Ph 20 ~~Problem Sets~~

Infrastructure Notes

1984

Computational Physics Laboratory

Technical Notes

9 January 1984

G. Fox

N. Wilson

J. Pine

A. Skjellum

C. Stassen

Contents

15 January 1984

I: General Procedures

3 January 1984
A. Skjellum

I. All assignments must be returned in machine-readable form on one or more floppy diskettes. Each diskette must be distinctively marked to indicate the student's name, and section. When more than one diskette is submitted, the diskettes should be numbered in the order in which they are to be reviewed.

   a. On each diskette, a file called README should be included in the top-level directory. The README file(s) should contain an overview of a diskette's contents. Actual write-ups, results and programs should be on separate files. A README file is therefore an index and an abstract for a diskette. Any particularly important parts of the assignment should be stressed in README.

   b. Any plots included in the returned assignment should NOT be submitted in hardcopy form, but instead as disk files suitable for plotting on the IBM XT. Data in such files should be in the PH 20 standard graphics format (PH20SGF) which is described in the Section III of this manual.

   c. New (BASIC or C) routines of general interest should be placed in separate files and include full documentation on their use in remarks included with the source code. The presence of such special files should be highlighted in README.

   d. Several text editors and word processors are available on the computers. Any of these may be used in preparing write-ups. However, it is strongly recommended that the VEDIT text editor by used, as this is the standard editor for Ph 20. WORDSTAR will also be available for those who wish to produce highly finished documents. Indicate the text editor used in the README file on your diskette(s).

   e. Students will be expected to keep machine-readable backup copies of the assignments which they submit.

II. Programming in Ph 20 will be done either in BASIC or in the C language. BASIC is the recommended language for most purposes.

   a. Several introductory BASIC texts will be available in the lab to serve as references.

   b. Programs written in Ph 20 will be expected in include a significant amount of internal documentation to explain the purpose and implementation of the problem under consideration. In BASIC, internal documentation is added via REMark statements.

   c. Any system software or compiler errors which are discovered should be documented in separate files and mentioned in README. A separate file should be included for each problem encountered.

   d. Tutorial programs will be available to introduce the computer system to new and novice users.

III. Report any hardware problems to A. Skjellum as soon as they are discovered. He may be reached at x3952 or in person at 307 E. Bridge, two doors down from the Computation Lab.

IV. Students are expected to purchase their own floppy diskettes Three-packs are available in the Downs Stockroom.

V. The Computer Room is in Room 304 E. Bridge, x3768. Keys to this room and the Bridge building are available from June Bressler in Room 103 of E Bridge.

II: Using the IBM PC for First Time Users

a. Turn on the computer CPU, Monitor and Printer. Wait for the prompt and enter the date (eg. 10-11-83) and time (E.G. 13:47) in 24 hour format.

b. Enter the USERS Directory by typing

```
cd \users
```

c. Create your personal directory by typing

```
md name
```

(Do NOT put SPACES in your directory name and limit the name to 7 characters).

d. Enter your personal directory by typing

```
cd \name
```

Your personal directory may now be used to store data files and files of your own programs. You may also call programs from the system directories while in your directory by simply typing the name of the program. The value of PATH determines which directories can be accessed in this fashion.

e. Next time you want to access your files, use directly

```
cd \users\name
```

Note that we clean up the fixed disk occasionally and files stored on your directory (\users\name) may or may not be present from the last session.

III: The Physics 20-2D Plot Program

A. How to Access Program on Computer

0) Move to your personal directory
1) Copy \PLOT*.* moves plot files to your directory
2) PLOT1
   Invokes plot program.

As a beginner you should probably now type

HE to plot help page.
follow with
LO DMPOSC1.P02 to load a file
PL 1 V 2, 1 V 5 to plot 2 curves

Now you are in the second stage of program

HE plots another help page
x -6, -10
y .4,8
PL Plots a scaled up version of plot
P prints current plot
S returns to Screen 1
EX exits

Physics 20-2D

Jerry Pine

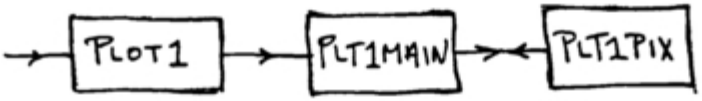
1 November 1983

B: Users Guide to Plotting Program (Plot 1)

Introduction:

The Plot1 program makes x-y plots of data presented to it from disk "plot files" which need to be written in the standard format. See the separate brief note which describes this format. You should limit the length of variable names to ten characters or less so that the space allotted for them in plotting program displays will be adequate. Variable units may take up more room (e.g., up to 20 characters, or even more).

The general structure of the program embodies three major units, as shown below:



Each unit is a separate compiled Basic program and they connect via the Chain command. The initial unit is called PLOT1.EXE, and serves to initialize the program parameters. It then transfers control to PLT1MAIN.EXE, which is the main program for the control of data. When a plot is to be made, PLT1PIX.EXE is loaded to create and to display the plot. Further plots are requested, or the program is exited, on return to PLT1MAIN.

The program utilizes the MAI graphics board for the IBM PC, and produces plots in four colors with a 640 dot by 400 scan line mode. Text displays are also produced, in an 80 character by 25 line format, with a four color 640 dot by 200 line display.

The program was written by J. Pine and Clare Stassen. We will be grateful for your constructive comments and reports of bugs, as well as for suggestions for improving the documentation. We will be glad to answer questions at our section meetings on Wednesday and Thursday afternoons.

Screen 1:

To start the program, type PLOT1, and after initialization is complete PLT1MAIN will produce the "Screen 1, page 1" display. The commands available at this level of the program are listed at the bottom of the screen. The command HELP will display a screen with brief descriptions of the syntax and functions of all commands. Type the command, followed by "carriage return" (or "enter"). More detailed discussions are given below, in the order in which commands might be entered in a session with the program. (For Screen 1, the first two letters of the command name suffice to define the command to the program.)

LOAD filespec . A standard plot file will be read into a single-

precision floating point buffer in high memory. The filename and the variables will be displayed on Screen 1 when the process is complete. The buffer is large, designed to hold up to about 25,000 data quantities. The variable names entered in the file are shown on the Screen 1 display followed by a comma and the number assigned by the program to the file from which they were read. Up to eight files may be read in, with up to a total of 32 variables, so long as the buffer is not filled.

PLOT (xvar1) vs (yvar1),(xvar2) vs (yvar2),... up to six variable pairs. The variables are specified by their Screen 1 numbers. As in all commands the arguments, such as (xvar1), "vs", (yvar1), etc.,can be separated by any number of commas or spaces to suit the taste of the user. The delimiter "vs" in this command can be any string. The first variable pair defines the auto-scaling of the plot and the units labels. The succeeding pairs are superposed, and the plot-maker is responsible for their units making sense, etc. (At present, the total number of variable values is limited to a maximum of 8,000.) Upon receipt of a plot command, the PLTIPIX.EXE program produces the graph, which we call Screen 2. The commands which may be executed from Screen 2 are discussed in the next section.

PAGE (page no.). One Screen 2 command saves the plot data, and another returns to Screen 1. When this happens the saved plot is written to a disk file, and an abbreviated description of the plot appears on the Screen 1, page 1 display. More detailed descriptions of saved plots may be viewed on pages 2-10 by using the Page command.

