE:
      BEGIN
        (* NOT AVAILABLE *)
      END;
    CTRL:
      BEGIN
        WRITELN(CHR(7), '<CTRL-C>');
        EXIT(PROCESSCOMMAND)
      END;
    LEFTARROW:
      BEGIN
        WRITE(KBCHR, ' ', KBCHR);
        SENDACIA(KBVAL);
        SENDACIA(LINEFEED)
      END;
    RIGHTARROW:
      BEGIN
        KBCHR := CHR(9);
        WRITE(KBCHR);
        SENDACIA(9)
      END (* CASE *)
  ELSE
    BEGIN
      WRITE(KBCHR);
      SENDACIA(KBVAL)
    END
  END;
END; (* SCANKEYBOARD *)

PROCEDURE XMITVSPC (*MESSAGE:LONGSTRING*);

(* This routine uses "SENDACIA" to transmit a line of characters to VSPC. A carriage
return is sent at the end of the line, and all chars sent are also echoed on the
Apple's monitor as they are transmitted. *)

VAR I:INTEGER;
BEGIN (* XMITVSPC *)
  MESSAGE := CONCAT(MESSAGE, ' ');
  MESSAGE[LENGTH(MESSAGE)] := CHR(13);
  I := 0;
  REPEAT

I:=I+1;
SCANKEYBOARD;
SCANACIA;
WRITE(MESSAGE[I]);
SENDACIA(ORD(MESSAGE[I]));
UNTIL MESSAGE[I]=CHR(13);
REPEAT
SCANKEYBOARD;
SCANACIA
UNTIL REPLYVAL=DC1;
END; (* XMITVSPC *)

PROCEDURE PROCESSCOMMAND;
("***********************
* THIS ROUTINE SERVES AS A COMMON *
* INTERFACE BETWEEN THE MAIN PROGRAM *
* AND THE COMMAND-PROCESSING PROC’S. *
* IT’S PRESENCE IS REQUIRED IN ORDER *
* TO ALLOW SCANKEYBOARD TO HAVE A *
* COMMON EXIT POINT FOR HANDLING A *
* USER <CTRL<C> COMMAND. *
*******************************
BEGIN (* PROCESSCOMMAND *)
CASE KBCHR OF
  'Q':QUITREQUESTED:=TRUE;
  'D':DUMBTERRMINAL;
  'P':PASSTRAFER;
  'T':TEXTTRANSFER;
  'C':TEXTCOPY;
END; (* CASE *)
END; (* PROCESSCOMMAND *)

("***********************
* THE MAIN ROUTINE (BELOW) HANDLES *
* GENERAL INITIALIZATION AND THE *
* PROMPTING FOR, AND INPUT OF, USER *
* COMMAND OPTIONS FROM ITS MAIN MENU. *
*******************************
BEGIN (* FTCOM *)
QUITREQUESTED:=FALSE;
PAGE(OUTPUT);
GOTOXY(0,5);
WRITELN('APPLE II COMMUNICATIONS INTERFACE ');
WRITELN;
WRITELN(' USING SSC TO VSPC ');
WRITELN;
WRITELN;
WRITELN(' BY');
WRITELN;
WRITELN('SEE HEAN QUEK');
WRITELN;
WRITELN;
WRITELN;
WRITELN;
WRITELN;
WRITELN('1983');

INITSSC;

FOR I := 1 TO 2000 DO
  J := J;

PAGE(OUTPUT);
REPEAT
  PAGE(OUTPUT);
  WRITELN('== SUPER SERIAL COMMUNICATIONS PROGRAM ==');
  WRITELN;
  WRITELN('== COMMAND MODE ==');
  WRITELN;
  WRITELN('OPTIONS ARE:');
  WRITELN('D = DUMB TERMINAL MODE');
  WRITELN('P = TRANSFER SATELLITE PASS FILES');
  WRITELN('T = TRANSFER ANY TEXT FILE');
  WRITELN('C = COPY ANY VSPC FILE');
  WRITELN('Q = QUIT');
  WRITELN;
  WRITE ('==ENTER COMMAND ==> ');
  REPEAT
    WRITE(CHR(7));
    READ(KEYBOARD,KBCHR)
    UNTIL KBCHR IN ['D','P','T','C','Q'];
  PAGE(OUTPUT);
  PROCESSCOMMAND;
  WRITELN;
  WRITELN;
  UNTIL QUITREQUESTED;
RESETIRQ;
PAGE(OUTPUT);
END. (* FTCOM *)