UNSAVE (plot no.). A saved plot can be "unsaved" by erasing it from the disk when the program is exited. A single such command may be used with up to 20 plot numbers as arguments. The file name suffix will be shown as ".UNS" for unsaved plots. During the plotting session they are equivalent to saved plots.

DISPLAY (plot no.). A saved or unsaved plot may be redisplayed by the display command. The full repertoire of Screen 2 commands will be operational, just as if the displayed plot were made from a plot command. In addition, if a plot file name is given as an argument instead of a number, plots saved from a previous session can be displayed. In this case, the Save and Unsave commands are not operational.

PRINT. Printer output will be produced which includes the Screen 1 information on pages 1-10. It is not directly copied from the screen, but the format is similar. A sample output is shown on the following page.

EXIT. Returns to DOS after erasing the unsaved files from the disk.

Files Read In:

1.DMPOSC1.P03 2.DMPOSC1.P03 3.DMPOSC1.P02 4.DMPOSC.PLT

Variables,File No.:

1. TIME,1	2. X1,1	3. X2,1	4. X3,1
5. X4,1	6. TIME,2	7. X1,2	8. X2,2
9. X3,2	10. X4,2	11. TIME,3	12. X1,3
13. X2,3	14. X3,3	15. X4,3	16. TIME,4
17. X1,4	18. X2,4	19. X3,4	20. X4,4

Plots Saved:

1.DMPOSP03.UNS: X1,1 VS TIME,1
X4,1 VS TIME,1
This is a crude approximation.
Y-LIMITS: -10 TO 10 TIMES 10^{-4} CM.
X-LIMITS: 0 TO 9 TIMES 10^{-3} SEC.

2.DMPOSP02.S02: X4,3 VS TIME,3
Maximum damping, and good approximation.
Y-LIMITS: -11 TO 1 TIMES 10^{-4} CM.
X-LIMITS: 30 TO 80 TIMES 10^{-3} SEC.

3.DMPOSPLT.S03: X1,4 VS TIME,4
Eight cycles, undamped, for reference.
Y-LIMITS: -10 TO 10 TIMES 10^{-4} CM.
X-LIMITS: 0 TO 10 TIMES 10^{-4} SEC.

Screen 2:

Screen 2 displays plots, and appears after either a Plot or Display command on Screen 1. Initially, the plot is automatically scaled to include the full range of data for the first-named variable pair. This can be changed by Screen 2 commands and other functions may be performed, as described below in detail. (For Screen 2 the first one or two letters, shown in capitals on the screen display, define the command.)

HELP . Gives brief descriptions of the command syntax and the functions described below.

XLIMITS (lower) (upper). New plot limits are assigned by the two arguments, which are to be in the same units as the axis scale of the currently displayed plot (e.g. 20,30 if a plot with x from 0 to 100 x 10^4 cm were shown). The limits become effective when the plot is redrawn with a Plot command. If no arguments are given, the x-range is reset to the initial default full range.

YLIMITS (lower) (upper). Same as Xlimits, for y-axis.

PLOT . Replots graph, using the x and y limits last entered.

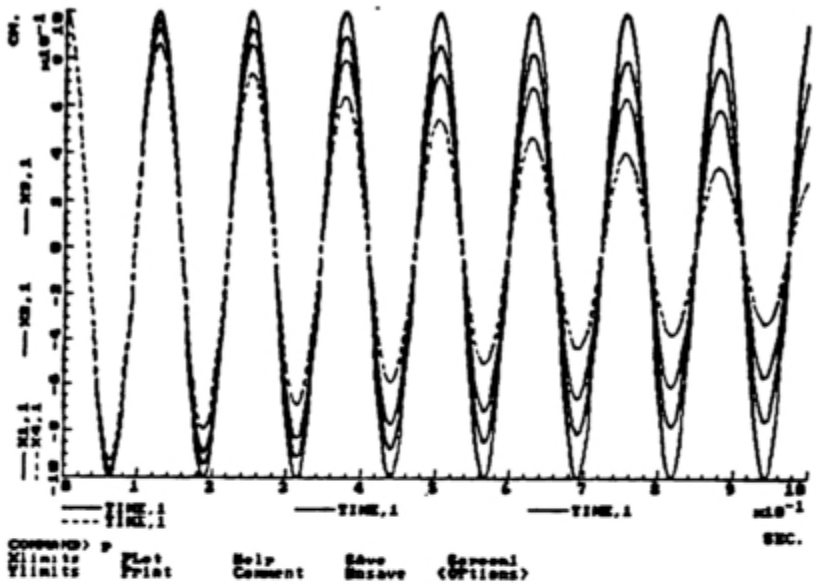
SAVE . Causes the full range of variable data for a plot to be saved on a disk file when the program returns to Screen 1. A default filename is constructed from the first five letters and the three-letter suffix of the file from which the first x-variable was obtained. A suffix Sxx is given, where xx is the saved file number. The user may override the default and specify any filename he prefers by entering it as an argument following the Save command. When a Saved plot is recalled with a Display command from Screen 1, the initial display is with the x and y limits in force when the plot was saved. However, the limits may be changed because the full range of data is available. A displayed file called by number from the current session of the program may be resaved, to show a different set of x and y limits when redisplayed.

UNSAVE . Identical to Save, except that the file will be erased from disk on exit from the program. The filename suffix .UNS indicates this.

PRINT . Produces a copy of the display on the printer. The slow printer speed causes this process to take 3-4 minutes. Ultimately, this command will accept as an argument a filename for a file in which the data can be stored for later printing, and storage to disk will only require about ten seconds. An example of a plot printout is shown on the following page.

SCREEN 1 . Returns to Screen 1, for more plots or displays, or exit.

OPTIONS . Reserved for future use, to expand the repertoire of command functions.


SEC.
COMMAND> P
Xlimits Plot Help Save Screen
Ylimits Print Comment Unsave (Options)

C. How to Construct Standard Plot Files

The PLOT1 plotting package works from disk files. To use it your physics program needs to write appropriate files. The easiest way to explain this is by example. The first of the two following pages is a typed file written in the proper format. It is a table, in this case five columns wide and 92 rows long, preceded by a few lines of header information. The first entry, 5, tells the number of columns. The five succeeding lines are the variable name and units appropriate to each column of "data".

To create this table is simple in Basic. The second attached page is a Basic program which calculates the exact solution for a damped harmonic oscillator, and was used to write the file shown on the preceding page. You can see that the Basic WRITE # statement does it all.

The file typed here is available as a test file for the Plot1 plotting package, so you can see how it works. Its name is dmposc1.p02. Another file derived from this one by taking every tenth data point is called dmposc1.p03. It is also available for your amusement, especially for running superposed plots of this "approximate" solution and the more exact one.

```
E DMPOSC1.P02
File not found

A>TYPE B:DMPOSC1.P02
5
"TIME","SEC."
"X1","CM."
"X2","CM."
"X3","CM."
"X4","CM."
0,1,1,1,1
.001,.9987503,.9984507,.9981512,.9975525
.002,.9950041,.9944072,.9938108,.9926189
.003,.9887711,.9878816,.9869929,.9852179
.004,.9800666,.9788912,.9777172,.9753735
.005,.9689125,.9674602,.9660101,.9631164
.006,.9553366,.9536185,.9519035,.9484829
.007,.9393728,.9374022,.9354357,.9315151
.008,.921061,.918853,.9166505,.9122611
9.000001E-03,.9004471,.8980192,.8955979,.8907746
.01,.8775826,.8749538,.8723328,.8671145
.011,.8525245,.8497158,.8469163,.8413451
.012,.8253356,.8223698,.8194146,.813536
.013,.7960838,.7929851,.7898985,.7837613
.014,.7648422,.7616366,.7584444,.7521002
.015,.7316889,.7284037,.7251332,.7186363
```

```
700001E-02,-.7593988,-.7420577,-.7251127,-.6923745
.078,-.7259322,-.7091427,-.6927413,-.661068
.079,-.6906511,-.6744751,-.6586779,-.628185
8.000001E-02,-.6536435,-.6381428,-.6230097,-.5938116
.081,-.6150025,-.6002381,-.5858281,-.5580376
.082,-.5748243,-.5608562,-.5472274,-.5209554
.083,-.5332087,-.5200958,-.5073053,-.4826602
8.400001E-02,-.4902608,-.4780606,-.466164,-.4432516
.085,-.4460878,-.4348563,-.4239077,-.4028304
.086,-.400799,-.3905906,-.3806423,-.3614993
8.700001E-02,-.3545089,-.3453759,-.3364782,-.3193646
8.800001E-02,-.3073326,-.2993252,-.2915264,-.2765331
.089,-.2693885,-.2525545,-.2459005,-.2331138
.09,-.2107961,-.2051808,-.199715,-.1892164
9.100001E-02,-.1616761,-.1573221,-.1530852,-.1449509
```

```
type dmposc1.src
'****DMPOSC1.BAS, A PROGRAM TO GENERATE A PLOTTING FILE
'****WHICH CONTAINS DATA FOR A ONE-DIMENSIONAL DAMPED
'****HARMONIC OSCILLATOR.
'
'** THE DISPLACEMENTS X1, X2, X3, AND X4 WILL BE GIVEN
'**AS FUNCTIONS OF TIME, FOR FOUR DIFFERENT DAMPING
'**CONSTANTS.  THE DISPLACEMENTS WILL BE GIVEN AT EACH
'**MILLISECOND, UP TO A TOTAL TIME OF ONE SECOND.
'
'*INITIALIZING
'
OPTION BASE 1 'SUBSCRIPTS BEGIN AT 1
DEFINT I,J,K,L,M,N
DIM X(4) 'DISPLACEMENTS
DIM B(4) 'DAMPING COEFFS
DIM VAR$(5) 'VARIABLE NAMES
DIM UNITS$(5) 'VARIABLE UNITS
IVAR = 5 'NUMBER OF VARIABLES
AMP = 1.0 'INITIAL DISPLACEMENT
DO I=1 TO 4
  READ B(I)
ENDDO
DATA 0.0,0.3,0.6,1.2
DO I=1 TO 5
  READ VAR$(I)
  READ UNITS$(I)
ENDDO
DATA "TIME","SEC.","X1","CM.","X2","CM.","X3","CM.","X4","CM."
'
'
'**FILE HEADER
'
OPEN "DMPOSC.P01" FOR OUTPUT AS #1
WRITE #1, IVAR
DO I=1 TO 5
  WRITE #1, VAR$(I), UNITS$(I)
ENDDO
'
'
'**LOOP TO GENERATE DATA
'
T=0.
DO J=0 TO 1000
  T=.001*J
  DO I=1 TO 4
    X(I)= AMP*EXP(-B(I)*T)*COS(50*T)
  ENDDO
  WRITE #1, T, X(1), X(2), X(3), X(4)
ENDDO
'
'
'*FINISH
'
CLOSE #1
```

IV: VEDIT Summary

DESCRIPTION OF FILES FOR THE IBM PERSONAL COMPUTER

FOR IBM'S PCDOS

The following is a brief description of the files currently supplied on diskette for the IBM Personal Computer. You will not have to perform the customization process, described in the manual, to produce a runnable version of VEDIT. You may use the pre-configured VEDIT supplied, which follows the "Example Keyboard Layout for the IBM Personal Computer". To customize a NEW version of VEDIT, follow the instructions in the VEDIT manual under Appendix A, (Customizing VEDIT). Note that this version of VEDIT is a Memory Mapped version which accesses the screen directly.

VEDSET.COM	The program used to perform customization. The manual describes the use of this program and the ".SET" file below.
VEDIT-86.SET	File for producing the VEDIT version for the IBM P.C.
VEDIT.COM	'Runnable' version of VEDIT.

EXAMPLE KEYBOARD LAYOUT FOR THE IBM PERSONAL COMPUTER

"ESCAPE MODE CHARACTER #1" NOT USED Type [RETURN]

[HOME]	[CTRL-HOME]	
[ZEND]	[CTRL-END]	
[CURSOR UP]	[Up Arrow]	
[CURSOR DOWN]	[Down Arrow]	
[CURSOR RIGHT]	[Right Arrow]	
[CURSOR LEFT]	[Left Arrow]	
[BACK TAB]	[F3]	
[TAB CURSOR]	[F4]	Useful for fast cursor movement.
[ZIP]	[F6]	
[NEXT LINE]	[F5]	
[LINE TOGGLE]	[CTRL-J]	
[SCROLL UP]	[HOME]	
[SCROLL DOWN]	[END]	
[PREVIOUS WORD]	[CTRL-V]	
[NEXT WORD]	[CTRL-B]	
[PREVIOUS PARAGRAPH]	[CTRL-Pg Up]	
[NEXT PARAGRAPH]	[CTRL-Pg Dn]	
[PAGE UP]	[Pg Up]	
[PAGE DOWN]	[Pg Dn]	
[SCREEN TOGGLE]	[CTRL-K]	
[BACKSPACE]	[<---]	
[DELETE]	[Del]	
[ERASE TO END OF LINE]	[CTRL-Z]	Also called [EREOL] in manual.
[ERASE LINE]	[CTRL-X]	
[DEL PREVIOUS WORD]	[CTRL-C]	
[DEL NEXT WORD]	[CTRL-N]	
[UNDO]	[CTRL-U]	
[TAB CHARACTER]	[TAB]	
[NEXT CHAR LITERAL]	[CTRL-L]	
[SET INSERT MODE]	NOT USED	Type [RETURN]
[RESET INSERT MODE]	NOT USED	Type [RETURN]
[SWITCH INSERT MODE]	[Ins]	
[REPEAT]	[CTRL-R]	
[INDENT]	[F8]	
[UNDENT]	[F7]	
[COPY TO TEXT REGISTER]	[F9]	
[MOVE TO TEXT REGISTER]	[ALT-F9]	
[INSERT TEXT REGISTER]	[F10]	
[PRINT TEXT]	[CTRL-P]	
[SET TEXT MARKER]	[CTRL-S]	
[GOTO TEXT MARKER]	[CTRL-G]	
[FORMAT PARAGRAPH]	[CTRL-F]	
[VISUAL ESCAPE]	[ESC]	
[VISUAL EXIT]	[CTRL-E]	Used to exit to command mode.
[RESTART EDITOR]	NOT USED	Type [RETURN]. (Use "EA" Command.)

Note: VEDIT will run on either the graphics or monochrome display.

VEDIT FOR THE IBM PERSONAL COMPUTER

EXAMPLE CUSTOMIZATION FOR MEMORY MAPPED VERSION 1.15

The customization session used to create the pre-configured VEDIT is listed below. The pre-configured keyboard layout is described by the enclosed layout sheet. This session begins with Task 3. of the customization. For clarity sake, each reply below is preceded by "--", which does not appear in the actual customization. Each numerical reply must be followed by the [RETURN] key.

```
3.)  HEX CODE FOR SCREEN CONTINUATION CHARACTER   (2D) -- AD
     HEX CODE FOR COMMAND ESCAPE CHARACTER        (1B) -- 1B
     HEX CODE FOR COMMAND ITERATION LEFT BRACKET  (5B) -- 5B
     HEX CODE FOR COMMAND ITERATION RIGHT BRACKET (5D) -- 5D
     HEX CODE FOR CURSOR CHARACTER          (5F) -- 5F
     HEX CODE FOR CLEAR SCREEN CHARACTER    (20) -- 20
     HEX CODE FOR STATUS LINE CHARACTER     (2D) -- 2D
     HEX CODE FOR TAB EXPAND CHARACTER      (20) -- 20

4.)  EXPAND TAB WITH SPACES         (0=NO, 1=YES) -- 0
     AUTO BUFFERING IN VISUAL MODE (0=NO, 1=FORWARD, 2=BACKWARD) -- 1
     BEGIN IN VISUAL MODE           (0=NO, 1=YES) -- 1
     POINT PAST TEXT REG. INSERT    (0=NO, 1=YES) -- 1
     IGNORE UPPER/LOWER CASE DISTINCTION IN SEARCH (0=NO, 1=YES) -- 1
     CLEAR SCREEN ON VISUAL EXIT    (0=NO, 1=YES) -- 0
     REVERSE UPPER AND LOWER CASE   (0=NO, 1=YES) -- 0
     IGNORE SEARCH ERRORS           (0=NO, 1=YES) -- 0
     EXPLICIT STRING TERMINATORS    (0=NO, 1=YES) -- 0

     CURSOR TYPE (0=UNDERLINE, 1=BLINK BLOCK, 2=BLOCK, 3=ATTRIBUTE) -- 3
     CURSOR BLINK RATE, SMALL # IS FAST (2MHZ - 20, 4MHZ - 40) -- 40
     INDENT INCREMENT  (1 - 20, SUGGEST 4) -- 4
     LOWER CASE CONVERT (0=NO, 1=YES, 2=CONDITIONAL) -- 0
     DECIMAL CODE FOR CONDITIONAL CONVERT CHARACTER (59) -- 59
     LINE AND COLUMN DISPLAY (0=NONE, 1=LINE, 2=COLUMN, 3=BOTH) -- 3
     RIGHT MARGIN AND WORD WRAP (0=OFF) -- 0

5.)  ENTER NUMBER OF SCREEN LINES IN DECIMAL -- 25
     ENTER LINE MOVEMENT FOR PAGING IN DECIMAL -- 21
     ENTER TOP LINE FOR CURSOR IN DECIMAL -- 3
     ENTER BOTTOM LINE FOR CURSOR IN DECIMAL -- 21
     ENTER SCREEN ATTRIBUTE CODE FOR NON-REVERSE VIDEO IN HEX (07) -- 07
     ENTER SCREEN ATTRIBUTE CODE FOR REVERSE VIDEO IN HEX     (70) -- 70
     ENTER SCREEN ATTRIBUTE CODE FOR CURSOR IN HEX            (F0) -- F0

6.)  SIZE IN DECIMAL OF SPARE MEMORY FOR AUTO READ -- 6144
     SIZE IN DECIMAL OF FILE MOVE TRANSFERS IN K BYTES -- 12

     DO YOU WISH TO USE THE DEFAULT TAB POSITIONS? (Y OR N) -- Y

     BEGIN IN INSERT MODE (0=NO, 1=YES) -- 0

     REVERSE VIDEO ON STATUS LINE  (0=NO, 1=YES) -- 1
```

'n' denotes a positive number. (# represents 32767)
'm' denotes a number which may be negative to denote backwards in the file.
'r' denotes a digit "0 - 9" specifying a text register.

'string','s1' and 's2' denote text strings.

'file' is a file name in the normal CP/M (MSDOS) format with optional drive and extension specified.

nA	Append 'n' lines from the input file. (0A)
-nA	Read 'n' lines back from output file. (-0A)
B	Move the edit pointer to text beginning.
mC	Move the edit pointer by 'm' positions.
mD	Delete 'm' characters from the text.
E	First letter of extended two letter commands.
nFstring<ESC>	Search for 'n'th occurrence of 'string'.
Gr	Insert the contents of text register 'r'.
Itext<ESC>	Insert the 'text' into the text buffer.
mK	Kill 'm' lines.
mL	Move the edit pointer by 'm' lines.
Mr	Execute text register 'r' as a command macro.
nNstring<ESC>	Search for 'n'th occurrence of 'string' in file.
mPr	Put 'm' lines of text into text register 'r'.
Ss1<ESC>s2<ESC>	Search for and change 's1' to 's2'.
mT	Type 'm' lines.
U	Print # of unused, used and text register bytes.
V	Go into visual mode.
nW	Write 'n' lines to the output file. (0W)
-0W	Write lines from edit pointer to VEDIT.REV file.
Z	Move edit pointer to end of text.

SPECIAL CHARACTERS

|	Search wildcard character. Each "|" will match any character in the text being searched.
<CTRL-Q>	Literal Character. Next char. is taken literally.
@	Precedes F, I, N, S command to indicate explicit terminator.
:	Precedes F, N, S command to suppress search error message.
#	Represents maximum positive number 32767. Signifies "forever" or "all occurrences of".

EXTENDED COMMANDS

EA		Restart the editor. (EX and EB).	
EBfile		Open "file" for Read & Write, perform an auto-read.	
EC		Change disks for reading or write error recovery.	
ED		Display disk directory. Opt. drive spec. and "?".	
EF		Close the current output file.	
EGfile[line range]		Insert the specified line number range of the file "file" into the text buffer at the edit position.	
nEI		Insert the character whose decimal value is "n".	
EKfile		Erase (kill) the file "file" from the disk.	
mEO		Send 'm' lines to the line printer. (0EO)	
EP n m		Change the value of parameter "n" to "m".	
	1	Cursor type	(0, 1 or 2)
	2	Cursor blink rate	(10 - 100)
	3	Indent Increment	(1 - 20)
	4	Lower case convert	(0, 1 or 2)
	5	Conditional convert character	(32 - 126)
	6	Display line and column number	(0, 1, 2 or 3)
	7	Word Wrap column (0 = Off)	(0 - 255)
EQ		Quit the current edit session.	
ERfile		Open the file "file" for input.	
ES n m		Change the value of switch "n" to "m".	
	1	Expand Tab with spaces	(0=NO 1=YES)
	2	Auto buffering in visual mode	(0=NO 1=YES 2=BACK)
	3	Start in visual mode	(0=NO 1=YES)
	4	Point past text reg. insert	(0=NO 1=YES)
	5	Ignore UC/LC search distinction	(0=NO 1=YES)
	6	Clear screen on visual exit	(0=NO 1=YES)
	7	Reverse Upper and Lower case	(0=NO 1=YES)
	8	Suppress search errors	(0=NO 1=YES)
	9	Explicit string terminators	(0=NO 1=YES)
ET		Set new tab positions.	
EV		Print the VEDIT version number.	
EWfile		Open the file "file" for output. Create Backup.	
EX		Normal exit back to CP/M after writing output file.	
EY		Finish writing and close output file.	

TEXT REGISTER COMMANDS

RDr	Dump contents of register 'r' on console.
RLrfile	Load register 'r' from file 'file'.
RPr	Send contents of register 'r' to line printer
RSrfile	Save contents of register 'r' in file 'file'.
RTr	Type contents of register 'r' on console.
RU	Display size of each text register. br

VEDIT prints a message (on the CP/M console device) when the user should be notified of an unusual or special condition. All messages are descriptive, and the user should not normally have to refer to this appendix in order to understand the message or error. The messages fall into three categories: fatal errors, non-fatal errors and other messages. Fatal errors result in an abort of the disk operation being performed and a return to command mode if possible, else a return to CP/M. These are caused by certain disk errors described below. The non-fatal errors usually just signify that a typo was made or that some small detail was overlooked. These only result in a message and the user can try again.

FATAL ERRORS

NO DISK SPACE — The disk became full before the entire output file was written. As much of the output file as possible was written. Refer to the section on disk write error recovery.

CLOSE ERROR — The output file could not be closed. This is a very unusual condition, but may occur if the disk becomes write protected.

READ ERROR — An error occurred reading a file. This error should never occur, since CP/M itself normally gives an error if there was a problem reading the disk.

NO DIR SPACE — There was no directory space left for the output file. Refer to the section on disk write error recovery.

REV FILE OPEN — You cannot change disks because the VEDIT.REV file is open while performing backward disk buffering.

NON-FATAL ERRORS

INVALID COMMAND — The specified letter is not a command.

CANNOT FIND... — The specified string could not be found. This is the normal return for iteration macros which search for all occurrences of a string.

NESTING ERROR — You cannot nest macros deeper than 8 levels.

MACRO ERROR — Your macro attempted to change the contents of a text register which is currently executing as a command macro.

BAD PARAMETER — Something was specified wrong with your "EI", "EP", "ES" or "ET" command.

NO INPUT FILE — There is no input file open for doing a read or append.

NO OUTPUT FILE — There is no output file open for doing a write, a close or an exit with the "EX" command. If you have already written out the text buffer and closed the output file, exit with the "EQ" command.

CANNOT OPEN TWO — You cannot have two output files open and there is already one open. Also given if an output file is open at the time of an "EC" command. Perhaps you want to close it with the "EF" command.

BAD FILE NAME — The file name you gave does not follow the CP/M conventions.

FILE NOT FOUND — The file you wanted to open for input does not exist. Maybe you specified the wrong drive.

OTHER MESSAGES

NEW FILE — The file specified with the EB command or with the invocation of VEDIT did not exist on disk and a new file has been created. If you typed the wrong file name, you may want to start over by issuing the "EQ" command.

BREAK — The command execution was stopped because insufficient memory space remained to complete the command (I, S, G, P and EG). For the "I", "S" and "EG" commands, as much text as possible was inserted. For the "G" and "P" commands, no text at all was copied or inserted. The message is also printed when command execution is stopped because you typed [CTRL-C] on the keyboard in command mode.

QUIT (Y/N)? — This is the normal prompt following the "EQ" command. Type "Y" or "y" if you really want to quit and exit to CP/M, otherwise type anything else.

INSERT NEW DISK AND TYPE [RETURN] — This is the normal prompt for inserting a new disk with the "EC" command.

V. Aztec C Compiler

A: Access to Compiler

In order to use the C compiler, certain files must be set up in your user subdirectory. For most uses the following will suffice:

```
\ccomp\math.lib
\ccomp\libc.lib
\ccomp\cc.bat
\ccomp\stdio.h
```

Use the Copy command to get these files into your directory, ie

```
copy \ccomp\????.lib c:\users\yourname
copy \ccomp\stdio.h  c:\users\yourname
copy \ccomp\cc bat   c:\users\yourname
```

Compiling

Compilation is most easily achieved by using the CC.BAT batch file. To use this batch file, enter the following command.

```
cc progname
ln progname.o math.lib libc lib
```

where progname c is the name of your C program These commands perform the compilation, assembly, and linkage steps automatically

If you need to combine several C program files, CC BAT can be used as a template for a more sophisticated batch file.

Here is an example of a batch file useful for compiling a program which is stored in two segments prog1.c, prog2.c. The link step is also included:

```
c86 prog1.c                                  (compile)
as86 prog1.asm                               (assemble)
c86 prog2.c                                  (compile)
as86 prog2.asm                               (assemble)
ln prog1.o prog2.o math.lib libc.lib         (link)
```

These commands could also be entered at the console directly.

As a further example, here is a simple compile, link, go batch file:

```
c86 %1.c

as86 %1.asm

ln %1.o math.lib libc.lib
```

It is **essential** that math.lib appear before libc.lib in ln command

B: A Recommended CC.BAT

```
if exist %1 goto :bad                                              [01]
c86 %1.c                                                           [02]
as86 -o %1.o %1.asm                                                [03]
erase %1.asm                                                       [04]
ln %1.o math. lib libc.lib                                         [05]
erase %1.o                                                         [06]
%1                                                                 [07]
goto :end                                                          [08]
:bad                                                               [09]
REM : %1 is a bad argument. HINT - don't use .c                    [10]
:end                                                               [11]
```

The above version of CC.BAT is the best way to avoid untimely destruction of your HARD WORK. (The line numbers in []'s are not part of the file.) If the old version of CC.BAT was executed by typing: CC CPROG.C, the source file (cprog.c) would be destroyed (remember you are supposed to type cc cprog).

This version will not allow your c file to be destroyed by a simple typing error Assume your file name is CPROG.C. If you type: CC CPROG , %1 becomes "CPROG" - "CPROG" is the first argument to CC.BAT. Line [01] checks to see if "CPROG" exists. Assuming you have no file name that is the same as our c file but with no extension, line [01] will discover that "CPROG" (no extension) does not exist, and will go on to compile the program without a problem. But, suppose instead you had typed: CC CPROG.C, accidentally typing the .C. Then line [01] would discover that %1 (now "CPROG.C") does exist, and skips the compiling section, typing out the REM statement. Lines [09] and [11] are batch file labels, line [10] is a REMark that will print whenever it is encountered (along with the REM in front). Line [01] is one particular example of conditional execution in a batch file. If you want to learn more about these (and other) special batch file commands, look in the DOS manual in the commands section under batch commands.

This particular CC.BAT example compiles [02], assembles [03], links with standard library and math library [05], deletes intermediate files (like .o and .ASM files) [04] and [06], and runs the .EXE file [07]. If, for example, you don't want to automatically run the file every time you compile, delete line [07]. Modify this file to do what YOU want it to do - you're the one who is going to use it!

VI: Programming Style

As computers are becoming more prevalent, almost everyone finds that they must deal with them at times. Since you expect to be doing a lot of specialized ("application-oriented") programming, you will find it far easier to modify someone else's code that ALMOST does what you need, rather than to write whole programs from scratch. Sometimes, you will come back to a program after having not seen it for several months. Often, other people will want to use or modify your programs, and it is helpful to them if they can figure out exactly what you're doing. In order for you to be able to make simple modifications without having to go through the whole program, you should try to learn and use some elements of programming style. Later in the course, lack of clarity in your code will be cause for severe point loss.

(1) MODULARIZE AND COMMENT

Do not write subroutines (or even main programs) that are longer than a page or two. If a routine takes more than two pages (about 50 lines), it should probably be broken up into two or more smaller routines. Make each routine do SOMETHING, but do not write routines that try to do too much.

At the beginning of each routine, put a small block of comments that describe (1) what the routine does, and (2) what variables the routine uses/modifies/returns. This will not only help you to debug your code, but it'll also mark the beginning of all of your routines in the listing.

(2) MORE COMMENTING

You do not need to comment on every line - overcommenting can be more confusing than no comments at all. Also avoid comments of the type:

```
A = A + 1     ' increment a
```

The comments in your code should say something that is not immediately obvious (when looking at the lines out of context). For example:

```
A = A + 1                                ' delete the remainder of the string
IF A$(A)=";" THEN A$=LEFT$(A$,A)         '    once parameter 2 is reached
```

More simply, a comment should give the reader an idea of how the given section of code fits into the program; what it does that affects the "outside world".

(3) READABLE CODE

(a) Space your code out -- place blank lines wherever you stop doing one thing and start doing another. This also implies ONE STATEMENT PER LINE (maximum).

(b) Use variable names that give some idea of what the variables are used for (exception: loop variables can have one or two-character names).

(c) INDENT your code. Indent one level for each loop or or subroutine level ("one level" is three or four spaces). Indent at other times if it makes your code clearer. Example,

```
if (a==0)
   b = c;
```

(d) Place spaces wherever you can to make your code readable. You don't have to worry about running out of memory; you DO have to worry about being able to read it (you have to worry even more about your TA being able to read it). Example,

```
IF  MAXLEN < LEN( LEFT$( STR$( EXP( EXPONENT ) ) ) )   THEN  ERRCODE = 2
```

-- or --

```
IF MAXLEN<LEN(LEFT$(STR$(EXP(EXPONENT))))THEN ERRCODE=2
```

Which would you rather read? (and which do you think would get a better grade from your TA?)

(e) Line up columns of things (if the elements share something in common). Example,

```
IF  INPSTR$ = "A"   THEN  GOSUB  2000
IF  INPSTR$ = "XY"  THEN  GOSUB 23400
```

(f) When you do outputting to the screen, include a description of the meanings of the values (until your program is debugged). Output data in tables, or in other neat fashions. Include lines in your program to output debugging data (intermediate results, etc). This will save you a lot of time in the long run. When your program finally works, comment the debugging code out but DO NOT DELETE IT. You never know when your program could break again! The code will also help anybody who wishes to modify your program because it identifies intermediate results.

(g) Do not make use of "magic numbers". Define variables (constants when the language permits) to hold these values. Assign all of them at the beginning of the routine, so that you can find and change them easily. Example,

WRONG:

```
'
' (40 lines of code)
'
FOR  LP = 1  TO  80                ' Output the line stored in LINBUF
   PRINT CHR$( LINBUF( LP ) );
NEXT LP
'
' (40 lines of code)
'
```

RIGHT:

```
NUMCOLS = 80               ' The width (characters) of the output device
'
' (40 lines of code)
'
FOR  LP = 1  TO  NUMCOLS          ' Output the line stored in LINBUF
   PRINT CHR$( LINBUF( LP ) );
NEXT LP
'
' (40 lines of code)
'
```

When your TA asks you to print 132 columns on a line printer, which program would you rather modify? (Remember that the number 80 may appear in many places, not all of which you would necessarily want to change. Also, 79 or 81 may appear in various locations. You should use NUMCOLS-1 and NUMCOLS+1 instead.

(h) Do rigorous error checking. Check inputs, intermediate results, and anything else that you can for errors. Have your code print informative error messages. Example,

RIGHT: "X-value (-47685) in line 32 of input is out of range"
WRONG: "Value out of range"

remember: YOU have to debug and deal with your programs. Taking a few simple steps can drastically reduce debugging time and frustration.

(i) Provide help in interactive programs. If you can't print a menu, then prompt for input with informative prompts. Example,

RIGHT: "X-velocity (M/S)?"
WRONG: "V?"

(j) Always leave a way out of your programs - and always clearly mark what it is. Example (see first BASIC program line 1590).

(k) Separate code and comments. Code should reside at the left edge of the screen (+ indents), comments at the right edge. It isn't possible to always separate them, but an effort should be made.

(l) Your code should be clear. Doing "obscure" things rarely has any advantage. A small speed increase does not justify obscurity. If speed is necessary, write assembly code, but comment it!

(4) EXAMPLES

```
1000 '****************** SIMPLE BASIC PROGRAM ****************************
  10 ' This program asks for a decimal number and a base to convert the
1020 ' number to.
1030 '
1040 '****************** history
1050 ' Written  06/12/83        C. Stassen
1060 ' Modified 09/30/83        J. R. Frosh   (added bases greater than 16)
1070 ' Modified 11/02/83        C. Stassen    (fixed bases greater than 16)
1080 '
1090 '*****************************************************************************
1490 '
1500 PRINT
1510 INPUT   "  BASE" ; BASE
1520 INPUT   "NUMBER" ; INPINT
1530 GOSUB 2000
1540 '
1550 PRINT
1560 PRINT  INPINT;" in base ";BASE;" is ";RETNUM$
1570 PRINT
1580 '
1590 INPUT  "MORE"    ; A$
1600 A$ = LEFT$( A$ , 1 )
1610 IF   (A$ = "Y") OR (A$ = "y")    THEN  GOTO  1500
1620 '
1630 END
1640 '
1450 '
 2000 '****************** CONVERT TO BASE n ********************************
2010 ' This routine accepts an integer (INPINT) and converts it to
2020 ' base BASE ( 2 <= BASE <= 36 ).  The result is stored in RETNUM$
2030 '*****************************************************************************
2040 '
2050 ORD$ = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"   ' "Digits" in order
2060 '
2070 INPINT  = ABS( INPINT )
2080 MAXITER = LOG( INPINT ) / LOG( BASE )      'log base BASE of INPINT is the
2090 '                                          'number of digits in RETNUM$
2100 '
2110 RETNUM$ = "                "    'Initialize RETNUM$
2120 '
2130 FOR LP=0 TO MAXITER
2140    CURDIG = INPINT MOD ( BASE ^ ( LP + 1 ) )  'Figure out what numeral
2150    CURDIG = INT( CURDIG / ( BASE ^ LP ) )     'the current digit is.
2160    '
2170    MID$( RETNUM$, MAXITER + 1 - LP ) = MID$( ORD$, CURDIG + 1, 1 )
2180    '                                                  'Put the digit
2190    '                                                  'in the
2200    '                                                  'proper place
2210 NEXT LP
2220 '
2230 RETURN
```

(5) EXAMPLES CONTINUED

```
/* MAIN - linked-list handling
** ****************************** history
**  11/02/83    C. Stassen
**  11/05/83    C. Stassen    Changed struct record from LONG to INT
**
*********************************
*/
#include "STDIO.H"
#define MAXLIST 20

main()
{
    struct {
             int  val;
             int *next;
           }  *listptr;

    int  *lstbgn;                       /* the beginning of the list */
    char *alloc();                      /* a routine which gets memory */
    int  lp;

    lstbgn = alloc ( sizeof (*listptr) );

    listptr = lstbgn;

    for ( lp=0 ; lp<MAXLIST ; ++lp )
    {
        listptr->next = alloc ( sizeof (*listptr) ); /* set up each list   */
        listptr->val  = lp*lp;                       /* element as the     */
        listptr = listptr->next;                     /* square of its rank */
    }

    listptr = NULL;

    for ( lp=0 ; lp<MAXLIST*MAXLIST ; ++lp )
    {
        listptr = find ( lstbgn , lp );           /* does the element exist */

        if ( listptr==NULL ) printf ( "." );      /* print "." for every */
        else                 printf ( "%d\n",lp );/* element we don't find */
    }
}
```

```
/* FIND takes:    listptr - a pointer to a linked list of integers
**                value   - an integer we are trying to find in the list
**
** FIND gives a pointer to the linked-list element equal to "value", or
**             null if no such value exists.
*/

find( listptr , value )

struct {                                /* linked-list element is a structure */
          int  val;
          int *next;
       } *listptr;
int  value;

{
    if ( listptr == NULL ) return ( NULL );  /* exit immediately if no list */

    while ( listptr->next != NULL )
    {
        if ( listptr->val == value ) break;    /* exit loop if we have found */
                                               /* the element, otherwise     */
        listptr = listptr->next;               /* continue until the end of  */
    }                                          /* the linked list.           */

    return ( listptr );
}
```

VII: Use of CITNET

15 January 1984
G. Fox, N. Wilson

VII.1 Introduction

All the IBM PC's are connected to the campus ethernet through a 9600 baud RS232C line. To use PC as a terminal emulator, consult IBM3101 emulator manual. Execute

```
cd \IBM3101
terminal        (This assumes setup has been run)
```

Now one is ready to communicate with network as described below. Note that Alt-F10 communicates with IBM-PC and allows you to quit program, transfer files etc. Note emulator does not recognize directories and you must copy any files to directory \IBM3101 before transmitting. Again, they will be saved on this directory if "S" command executed to IBM-PC. See Sections VII.8 and VII.9 for more details on use of IBM-PC as a terminal emulator. The following manual is based on one produced for High Energy Physics (HEP) by Norman Wilson.

Note that IBM3101 emulator is not very clever with control-S and control-Q due to internal buffering.

VII.2 Making Connections

If your terminal is hooked to the network, the first thing you'll see will probably be a greeting from the NIU:

You may now enter Net/One commands

>

You will need to type control-V to obtain this prompt on machines in the physics lab. If you don't see this prompt, hit a return to get it. If you still see nothing, see the section below on problems

If you call up a modem connected to the network, it should greet you spontaneously. If it doesn't, hit a return, which should give you at least a >.

Once you're being prompted with >, you're allowed to enter various mode-setting commands, or to ask for a connection. Only connection-making will be covered here; see the U-B manual for other info. (This is Net/One User's Guide available from the bookstore).

To hook yourself to a machine called, say, *hippo*, give the command **c *hippo**. You should see something like this:

```
>c *hippo
connecting ... 2351a3 success
```

Some machines will give you a login request, others need an extra return at this stage. The 'success' message means that all went well. You're now talking to *hippo* almost as if your terminal is directly attached. (See below for an explanation of 'almost'.) If all didn't go well, the NIU will print an error message and type another >. The most likely errors are: 'Resource not available' or 'All ports busy,' which mean that there aren't any free lines to *hippo* and you'll have to wait (if this happens frequently, perhaps *hippo* needs more lines; talk to the powers that be), and 'No response for specified name', which means there's no machine named *hippo* anywhere on the network.

Note that you have to put an asterisk '*' in front of the machine name. This isn't actually always needed, but never hurts The conditions under which it's mandatory are too complex to explain here.

VII.3 Breaking connections

When you've finished dealing with the computer you requested, and have logged off it, you'll often still be connected (You can even log back in if you so desire). To break the connection, you should type the 'disconnect sequence': the two characters control-E, carriage return This should produce the 'You may now enter Net/One commands' message and the > prompt again.

Sometimes, the network connection will be broken automatically when you log out. If this is the case, the 'Net/One' banner will appear immediately.

When you're completely through with the NIU for the moment, you can type the 'quit' command **q** to tell it so The result will look like:

```
>q
IDLE
```

This isn't very important for lines hooked to terminals If you leave the NIU sitting in command mode for several minutes, it will IDLE itself

If you've dialled up one of the modems hooked to the network, the NIU will refuse to talk to you once it's become IDLE (unless you hang up and dial it back). If you hang up (or carrier drops for some other reason) while you're connected, the connection is automatically broken

Breaking a connection with the disconnect sequence is *not* guaranteed to log you out if necessary; it depends on how the computer in question is set up In general, assume it won't, log out before breaking the connection unless it's impossible to do so (because the system crashed, for example).

Please remember to log out (and to disconnect if necessary) when you're finished, so the port you're using is free for someone else. This is, after all, part of the point of the whole scheme!

VII.4 Caveats

Once the NIU has connected you to a machine somewhere, you can almost pretend that your terminal is hooked up directly and that the NIU is merely the product of a deranged imagination. There are a few differences, however

Depending on how the ports at the computer end have been set up, the 'break' key might or might not function properly. Most systems don't need it for anything, so this shouldn't be a problem.

A more serious problem is that there are special characters which are not passed transparently by the NIU. Control-S causes output to be frozen, control-Q allows it to resume. They are *not* passed through to the computer at the far end of the connection, but are handled locally by the NIU. However, if the computer tries to send characters when the NIU doesn't want to accept them (perhaps because the terminal is running at a slower speed than the computer is sending, or because the user has typed control-S himself), the NIU will send it a control-S of its own volition, and a control-Q when it's ready to accept characters again. Since most operating systems these days interpret control-S and control-Q in exactly this way, this will normally not be a problem. However, if you use a program which uses special terminal modes, be aware that it may receive spurious ^S/^Q characters. In particular, don't try to use a screen editor which uses these characters as commands over the network, the results may be catastrophic.

Two other characters are treated specially by the NIU. Control-E is the beginning of the 'disconnect sequence' used to break a connection. If you want to transmit control-E, precede it with control-P. Control-P is a quoting character which undoes the special meaning of the other special characters, including

itself; to send a literal control-P, type it twice.

These restrictions on characters sent from the terminal to the computer don't always apply to characters coming the other way; it depends on how the port at the computer end is configured. It's safest to assume that they do apply, however, and that you can't, for example, send control-E to your terminal. Connections to the UNIX machines within HEP are set up to allow it, but machines elsewhere on campus may not choose to do so (as the VMS machines here have not).

It's likely that some of the specialness of control-S and control-Q will go away eventually for the UNIX VAXes, but not for other machines. Control-E and control-P will probably always be magic characters.

VII.5 Problems

Some suggestions for when the network seems to be broken:

Did you type control-V?

If your terminal seems dead, check the obvious non-network things: has the cable fallen off the back? Are you running a program on the machine to which you've connected which never says anything? If the network seems a likely cause, the first thing to try is typing control-Q, in case the NIU thinks output is frozen. If the line still appears dead, try the disconnect sequence (control-E, return). This should give you the **Net/One** banner again. Check the parity settings on your terminal (see below). If it still won't respond, consult a guru.

If the NIU will talk to you, but won't let you connect to the computer of your desire, check the obvious first: are you spelling the machine's name correctly? Is that machine perhaps down at the moment? An occasional weirdness that can show up is that the NIU will make the connection, but the far end will seem dead, and after a few seconds the NIU will gratuitously disconnect you. This usually means either that the computer in question isn't working, or that a cable has been knocked loose from one of its ports.

If the remote machine crashes, or some other disaster occurs, you can always use the disconnect sequence (control-E, return) to get back to the NIU.

If the NIU behaves strangely, rejecting apparently valid commands, ignoring control-S or control-Q, or refusing to accept the disconnect sequence, check the parity setting on your terminal. It should be set to produce no parity (or space parity). The NIU will pass eight-bit data through a connection, but will recognize commands and special characters only if the parity bit is off.

VII.6 Machine names

Names of useful computers on the network:

***zebra** General Automation Zebra 68000 (Logon phylab, Password phylab)

***frodo** Data General MV4000 (Logon phylab, password phylab)

VII.7 Mail

One can send CITNET mail between some machines on the network. The use on UNIX machines is

Mail machine # person

where machine is

cithep	HEP UNIX VAX11/780
nncpb	HEP UNIX VAX11/750
timevx	Institute "Time" VAX

cithep # gcf will reach G. C. Fox

This only works for machines with Ungermann-Bass parallel interfaces.

Note the UNIX-UNIX intermachine mail system (which uses uucp) uses ! instead of #.

VII.8 Use of IBM PC as a terminal emulator

The following dialog shows the setup procedure for the IBM-PC with the IBM3101 Program.

Line Speed (Baud Rate) to be used? [9600]:
Block Mode? (Y=Block N=Character) [N]:
Half-Duplex? (Y=Half-Duplex N=Full-Duplex) [N]:
Parity? (1=Odd 2=Even 3=Mark 4=Space) [4]:
Stop Bits? (1 or 2) [1]:
Automatic New Line? (Y=Yes N=No) [Y]:
Automatic Line Feed? (Y=Yes N=No) [N].
Carriage Return? (Y=CR N=CR-LF) [Y]:
Character Sent at End of Message? (1=ETX 2=CR 3=EOT 4=XOFF) [2]:
Scrolling? (Y=Yes N=No) [Y]:
Prompt Character from Host? (0=none) [0].
START/STOP (XON/XOFF) enabled? (Y=Yes N=No) [Y].

The speed (9600) is appropriate for physics lab set up. It would be 300 or 1200 if connecting through a modem Note that important response (space or 4) to parity request. The reason for this is explained in VII.5.

VII.9 File Transfer

Suppose we wish to transfer the file lobster on a PC to the *zebra* Then

1) Copy *lobster* to the \IBM3101 directory
2) Logon to *zebra* using IBM3101 emulator
3) Start an editing session on *zebra* :
   e.g. ed lobster_zebra
   a (append command to UNIX ed)
4) Transfer a File:
   ALT-F10 (transfer control to PC)
   T (transmit command)
   LOBSTER (file to transfer from PC)
   . (end append on zebra)
   w (write file on zebra)

Suppose we wish to transfer the file *herring* from the *zebra* to the PC

1) Logon to *zebra*
2) Type UNIX command to list file on screen
   cat herring (**no ENTER yet**)
3) Arrange to save file on PC
   ALT-F10
   S

HERRING followed by ENTER

4) Complete command in 2) by:

ENTER

5) When file is completely transferred

ALT-F10

E (end S command)

6) Edit the stray lines in *herring* on PC. Probably there will be an unwanted prompt as last line in file.

If you just wanted to print a file on *Zebra* on IBM-PC printer, replace HERRING in 3) by LPT1. Be sure to do Step 5) - printer will not get its last buffer otherwise.

VIII: Use of Zebra MC68000 XENIX Computer

15 January 1984
G. Fox, N Wilson

VIII.1 Introduction

The Zebra is a MC68000-based system running XENIX, which is roughly V7 UNIX. It has a megabyte of physical memory. There is no support for virtual memory; hence the biggest process one can run is somewhat less than a megabyte. After system overhead, there are about 80000 blocks free on the user disk.

Anyone who has a login on the UNIX VAX automatically has one on the Zebra; the password files on the two machines are identical. (In fact, the password file from the VAX is copied over periodically; don't change your password on the Zebra and expect it to stay different from the VAX.)

Access to the Zebra's terminals is through the network; nodename 'zebra'. See the network writeup (Section VII) for more about using the network.

There are no printers on the Zebra at the moment; files can be transferred to the UNIX VAX via uucp or tape and printed there. Eventually this will be automated in some way.

Please don't use the Zebra's console as a hardcopy terminal, and DO NOT walk away with paper from it. Important system messages are printed there; especially since the machine is here to be tested, it's important not to lose them.

Files can be transferred by tape (there's a 1600 bpi tape drive), or by uucp. See the UNIX manual for the latter. Remember that uucp will only be able to create files in directories which have universal write permissions; note also that there's a bug in the Zebra's uucp which prevents existing files from being overwritten, regardless of their permissions. The Zebra's uucp nodename is also 'zebra.'

Mail between the Zebra and the UNIX VAX (cithep) also works; this can be used as a poor man's non-binary file transfer (by mailing a file to yourself on the other machine).

Both mail and file copy are primed from the Zebra: the Zebra will call the VAX whenever there's uucp work, and once an hour in any case, but the VAX will not call the Zebra (but will wait for the Zebra to call it).

You almost certainly do not want to transfer files of binary data; the 68000 uses different byte ordering for integers and floating point numbers, so binary data written on the VAX has to be converted before it can be read sensibly on the Zebra.

Miscellany: No accounting of any kind is being done at the moment. Nonetheless, please be careful not to hog resources too much. Disk space is especially likely to run out.

Backups are being done only sporadically; pretend they aren't done at all, and back up any crucial files yourself.

User directories live in '/usr/u', not '/u1'.

The default erase, kill, and interrupt characters are #, @, and delete. There's no easy way to change this on a system-wide basis; however, you can set them to something else in your .profile if you wish. ('stty intrc "^c" erase "^h" kill "^u"' will change to the defaults on the VAXes). To see what these

characters are, list the .profile file for given login.

Note that the zebra has curious problem of not giving "Login" message when accessed through network. Just type your login, when you get connected.

VIII.2 uucp

One can use *uucp* to transfer between zebra and other UNIX machines.

If *gcf's* home directory has defined on it a directory *uucphold* (which is always with *uucp* must be writeable to world), then the shell script:

```
for i in $*
do
uucp $i machine!~gcf/uucphold/$i
done
```

will transfer all files names in its arguments to given machine (*machine* could be *cithep* or *zebra*).

VII.3 Phylab notes

Files referred to in lessons exist on */usr/u/gcf* - directories *basic*, balls and *pot*

General access is login phylab - password phylab

GCF has found a bug in C compiler using floats - doubles work OK as I know.

IX Use of the Basic Compiler

Usually one uses BASIC in interpretative mode which is best for debugging. Many programs can be compiled and you will find that they will run about ten times faster. The BASIC compiler is older than the interpreter and does not support some of the newer features. The BASIC **interpreter** manual indicates for each command if it works for the compiler.

To use compiler, save your BASIC file in .ascii (,A option) mode. Then copy to your directory, the files

\bascom\bascom.com
\bascom\basrun.lib
\bascom\basrun.exe

Then we compile the program HIPPO.BAS as follows:

BASCOM reply HIPPO to first question "enter" to all others.

LINK reply HIPPO to first question "enter" to all others

Typing HIPPO will now run the compiled program.

Further details maybe found in the BASIC COMPILER Manual.